PCR: APPENDIX-I; PART-I

RAVEPC: Remotely Accessible Visualizer & Explorer of Point Clouds: An Interactive Visualization Application for LiDAR Data

A Report for the Project titled,
"LAN-based Interactive Visualization of Three-dimensional LiDAR Point Cloud Data",
on the work conducted at IIIT Bangalore,
in August 2013 - February 2015,
funded by NRDMS (DST, GoI).

by

Beena Kumari, Avijit Ashe, Dr. Jaya Sreevalsan Nair [*],
Dr. Kiruba Bhagirathi, and Pavithra Rajendran,

Graphics-Visualization-Computing Lab
Center for Data Sciences, IIIT Bangalore.
`http://cds.iiitb.ac.in/gvcl`
[*] Project PI; e-mail: jnair@iiitb.ac.in

International Institute of Information Technology, Bangalore.
26/C Electronics City, Hosur Road, Bangalore 560100
March 2015

This document is Part I of 2-part series of the report of our project, "LAN-based Interactive Visualization of Three-dimensional LiDAR Point Cloud", submitted to Department of Science and Technology (DST), Government of India. Part I comprises the introduction, background work, literature survey, and our contributions to feature detection and extraction. Part II includes our contributions to feature tracking and remote visualization system.

Acknowledgements

The authors would like to thank the Department of Science and Technology, Government of India, for funding this project under the NRDMS (Natural Resources Data Management System) Programme. The authors would specially like to thank Dr. P. S. Acharya (DST), and Prof. N. L. Sarda (IIT Bombay), from the expert committee, who have supported the project in various ways. Special mention of gratitude to Prof. S. Rajagopalan at IIIT Bangalore for his valuable advice on managing the project. Last but not the least, the project reached its completion owing to the encouragement and help of the administrative staff, faculty, and students at IIIT Bangalore.

Table of Contents

List of Figures

List of Tables

Preface

(Salient excerpts from the project proposal)

Project title: "LAN-based Interactive Three-dimensional Visualization of LiDAR Data."
Project objectives: The following are the objectives of our proposal:

- To explore LiDAR for obtaining interesting datasets on 3D campus GIS in order to develop user interactive visualizations and if required, processing them to obtain point datasets.
- To explore LiDAR for obtaining time series datasets on 3D campus GIS in order to develop temporal feature tracking algorithms to enable user interactive visualization and if required, processing them to obtain point datasets.
- To develop various multi-resolution algorithms for performing scalar field topology based analysis on the point datasets and compare with those specified in related work.
- To develop algorithms for temporal feature tracking by used scalar field topology based analysis.
- To implement a visualization system for a server-client architecture for remote visualization, enabling (a) a server with high end graphics card and computational capability to perform data processing and final image computation, (b) one or more clients, which are thin interfaces, functioning as display devices as well as user interaction receiver, and (c) the network between server and client(s) to transport the user feedback and final image data.

Approaches/ methodologies for the work plan: Our approach to implementing the work plan is driven by an end product which is to be deployed on local area network (LAN) of an organization willing to share their LiDAR datasets.

- We will identify such organizations with a need to analyze LiDAR datasets as well as for collaborative analysis across a LAN.
- All algorithms for topological analysis as well as for developing the remote visualization will be implemented as desktop applications.

End-of-project status: Our implementation of the remote visualization tool will include a equal mix of algorithms in computational geometry and visualization, and GPU virtualization technologies.
Suggestions for replicability of the research outcomes: The algorithms, modeling methodologies, and tools that we develop and implement will be made available to both the funding agencies as well as other interested researchers in the community. We will be publishing our findings in leading conferences and journals in related areas.
Suggested plan of action for utilization of expected outputs from the project: The following is our plan of action for utilization of expected outputs from the project:

- We will be hosting our application at an appropriate organization which will have a requirement to analyze and explore LiDAR datasets.
- We will be sharing our source code with researchers in similar areas of interest.
- We will be using the results of our research and implementation to publish at top conferences.
- We will publish at our website an extensive documentation for using our visualization tool.

Publications from the project:
We are in the process of publishing our work in several top tier conferences. We have the following publications so far from this work:

- Kumari, B., and Sreevalsan-Nair, J.,An Interactive Visual Analytic Tool for Semantic Classification of 3D Urban LiDAR Point Cloud (to appear) in Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, Seattle, November 2015.
- Kumari, B., Ashe, A., and Sreevalsan-Nair, J. (2014). "Remote Interactive Visualization of Parallel Implementation of Structural Feature Extraction of Three-dimensional Lidar Point Cloud," In Big Data Analytics (pp. 129-132). Springer International Publishing, December 2014.
- Kumari, B., and Sreevalsan-Nair, J. (2013). "Three-dimensional Visualization of LiDAR Point Cloud Using Structural Feature Extraction," in Proceedings of NSDI (National Spatial Data Infrastructure) 2013 and Poster presentation, IIT Bombay, November 2013.

Chapter 1

Introduction

LiDAR, popularly known as an acronym for Light Detection And Ranging, is actually a portmanteau of the words 'Light' and 'Radar' but as the word 'radar' was already an acronym for Radio Detection And Ranging, people in the scientific community thought it appropriate to label it as an acronym instead. This technology has been around since 1960 when laser was introduced, radar had been used for remote sensing purposes and it was a wise idea of combining these two technical innovations to form another method of remote sensing much more accurate and precise called LiDAR.

Currently, LiDAR is being extensively used in exploration and studying topographic data through the use of airborne altimetry, which is one of the three main type of LiDAR collection techniques, as of now. LiDAR is often available in the form of dense or sparse but a huge collection of unstructured data-set of 3D points over a three dimensional space, where every point has its own coordinates with respect to a given local frame or the world frame of reference. These dense clouds can accurately collect information about even complex structures on the terrain like trees, buildings, coastal belts, mountains etc. at high resolution. As a result, multi-resolution and multi-scale representation of the data is made possible which allows a three-dimensional perspective to studying various salient features associated with them. This further allows the user to decide the amount of information to be included in the final display. Our project revolves around attaining these preliminary objectives.

1.1 Motivation

Since LiDAR are large scale dataset of the order of gigabytes, the processing of LiDAR data become expensive both in terms of time and power for real-time applications. Thus, there is a severe need of cost effective solutions. Secondly, LiDAR datasets can often contain billions of points and most of them may be redundant. We aim to reduce the complexity of data-set by extracting important points, i.e. features without losing the salient information of the data.

Another important requirement is to design a visualization system that not only can be easily deployed into existing infrastructure of any organization but also leverage the advantages of modern technical advancements. We have proposed a server-client based architecture for our visualization system where server will perform all heavy-duty task and client will use a thin display for transmission of user feedbacks over the network and final display. Therefore, Server should have the high computational capability with dedicated hardwares.

1.2 Project Objectives

The project overall aims to perform three implementations and thus has been segregated thinly into three sub-objectives. There are two algorithmic objectives and one pragmatic objective.

1.2.1 Input of Interest

Essentially there exists three kinds of LiDAR data; airborne, bathymetric and terrestrial. Our focus is strictly on acquiring dense *airborne LiDAR scans* of institutional 3D GIS data such as scan of a college campus, and thereby use it for our proposed objectives. This could be extracted from raw LiDAR data or downloaded directly in preprocessed and cleaned form from various available open and closed commercial repositories. Thus, we deal with point data throughout our project and any reference to LiDAR data, point cloud data or PCD would essentially mean the same.

1.2.2 Feature Detection

This is supposedly the first and foremost objective in this project. Point cloud datasets can be huge but they can be efficiently reduced by removing redundant points and outliers or background noise. Therefore a point cloud reduction algorithm is required as a initial step in this process.

Our focus will be in finding salient features to identify building structures and other structural components in a campus environment or like in a city. These schemes will enable us to model the data appropriately for rendering and display purposes.

1.2.3 Feature Tracking

This is the second but a challenging goal to be achieved. A lot of study has been made in this context earlier but there is still some lack. Time-series or temporal feature tracking is difficult to implement as we need to locate features pertaining to a point dataset that remains persistent over time and track the changes brought about in the original data over the passage of time.

1.2.4 Server-Client Architecture

Apart from our algorithmic objectives, our pragmatic objective is to deploy such a system on to a local area network (LAN). We will be using VirtualGL as a means of running our system in a server-client architecture. Our main aim is to have a server with high computational capability and a high-end graphics card which will be remotely implementing the computations for preprocessing, data modeling and data processing. The clients are essentially thin clients which will need high-end display capability which will rendering the image. The network connection between the server and the client will transport user interactions to the server and final image data to the client.

Summary

The following are the objectives of our project:

- To implement a visualization system for a server-client architecture for remote visualization, enabling
    1 a server with high end graphics card and computational capability to perform data processing and final image computation
    2 one or more clients, which are thin interfaces, functioning as display devices as well as user interaction receiver
    3 the network between server and client(s) to transport the user feedback and final image data.
- To develop a multi-resolution algorithm for performing scalar field topology based analysis on the point datasets and compare with those specified in related work.
- Temporal feature tracking by using scalar field topology based analysis

1.3 Proposed Approach

We specifically work with point clouds extracted from LiDAR datasets because point-based approaches work directly on sample points without having to compute the connectivity between the points and generating meshes [15]. Meshes are more heavyweight data structures compared to points, and since we are dealing with sizes of dataset of the order of gigabytes, we aim to reduce the complexity and thus the size of the processed data to the lowest possible extent, without losing the salient features of the data. LiDAR datasets being measurement data are raw and require proper preprocessing and filtering to remove noise, which largely includes instrumentation errors. This clean-up exercise is highly essential for revealing scientifically relevant structural information.

Topological feature based methods for exploration of point datasets has the additional advantage of isolating features of interests which can be tracked over time. Henceforth, our objective for this project is to explore techniques of feature tracking of LiDAR data using topology based schemes.

Since LiDAR is a large scale data-set and lots of complex mathematical operation involves in feature extraction techniques, we have exploited the power of GPGPU to make our application user interactive. The GPGPU has potential of providing parallel support for computation and visualization. Parallelization using GPGPU on NVIDIA's CUDA enables scalability.

Implementing the idea over a network requires making a survey of already available remote connectivity tools. Our project requires extensive ability to run graphic intensive applications off the server without negatively affecting the speed, user-interactivity and visualization. Also, the proposed remote visualization application should support VirtualGL natively along with the above stated requirements.

1.4 Report Structure

We explain more about LiDAR, its different types, methods of collection etc. in chapter 2. In the chapter 3, we have done a literature survey on existing algorithms for preprocessing of LiDAR data and extractions of features such as lines, corners, edges, etc. In chapter 4 we describe in detail about the implementation of the first and foremost objective, feature detection and extraction. Following that we describe the second objective, that is time-invariant feature tracking. In chapter 6 we discuss the remote visualization capability introduced in our application and about ThinLinc. It discusses in detail about its installation, accessing server from desktop and browser based clients and other details. The next chapter will tell you about the system architecture, how RAVEPC has been built, the various components etc. elaborately. In the final chapter 8, we discuss the results and future work.

Chapter 2

Airborne LiDAR Technology

Although this technology has been there for such a long time, it was made public or to be more specific attained public awareness only after Apollo 15 mission in 1971 when the astronauts used laser altimeter to survey and map the surface of the moon. Eventually as it gained popularity, it got wide acceptance as a technology that uses light of different wavelengths for detecting distant objects. This particular event was also a motivation for the wider acceptance of the acronym, somewhere around 1970s, based on the earlier presented facts. Although 'RADAR' has been replaced with un-capitalized form 'radar' making it seem no more an acronym but a generic word, however, LiDAR is still used the same way.

2.1 Methods of Remote Sensing

LiDAR system can be classified into airborne and ground based lidar system based on their collection method. In the airborne lidar system, it can be further classified into topographic and bathymetric as shown in the Figure **??**. We have used topographic lidar data of urban area.

LiDAR is a remote sensing technology which collects geo-referenced information of target object by illuminating the object with laser light. Based on

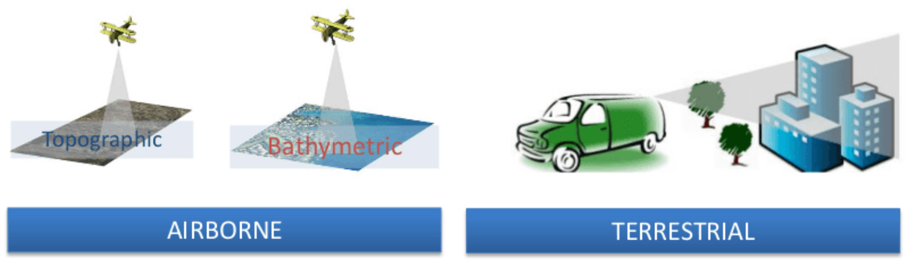


Figure 2.1: Methods of LiDAR Data Collection

the source of energy used for collecting the geo-reference information of objects, remote sensing techniques can be broadly classified into two categories:

- Active Systems
- Passive Systems

Active remote sensing systems make use of devices which not only detect electromagnetic waves but also generate these waves themselves. In other words they act as the source of light beams used for scanning objects and have the detection sensors to detect the reflecting waves too. The passive systems on the other hand only have half of the work of detecting the incoming waves from a natural source like sun, radiations from within the earth, etc.

2.2 LiDAR Technology

It can be defined as a technology that has the ability to produce precise and directly geo-referenced spatial positions of structures on the surface of earth. It is a method that uses direct pulses of intense, focused laser light either in the visible or invisible range to detect objects on its way by the help of reflection and scattering. Mostly the range is near-infrared but sometimes it also uses band in the green color for bathymetric measurements. Bathymetric measurements are those which are used to map shoreline areas and use LiDAR units mounted on boats to scan the shoreline and its periphery and sometimes even the sea bed in shallow water regions.

A laser scanner and generation unit repetitively sends focused light beams towards a target object, collects the reflected beams back by the help of light detecting sensors, and determine information like distance, elevation, etc., depending upon the time taken by reflected pulse to return. Figure 2.2 shows the diagram for collection of topographic data using airborne lidar system. As it makes use of laser beams, it can focus onto very fine and precise points and therefore is highly useful in creating high resolution maps of terrains, shoreline areas, shallow-water beds, forested terrains and much more. The major advantage is its usage in forested areas where the land beneath the vegetation is hardly penetrable through the voids within the leaves and branches. The fine and focused light beams can manage to pass through the very small spaces available within the dense vegetation. As it's frequency is high, it is able to collect enough points to classify the terrain beneath the vegetation. LiDAR is usually flown at night when the sky is relatively clearer than the day time and also the air traffic is less. The advantages in comparison to the older techniques are in accuracy(a very important aspect), density of data and advanced technology.

2.3 LiDAR Data Collection

LiDAR provides an efficient means to collect very dense and precise data over large area in a short time span. It is similar to a radar except that it uses laser pulses instead of radio waves. The data is usually gathered by flowing airborne vehicles like small charters, helicopters, planes etc. over the whole area. The point data can also be collected from ground based stations or mobile units like in bathymetric measurements as previously mentioned.

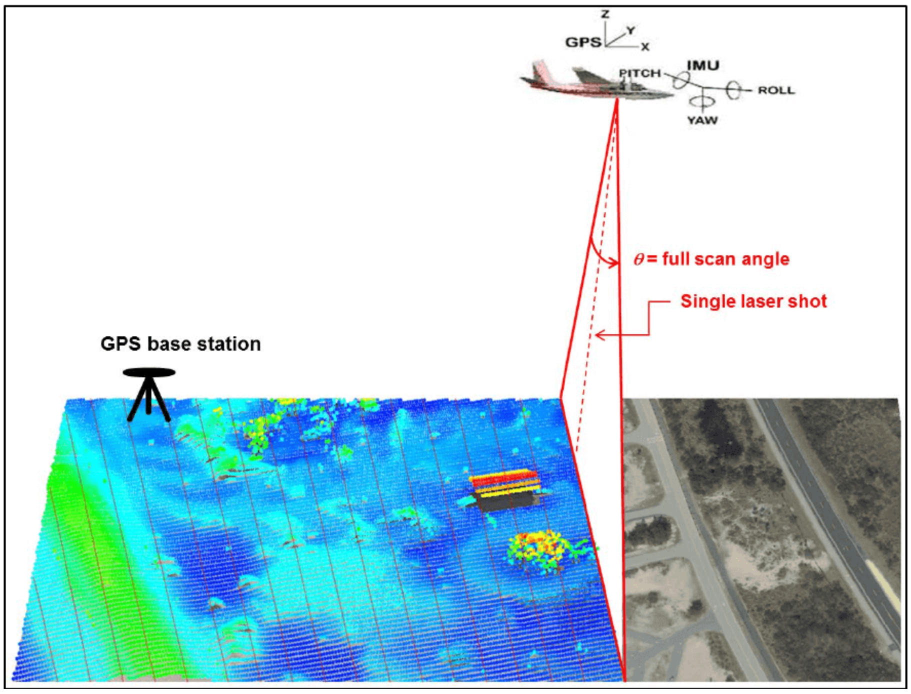


Figure 2.2: Topographic Airborne LiDAR System [28]

2.4 LiDAR Detection

Usually there exists two types of LiDAR detection methods, one is called incoherent or direct measurement and the other one coherent or indirect method. In incoherent method of LiDAR mapping, the energy of the light is directly measured to help us determine the distance of the objects from the source. This is actually an amplitude measurement principle. The other method is called coherent detection that is usually adopted for doppler and phase sensitive measurements. This process makes use of coherent heterodyne detector which has an added advantage over the incoherent detection. It allows the instrument to operate at much lower power for the same level of detection but at the expense of a complex transceiver used for detection.

Both of these methods can be further classified into two more types depending upon the manner in which laser light is created and used for focusing on the objects. They are micro-pulse systems and high energy systems. Micro-pulse system for laser creation uses much less power to create laser beams and have been developed as a result of advances in the laser technology and increasing computing power within reducing form factors and lowering price tags. As they are used at low power or low energy which is in the order of micro-joules, it is also safer than the other method. They are eye-safe to be specific and may be used for practical applications by common people without any safety precautions. In contrast to these low power systems high power systems, although are not so safe, are being used for atmospheric research like air and cloud parameter measurements, weather details like temperature, humidity, elevation, concentration of gases like carbon dioxide, methane, ozone, etc.

2.5 Advantages

LiDAR is used heavily for exploration of topographic data, which is procured through high speed airborne altimetry. As stated in [19]:

> “This technology offers several advantages over the conventional methods of topographic data collection viz. higher density, higher accuracy, less time for data collection and processing, mostly automatic system, weather and light independence, minimum ground control required, and data being available in digital format right at beginning”.

Owing to these advantages, this technique is popularly used for flood-modeling and similar applications such as, bathymetry, geomorphology, glacier modeling, etc. LiDAR technology provides datasets which are typically very large, containing diverse and highly complex objects. We focus on using specific algorithms to extract features from the LiDAR data which has been processed to a point cloud format.

2.6 Airborne LiDAR and Building Dataset

We have implemented our algorithms to urban airborne lidar data. We are mostly interested in 3D GIS dataset that include urban scenarios consisting of streets, buildings, community areas like stadiums and sports complex, flyovers, shops, malls, traffic and vehicles on road and other natural structures as well that might exist in such surroundings. One such important and easily available dataset can be college campus 3D LiDAR scans. Thus, acquiring these datasets is the primary goal that leads us to reviewing the literature on LiDAR technology.

Chapter 3

Literature Survey

LiDAR is an airborne technique known for collecting vast amount of information in the form of 3D point cloud data which are useful for extracting important spatial features. But in order to efficiently extract these features, the large amount of data has to be processed such that no feature information is lost in the process. Our work contributes toward analysis of various techniques and methods that had been carried out so far to reduce the point cloud data. The remaining portion of this chapter comprises of the following. Section 3.1 explains the work of Pauly et. al. [24] for extraction of important features lines using a multi scale approach. Section 3.2 explains the recent work by Keller et al. [16] which extracts important structural features using the multi-scale approach. The remaining sections look into the alternative methods that can be used to reduce the point cloud dataset such that the output obtained is in par or better than Keller et al. [16] work and how parallelization can be useful for reducing the computational cost and time.

3.1 Multi Scale Feature Extraction

Pauly et al. [24] describes the method where an unstructured point cloud data is classified into line type features by using principal component analysis based on local structure of neighborhoods and a minimum spanning graph containing important line-type features is formed. The feature extraction step comprises of finding the weight of each point and classifying the point to a particular feature according to some given threshold. This is done using the multi-scale surface variation $\sigma_n(p)$ as shown in Eq 3.1 where λ_i are eigenvalues of a $3x3$ covariance matrix C of any local neighborhood N_p for point $p \in P$.

$$\sigma_n(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{3.1}$$

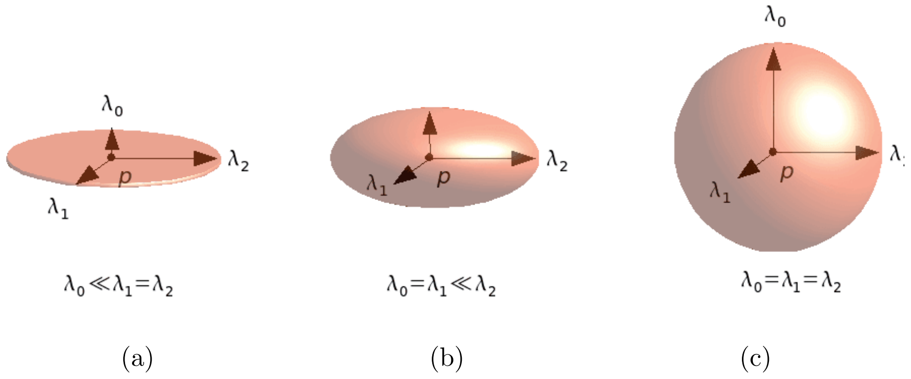
The next step follows with finding points with high probability form the feature nodes belonging to each of the feature described and a minimum spanning graph is constructed and this graph is modeled using snakes to make the feature lines smooth. The main advantage of such multi-scale approach is its robustness towards noisy datasets, extraction of coarse features despite the low curvature and provision of additional information such as width of these feature lines.

3.2 Multi Scale Structural Feature Extraction of LiDAR Data-sets

Keller et al. [16] uses a multi scale based approach for extraction and visualization of structural data by finding the geometric property of local point neighborhoods, thus making it useful in terms of geographical data exploration. Unlike previous works, this paper identifies regions based on special structural features such as creases, borders, ridge-lines etc. and represents them in the form of a feature graph. The method can be divided into four major steps which are explained in detail in the following subsections.

3.2.1 Preprocessing of 3D point cloud data

In this step, the LiDAR dataset known for its collection of billions of points gets arranged in an octree-based voxel structure for improvising the computational speed. Outliers removal process [32] is carried out where all the points for which the mean distance from their respective $k-$neighborhood is above a fixed threshold get removed. Once all the unnecessary points are removed, they are classified stochastically by dividing the points into subsets and determining the type of these points based on the shape of its local point neighborhood $N_r(p)$ where r is the radius of a sphere with center p. The points are classified into three main types as shown in Figure 4.2 namely sphere-like, cylindrical-like and disc-like neighborhood based on the eigenvalues λ_1, λ_2 and λ_3 obtained from the covariance matrix of $N_r(p)$. Any point that is not a part of a curve or a surface is considered as a critical point. The root mean square curvature approach is used for determining the feature properties from surface-related structures and it is observed that it produce best results in comparison with other curvature methods.



λ_0
λ_2
p
λ_1
$\lambda_0 \ll \lambda_1 = \lambda_2$
(a)
λ_2
λ_1
p
$\lambda_0 = \lambda_1 \ll \lambda_2$
(b)
λ_0
p
λ_1
$\lambda_0 = \lambda_1 = \lambda_2$
(c)

Figure 3.1: Stochastic method used for point classification based on shape of local neighborhood, as proposed by Keller et al. [15, 14], namely, (a) planar or disc-like, (b) cylindrical, and (c) spherical neighborhoods [14].

3.2.2 Post-processing

In the post-processing step, feature values are filtered to eliminate the noise. Weighted average filter is used to remove the noise on the local neighborhood $N_r(p)$.

3.2.3 Feature graph generation

The feature graph is generated using the approaches in [8, 24, 29] such that initial seed nodes are identified and critical nodes are connected together. The seed nodes are chosen such that they satisfy the minimal probability type condition among other nodes of the same type. Two graphs has been constructed, one for line-type feature and other for surface-type feature points, based on the curvature and eigen related feature strength using the likelihood parameter of these seed nodes. Further, instead of using minimum spanning tree as in Pauly et al. [24], edge propagation method using principal component analysis algorithm over the local neighborhood is used for connecting the nodes. Lastly, the critical nodes are connected by clustering the neighboring critical nodes and eliminating unwanted critical nodes. The graph is pruned and simplified by removing unwanted loops and branches and finally smoothened.

3.3 Alternative Methods: Point Cloud Reduction

3.3.1 Octree based method

Wang and Tseng [31] suggested an octree based split- merge-intersect method to organize the 3D point cloud data and extract important features. The first step is the split process in which the entire point cloud data is considered as the root of an octree structure. This data space will be split further into sub-spaces if the data present does not satisfy the plane fitting test. Each of these sub-spaces are again considered for the plane fitting test and split if needed and this process continues until all the sub-spaces contain data set such that they satisfy the plane fitting test. Automatically, the point cloud data gets sorted due to octree structure.

The plane fitting test uses the least squares estimation technique to check if each of the sub space is a best fit plane. For a 3D plane in the 3D Euclid space(Eq 3.2), the three parameters A, B, C represent the normal vectors of the plane such that at least one of them will always be a non zero value. For instance, if $A = 0$ then we can find a k such that $kA = 1$. The plane equation can be reformulated as in Eq 3.3.

$$Ax + By + Cx + D = 0 \tag{3.2}$$

$$ax + by + cz + d = 0 \tag{3.3}$$

where $b = kB$, $c = kC$, $d = kD$.

But we need to find which normal vector among a, b, c is non zero value for setting the constraint. Firstly, we need to find six points $Pxmin$, $Pxmax$,

$Pymin$, $Pymax$, $Pzmin$, $Pzmax$ such that they denote the maximum and minimum coordinate values in X, Y, Z axis respectively. The distances $(Pxmax, Pxmin)$, $(Pymax, Pymin)$ and $(Pzmax, Pzmin)$ are compared and the two points with the longest distance are chosen. A third point with the longest distance from the above chosen points is considered and the constraint parameter is chosen accordingly by solving the plane parameters. The data points are split further if the distance residual is above the given threshold. The point clouds may not be distributed evenly on the fitting plane and to avoid this problem, we find the planes with respect to the point cloud data set by finding the corresponding distribution density. If the density is above a given threshold, then the point cloud data set is split further.

TIN(Triangulation Irregular Network) is constructed and a neighboring patches relation table is created which contains information about each plane formed in the splitting process which are considered as patches. The octree can be represented as a group of such patches and each node in the octree consists of three types of neighboring nodes as shown in Figure 3.2. The octree structure helps to find these neighboring nodes and hence we can setup a neighboring patches relation lookup table using node searching algorithms. The merge process tries to merge the neighboring patches and makes sure that no two parallel neighboring patches are joined together. The insertion process merges the neighboring planes to find corners, edges, etc. This method has various advantages such as data filtering and data compression with the octree structure useful for reconstruction of the 3D model containing the extracted features.

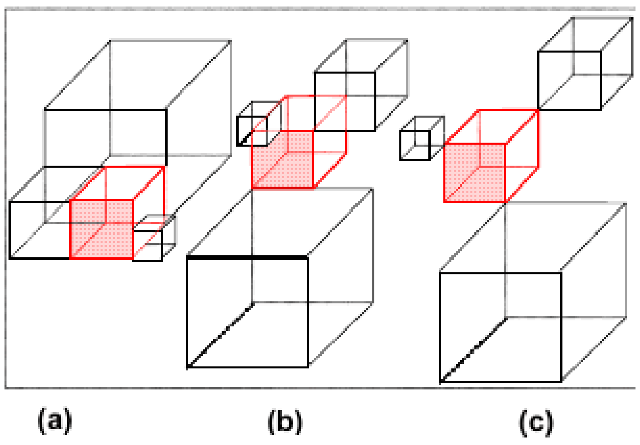


Figure 3.2: Octree Neighboring nodes type [31]

3.3.2 Diffusion Maps and Diffusion Principal Component Analysis

Farbman et. al. [11] discusses the usage of diffusion maps, a non-linear dimensionality reduction in place of Euclidean distance and has shown better distribution of pixels in the feature space. For a given data set, a pairwise affinity matrix is formed as in equation 3.4 and normalized diagonally. The corresponding set of eigenvectors $\psi_0, ..., \psi_n 1$ mapped from the original dataset and scaled using their respective eigenvalues $\lambda_0, ..., \lambda_{n-1}$ represent the diffusion map at time t as given in Equation 3.5 and the difference between any two values in the diffusion map gives the diffusion distance as shown in Equation 3.6.

$$W_{i,j} = k(x_i, x_j) \tag{3.4}$$

$$\phi_t(x) = (\lambda_1^t \phi_1(x), \lambda_2^t \phi_2(x)......\lambda_n^t \phi_n(x)) \tag{3.5}$$

$$D_t(x, y) = \|\psi_t(x) - \psi_t(y)\|_2 \tag{3.6}$$

Since the normalization of the affinity matrix W requires heavy computational cost, Nystrom approximation method is used for calculation purposes. The entire pixel dataset is divided into two samples where the smaller sample set is distributed over the rest of the image and pairwise affinity matrices are calculated over this smaller dataset, thereby reducing the computational steps involved. The work [11] basically uses the color and pixel spatial position which sometimes does not work well for textures having a lower spatial resolution.

According to [9], diffusion maps were observed to integrate local connectivity to recover parameters of change at different time scales. In that paper, Diffusion maps were compared with three other data reduction techniques, (PCA(linear),MDS(Multidimensional scaling) and isomap) and it was found that the diffusion mapping is more robust to noise perturbation, and it is the only technique that allows geometric analysis at differing scales. Due to their locality preserving nature diffusion maps were thought to be relatively insensitive to noise. This was numerically demonstrated by Lafon and Lee in [17].

Construction of Diffusion Maps

The construction of the diffusion maps and its properties goes as follows as in [6].
The data set X is taken and a kernel function k is defined on the data points as $k : X \times X \to R$ satisfying the following properties:

- k is symmetric: $k(x, y) = k(y, x)$,
- k is positivity preserving: $k(x, y) \geq 0$.

From the definition of the kernel function, it is understood that it represents affinity or similarity between data points of X.

The data points are taken to be the nodes of a symmetric graph (since kernel function is symmetrical) and the kernel value $k(x, y)$ is taken to be the weights $w(x, y)$ of the edges. Care should be taken in constructing the kernel function because as said in[6] "a given kernel will capture a specific feature of the data set, its choice should be guided by the application that one has in mind." In the papers [6, 13], only the Gaussian function is taken to be the standard kernel function and no other kernel functions are considered. From the graph defined by (X, k) a reversible Markov process is considered on X. The one-step transition probabilities p_{ik} from node i to node k are obtained directly from these weights:

$$p_{ik} = \frac{w_{ik}}{\sum_j w_{ij}}.$$

P is viewed as the transition matrix of the Markov process of jumping from the node i to k in one time step. Here t is viewed as a parameter.

In [13] it is also assumed that the starting point for the Markov random walk is chosen uniformly at random.(i.e) $p(i) = 1/N$.

The Diffusion distance D_t between the points x and z is given as [17]:

$$D_t^2(x, z) = \|p_t(x, .) - p_t(z, .)\|_{1/\phi_0}^2 = \sum_{y \in X} \frac{(p_t(x, y) - p_t(z, y))^2}{\phi_0(y)} \tag{3.7}$$

The powers of P are observed as the corresponding scales to study the geometric structures of X at various scales. P is taken to be the operator on X and its spectral properties in terms of eigen values and eigenvectors are studied in [6] to have an understanding of the of powers of P.

The Markov random process constructed using the kernel function from the neighborhood graph of the data set X is seen to have the following properties: The Markov chain has a stationary distribution which is given by: $\pi(y) = \frac{d(y)}{\sum_{z \in X} d(z)}$. The chain is reversible, $\pi(x)p(x, y) = \pi(y)p(y, x)$.

If X is finite and the graph of the data is connected, then the chain is ergodic.

From [17], it is understood that the transition matrix P that is constructed has a set of left and right eigenvectors and a set of eigenvalues$|\lambda_0| \geq |\lambda_1| \geq \ldots\ |\lambda_{n-1}| \geq 0:$

$$\phi_j^T P = \lambda_j \phi_j^T \quad \text{and} \quad \mathrm{P\psi_j = \lambda_j \psi_j}$$

where it can be verified that $\lambda_0 = 1$, and $\psi_0 \equiv 1$ and $\phi_k^T \psi_l = \delta_{kl}$.

So ϕ_j^T are left eigenvectors and ψ_j are right eigenvectors of P belonging to the same eigenvalue λ_j and they are dual to each other as it is with all the symmetric matrices.

If $p_t(x, y)$ is the kernel of the tth iterate P^t, then the following is the bi-orthogonal spectral decomposition:

$$p_t(x, y) = \sum_{j \geq 0} \lambda_j^t \psi_j(x) \phi_j(y)$$

The first k terms provide the best rank-k approximation of P^t. So it is seen from (3.7) that

$$D_t^2(x, z) = \sum_{j=1}^{n-1} \lambda_j^{2t} (\psi_j(x) - \psi_j(z))^2.$$

Now the above sum is approximated as follows:

Let $q(t)$ be the largest index j such that$|\lambda_j|^t > \delta |\lambda_1|^t$. The diffusion distance can then be approximated to relative precision δ using the first $q(t)$ non-trivial eigenvectors and eigenvalues as

$$D_t^2(x, z) \simeq \sum_{j=1}^{q(t)} \lambda_j^{2t} (\psi_j(x) - \psi_j(z))^2.$$

If $\Psi(t)$ is the Diffusion map then

$$\Psi_{(t)} : x \to \begin{pmatrix} \lambda_1^t \psi_1(x) \\ \lambda_2^t \psi_2(x) \\ \vdots \\ \lambda_{q(t)}^t \psi_{q(t)}(x) \end{pmatrix}$$

then,

$$D_t^2(x,z) \simeq \sum_{j=1}^{q(t)} \lambda_j^{2t} (\psi_j(x) - \psi_j(z))^2 = \|\Psi_{(t)}(x) - \Psi_{(t)}(z)\|^2.$$

The mapping $\Psi_{(t)} : X \to \mathbb{R}^q(t)$ provides a parametrization of the data set X as well as the realization of the graph G as a cloud of points in a lower-dimensional space $\mathbb{R}^q(t)$, where the re-scaled eigenvectors are the coordinates.

The diffusion distance defined above between any two data points is used to compute the corresponding Voronoi Cell. We then perform the Principal Component Analysis on the Voronoi cell and classify the points according to the eigenvalues of the covariance matrix. These eigenvalues chosen within certain threshold levels are used to detect sharp points as well as smooth points according to their magnitude. This method gives the multi scale view of the data as we vary the parameter t.

This way, a smaller data set consisting of feature points is obtained, where different values of t give different scales of view of the data.

3.3.3 Difference of Normals(DoN)

A multi-scale operator, Difference of Normals is used to segment large 3D unorganized point cloud datasets. The major idea of this paper [12] is that for any given radius, the surface normal estimation provides an overview of the geometry of the surface. In general, if two surface normals are calculated based on two radii, one larger than the other, the direction of both the normals will not differ at a larger scale if the surface does not change for both the radii. The DoN operator Δ_n can be represented for any two radii $r1$ and $r2$ for a point $p \in P$ as in Eq 3.8

$$\Delta_{\hat{n}(p,r_1,r_2)} = \frac{\hat{n}(p,r_1) - \hat{n}(p,r_2)}{2} \tag{3.8}$$

Here $r1 < r2$ and $\hat{n}(p,r)$ is the surface normal estimate for a point p over a support radius r. Figure 3.3 shows an illustration of the DoN operator

The principal components of the local neighborhood for a fixed radius were used for estimating the normals and each of the neighborhood contains more than three points. The points were reduced by eliminating those points whose DoN magnitude was lesser than a specified threshold value. Since the normal calculation over a neighborhood search for each radii value requires tedious computational work, the point cloud is sub-sampled using the uniform re-sampling algorithm where the point cloud is voxelized based on the decimation parameter d. This makes the computation faster and using GPU parallelization, the computation is reduced further.

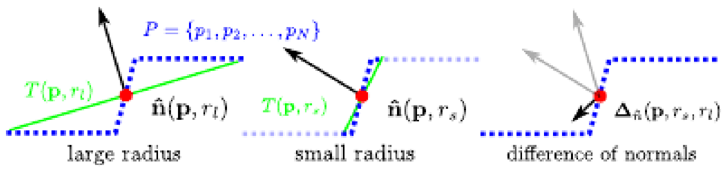

$P = \{p_1, p_2, \ldots, p_N\}$
$T(\mathbf{p}, r_l)$
$\hat{\mathbf{n}}(\mathbf{p}, r_l)$
$T(\mathbf{p}, r_s)$
$\hat{\mathbf{n}}(\mathbf{p}, r_s)$
$\Delta_{\hat{\mathbf{n}}}(\mathbf{p}, r_s, r_l)$
large radius
small radius
difference of normals

Figure 3.3: DoN Operator [12]

This method was effective to sustain the points within an object scale while discarding the points away from the object. Urban-based LiDAR data points were segmented qualitatively with good reduction of data points.

3.4 LiDAR Data Analysis

Most of the high dimensional data in case of LiDAR, image processing, etc. are in the form of point cloud which is difficult to reconstruct and hence several theories are carried out to represent the topological information and analyze them. Several techniques such as contour trees [30], morse theory [10], reeb graphs [23], etc. represent the topological structures using the concept of scalar field topology. This section discusses these data analysis techniques and concepts in detail. In general, the data points are defined in terms of scalar functions

3.4.1 Morse Theory

This theory denotes a function f which satisfies certain constraints over a known surface data S. Baby morse theory [20], a subset of the morse function is based on the Poincare function(equ.n 3.9) and is used for deriving the morse polynomial equation 3.10 which provides with the critical points present in the scalar function. This morse polynomial function can be either weak or strong in comparison with the Poincare function and when both the functions are equal, then the function is termed as perfect. In general, a morse function [20] can simply be expressed as a smooth function where no two critical points cannot have the same function value. Mathematically, the morse lemma shows that the neighborhood of the critical point is quadratic in nature and hence the function can either be a paraboloid or saddle surface.

$$P(S) = \sum_k b_k t^k \tag{3.9}$$

where b_k is the Betti number and 1 t 1.

$$M(f) = \sum_{P_i} t^{n_i} \tag{3.10}$$

where P_i = critical points of f, n_i = number of non-decreasing directions for f.

Lemma 1. *Consider p as a critical point of function f such that for p as origin the local coordinates are denoted by $(x1, x2, ..., xn)$, then f can be expressed in terms of these local coordinates as follows:*
$f = f_(p) \pm x_1^2 \pm x_2^2 \pm ...... \pm x_n^2$

The morse inequalities helps to identify the topological structure only when the function is perfect and hence Lacunary principal [3] is used for searching such functions. This theory is used for estimating the genus of a 2D surface from a collection of 3D point cloud data and the term genus referred here characterizes the number of handles present which forms the 3D model. Hence this method is useful for reducing the dimensionality and identifying the topology using the empirical distributions. This study has been extended towards observation of topological structures using scalar fields.

3.4.2 Topological Persistence and Persistence Diagrams

Topological persistence gives the difference between any two functional values representing the critical points and hence provides with the maxima that is either obtained through the signal or the noise. All these critical points can be paired and this in turn provides with a set of intervals known as persistence bar-code which gives the homological information pertaining to the subsets of the function f. Such pairs are removed iteratively using the persistence bar-code of the function. The noise obtained was further analyzed and reduced [27] by canceling the pairs that have a short interval with respect to the bar-code. On the other hand, these intervals gives rise to a set of points in a plane known as persistence diagram which are useful for retrieving critical points corresponding to a scalar field.

Chazal et. al. [5] proposed an algorithm for extracting structural information from a point cloud data by deriving persistence diagrams over their scalar fields i.e for a point cloud subset L present in some structural space X, if there exists some real valued function f, then the persistence diagram of f was obtained through an algebraic construction among the nested families of complexes that are obtained using the Rips Complex (R_δ(L)) .The algorithm comprises mainly of the following steps:

1 For an n-dimensional vector v containing data points, a pairwise distance matrix D is constructed and two families of nested Rips complex is constructed.

2 For k:=0 to nth complex dimension, compute the kth persistence diagram.

This core algorithm is improvised for extracting spatial information through the assumption that all the data points in the function f are different. Firstly, the core algorithm produces a 1-skeleton graph of the Rips complex known as the Rips graph G. The new improvement approximates the gradient vector field of the function f with respect to the vertices present in G and clustering is done using the basins of attraction in G which are merged using the $0th$ persistence diagram of function f. A set of values(v, e) is obtained where v is the local minimum of f and e is an edge connecting the component to other older components, both produced by G.

3.4.3 Reeb Graphs for data skeletonization

A graph that monitors the iso-contours of scalar functions over a well-defined manifold is termed as reeb graph. This graph [23] is obtained for any continuous function f defined over a space X such that for any two points x and y from the same connected component of f within a level set, the reeb graph will be an image of a continuous surjective map $\phi : X - \zeta R_f(X)$ such that $\phi(x) = \phi(y)$. For instance Figure 3.4 gives the reeb graph of the height function f which is defined over a topological torus where the connected components are present in the cases of degree-1 nodes and split up or merge in the cases of degree-3 nodes. When these components are connected without loops in a reeb graph, it is termed as contour trees.

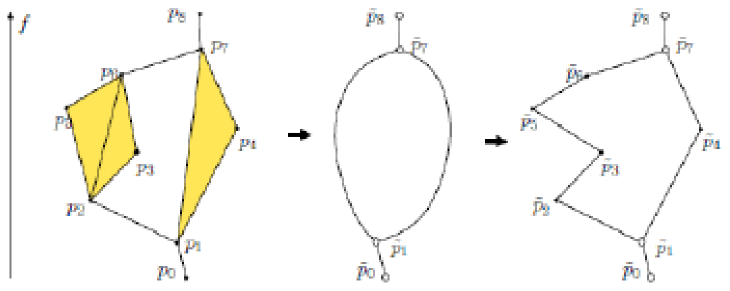


Figure 3.4: Reeb graph of height function f [23]

The algorithm developed by [23] explains how unstructured point cloud data is dimensionally reduced with the help of reeb graphs.

3.5 GPU Parallelization

GPU parallelization using NVIDIA with CUDA will help to speed the processing in the methods mentioned above. However, we need to first analyze which methods will produce sufficient results even after parallelization as this might produce distortions in the results. Both the Keller et al. [16] and the DoN [12] methods are embarrassingly parallel and hence can be improvised using GPU functions. For this project, the Keller et al. method produces good results and DoN can be used as an alternative for reduction of the points in the point cloud data and hence both these methods need to be focused upon. The PCL library has inbuilt GPU-based DoF algorithm which can be useful for this purpose. This operator can be used within the algorithm [10] for reducing the points apart from the outliers removal used by the algorithm.

3.6 Conclusions

This literature review provides with the existing methods [10] [12] for extracting important features from point cloud data which reduces the huge point cloud size by removal of unwanted data and further optimizing to provide with good results. Certain alternative methods for reduction of point cloud data was also sought after but many of these methods also had the disadvantage of removing data points representing the important features and also the fact that the results were not as good as Keller et al [16]. The major aim of this project is to improvise the Keller et al. method in terms of time consumption as well as point reduction which can be done through GPU parallelization.

Chapter 4

Feature Detection and Extraction

Since LiDAR is a large scale data, the processing of LiDAR data-set become computationally very expensive and time consuming. One of our goal is to reduce the point cloud to essential points that encodes structural characteristics of a point. These essential points may be defined as feature points. We have focused mainly on airborne urban lidar data-set. Talking about urban datasets, features may be defined as curves, corners or patches. In case of 3D points, curve features usually includes crease, border, ridges, etc. Apart from point reduction, feature extraction has other application also, for example, curve feature can be useful to geologists to extract meaningful information about the data. We are also interested in these type of features. It has other applications also like urban planning, building change detection after natural hazards, civil engineering and lots of other areas.

Lidar point cloud are noisy, sampled biased and sparse in nature due to atmospheric conditions in which data has been collected. Also, no connectivity information is available in the data i.e. it is unstructured. Due to all these problems, feature detection and extraction in lidar point cloud becomes a challenging task. We have implemented an already existing user-assisted heuristic algorithm proposed by Keller et al. [15] to detect and classify the feature points. This algorithm is computationally very expensive and does not show real time performance. Therefore, to make application real-time, we have done the parallel implementation for Keller et al. [15] to make application real-time and user interactive.

Feature detection and extraction results in reduced subset of the point cloud which can represent the underlying geometry of the surface. But the data is still in the point format. Mesh can be constructed from the reduced point cloud to build a 3D model of the surface. Mesh computation is a very expensive task. Therefore, we have used the reduced point cloud to generate the mesh. Triangle primitive is used to construct the mesh. Triangular mesh are generated using the points within the point cloud and it is continued until all the points are joined. As a result, a mesh of triangles are formed which can approximately model the surface.

In this chapter, we have discussed the algorithm proposed by Keller et al. [15]

in section 4.2. Section-4.3 discuss the parallel implementation of algorithm proposed by Keller. Surface is constructed from the reduced point cloud based on region growing algorithm and it is discussed in details in the section 4.4. Design flow of serial and parallel implementation is explained in section 4.5. The system architecture of the complete application is discussed in the chapter 7.

4.1 What is a feature?

Definition of feature is application specific and feature points may be defined as those points which gives some useful information about the data and remain persistent with the time. Slight perturbations in data do not disturb the nature of feature point. In our case, feature points are those points which represents geometrical structure of the data. We are mainly interesting in detecting point-type, curve-type and patch-type feature points. Figure 4.1 represents important classes of the features in 3D point cloud. Point-type features are those points where behavior in not defined i.e. they are isotropic in nature. These are corner and junction points or end points of curve as can be seen in the Figure 4.1. Curve-type features may be defined as creases, borderlines or ridges present in the data. These features are used to composed the surface patch in the 3D data. They can also be defined as continuous loci of points pass along a ridge of maximum inflection. Patch-type feature is a surface of similar curves and are formed by curves lines passing on the surface.

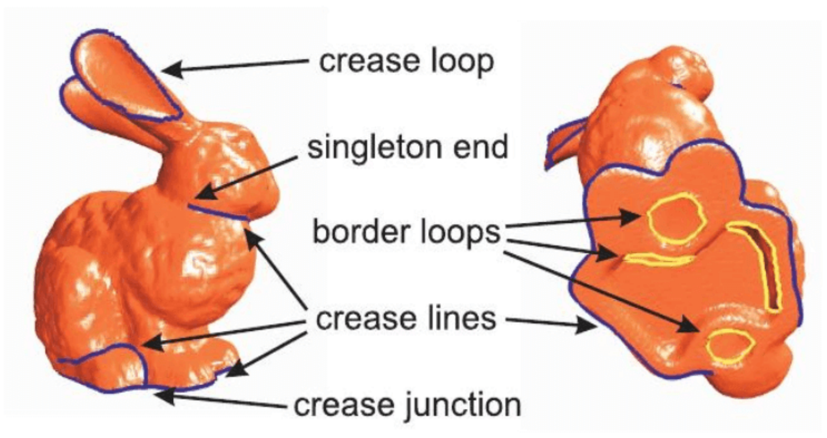
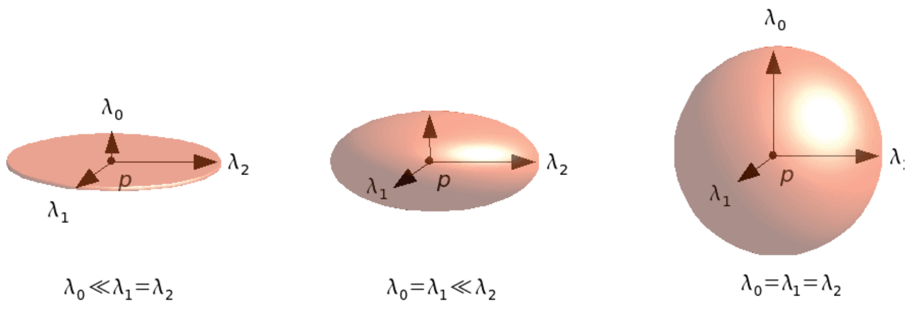


Figure 4.1: Different type of feature points in 3D point cloud[15].

4.2 Approach

We have implemented an already existing algorithm proposed by Keller [15] which is motivated by the aim to detect and extract structural features of the objects in the lidar point cloud over multiple scales. These features include corners, border-, crease-, or ridge-lines in the form of a feature graph. Our application uses this algorithm to down-sample the data such that topology of the data is preserved. Keller et al. mainly consists of following steps:

- Preprocessing of the 3D point cloud using an octree data structure,
- Stochastic point classification based on local point neighborhood and specific feature values,
- Post-processing by de-noising and smoothing obtained feature values using weighted average filter,
- Construction of the feature graph using an edge-growing approach from selected seed nodes.



λ_0
λ_2
p
λ_1
$\lambda_0 \ll \lambda_1 = \lambda_2$
λ_2
λ_1
p
$\lambda_0 = \lambda_1 \ll \lambda_2$
λ_0
λ
λ_1
p
$\lambda_0 = \lambda_1 = \lambda_2$

(a) (b) (c)

Figure 4.2: Stochastic method used for point classification based on shape of local neighborhood, as proposed by Keller et al. [15, 14], namely, (a) planar or disc-like, (b) cylindrical, and (c) spherical neighborhoods [14].

Keller et al. have focused specifically on using a multi-scale method to express the membership of local point neighborhood to corresponding geometric shape classes using probability based on user defined parameters. For each point in the point cloud, the likelihoods of the local neighborhood of the point corresponding to the shape classes, namely, spherical, cylindrical, and disc-like neighborhood are computed. This is a stochastic point classification technique which classifies each point in the point cloud based on its specific structural properties. The shape classes are as shown in Figure 4.2. Subsequent to the point classification, the algorithm uses shape-specific parameters, also known as feature values, to identify feature points and feature lines which help in recognizing shapes like buildings. Figure 4.3 shows the curvature value which is used as feature strength for disc-type shape class. Planar region has low curvature value while non-planar region has high curvature value. Point Classification into different classes: curve, surface, critical curve and critical surface points of area 1, area 2, area 3 site of vaihingen data-set are shown in the Figure 4.4, 4.5 and 4.6 respectively.

The subsequent construction of the feature graph additionally requires critical points, which are points in the lidar data correspond to points with highest likelihood of having spherical neighborhoods, implying isotropic nature. Seed points for constructing graphs have been identified based on likelihood and feature strength, and are connected based on proximity and propagating direction.

The theory behind the shape classes of the neighborhood as well as the technique used for edge propagation direction of the feature graph is based on the principal component analysis (PCA) of the local neighborhood of each point in the data set. The points that are not reachable by the feature graph are finally eliminated, thus results in down-sampling of the data set as shown in the Figure 4.7.

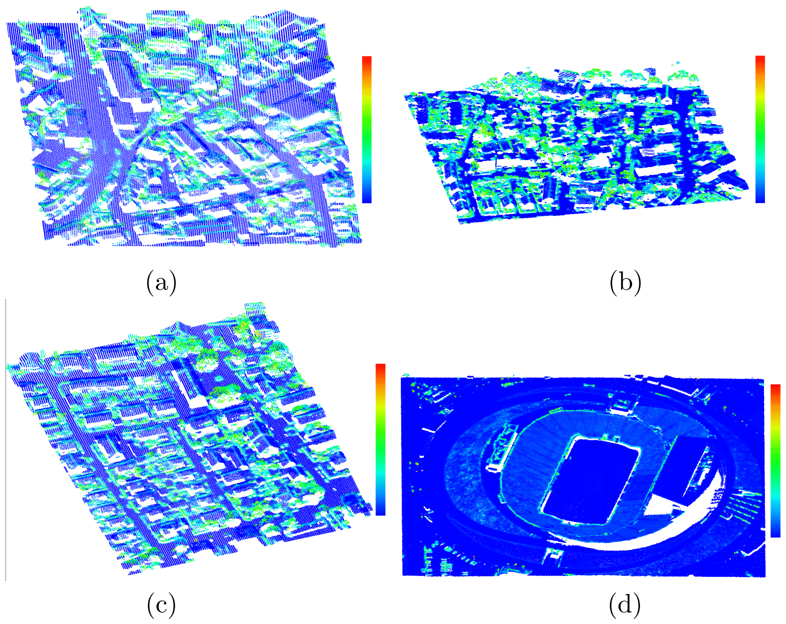


Figure 4.3: Mean principal curvature are shown using RGB color spectrum for (a) Area 1, (b) Area 2, (c) Area 3 of Vaihingen data-set [7] and (d) Oregon's Autzen Stadium data set [4]. Blue shows low curvature value while red shows high curvature value

The feature extraction module is highly computationally intensive and time consuming. In order to make the application interactive and real-time, we have proposed a parallel implementation for Keller et al. by exploiting the data-parallel characteristic of the algorithm using GPGPU computing techniques [22]. PCL [26] and CUDA [21] library has been used for parallel implementation.

CUDA, Compute Unified Device Architecture is a parallel computing device architecture developed by NVIDIA for GPGPU. CUDA is supported only by NVIDIAs graphics card from the G80 series onward. Unlike shader programming, where GPU can be used for graphics related applications, CUDA provides the facility to use GPU for general purpose applications known as GPGPU computing. With the help of CUDA, GPU can be utilized to its full potential as general purpose computing can also be done.

PCL is an open source library released under BSD license and free to use for research and commercial purpose. It is a template C++ library for 2D/3D points processing with CUDA integration for GPU computing. It internally uses eigen library for fast computation of linear algebra methods and boost shared pointers to passes data between different modules. It also uses OpenMP and TBB for parallelization.

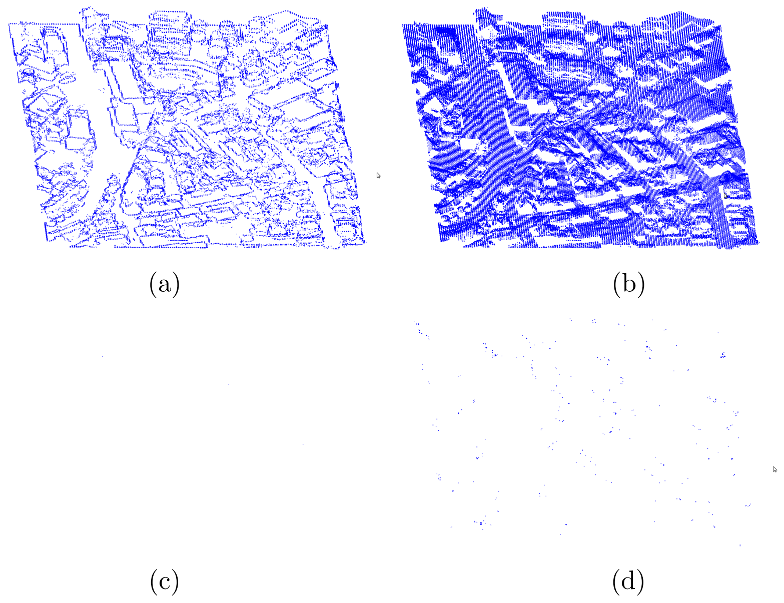


Figure 4.4: Feature extraction for area 1 of vaihingen data-set: (a) Curve Points, (b) Planar Points, (c) Critical Curve Points, (d) Critical Planar Points. Figure (c) and (d) has only 3 and 397 points, therefore images almost look white.

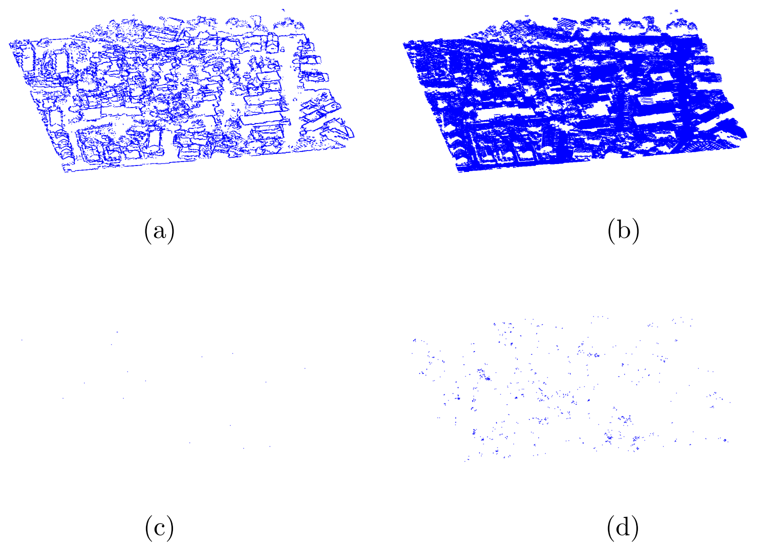


Figure 4.5: Feature extraction for area 2 of vaihingen data-set: (a) Curve Points, (b) Planar Points, (c) Critical Curve Points, (d) Critical Planar Points. Figure (c) and (d) has only 17 and 1,129 points, therefore images almost look white.

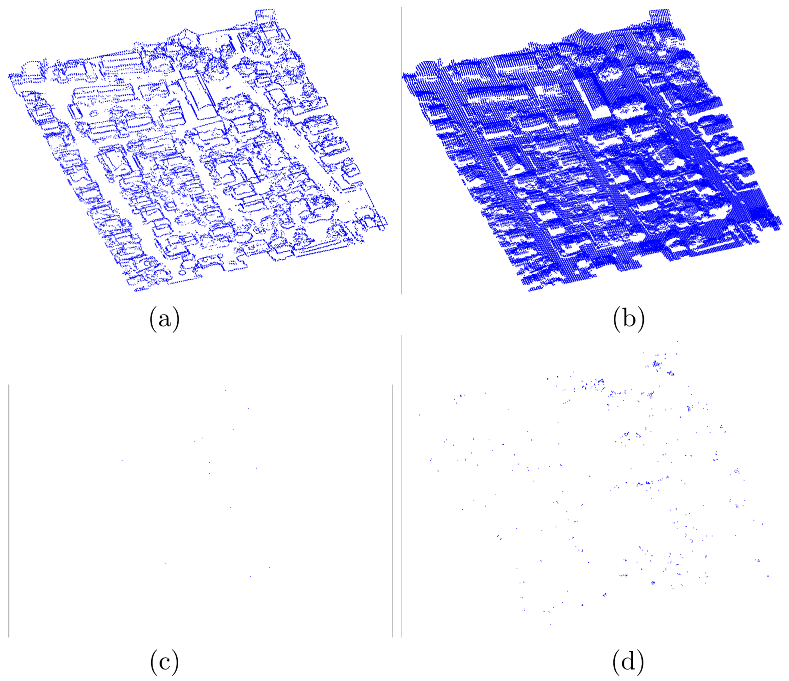


Figure 4.6: Feature extraction for area 3 of vaihingen data-set: (a) Curve Points, (b) Planar Points, (c) Critical Curve Points, (d) Critical Planar Points. Figure (c) and (d) has only 14 and 707 points, therefore images almost look white.

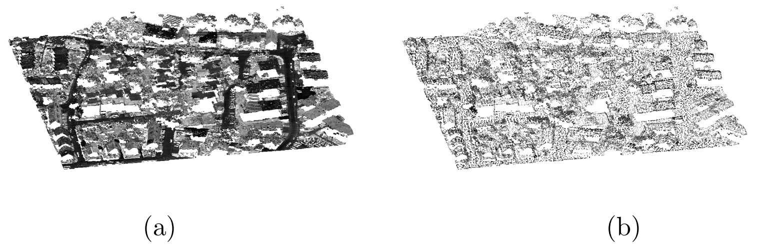


Figure 4.7: Down-Sampling of area 2 of vaihingen site: (a) Original Points, (b) Reduced Points. Intensity value is encoded using grey-scale spectrum. Figure (a) contains 2,33,782 which is reduced to 61,915 in Figure (b).

4.3 Parallel Algorithm

4.3.1 Outlier Removal

In Keller et al., statistical outlier removal filter has been used to remove the outliers. This method needs to search $k-$neighbors for each point which is computationally very expensive. Instead of statistical outlier removal filter, we have used conditional outlier removal filter to remove the outliers. Computation time is reduced by implementing the parallel algorithm for outlier removal. Data has been stored using octree data structure as search operations like k-nearest and r-nearest neighbor search can be performed in a fast way using octree. Octree has been built in parallel using PCL library [26]. Conditional outlier removal filter removes outliers by comparing density i.e. number of points within a sphere of a given radius with a particular threshold. If density is less than threshold, point is considered as outliers. $r-$nearest neighbors and density measurement for a voxel is independent of the other voxels. Therefore, parallel implementation has been done for $r-$nearest neighbors and density measurement and pseudo code is given in the algorithm 1.

Algorithm 1 Skeleton code for Outlier removal

```
 1: Store the data in octree data structure using PCL library
 2: Copy the data from CPU memory to GPU memory
 3: Search $r-$nearest neighbor for all voxels using PCL library
 4: Copy the result from GPU memory to CPU memory
 5: Copy the voxels and $r-$nearest neighbors for all 3D points from CPU to
    GPU memory
 6: Assign $N$ threads Where $N$ is equal to total number of 3D points and launch
    a cuda kernel
 7: for every voxels in parallel do
 8:     If $r-$nearest neighbor of $i$th is less than a threshold, remove that point.
 9: end for
10: Copy the indices of outliers from GPU to CPU
11: Delete the outliers from octree
```

4.3.2 Point Classification

Stochastic point classification involves finding a spherical neighborhood of each point and estimating the feature value of each point by computing the curvature of the polynomial fitted to the point neighborhood of each point using Moving Least Squares (MLS) method. The radius of the spherical neighborhood is user-defined and the feature strength for spherical and cylindrical class is computed using eigen values while for disc class, it is the root-mean-squared (RMS) curvature. The nearest neighbor search and curvature estimation are embarrassingly parallel tasks, therefore, we have implemented the parallel algorithm using GPU computing.

Octree data structure is built using PCL library. The r-nearest neighbors search are also done using the PCL library. Pseudo code for parallel implementation of curvature estimation are given in algorithm 3 . The probability measurement to classify the point into different class is also embarrassingly par-

Algorithm 2 Build a 3x3 Covaraince matrix

```
1: Set all value of a 3x3 covariance_matrix to zero
2: $N$ is the total number of points
3: for $i = 0 \to N$ do
4:     $pt = points[i] - centroid$; {pt and points is a 3D point}
5:     temp = $pt^T * pt$ {$temp$ is a 3x3 matrix}
6:     $covariance$_matrix = $covariance$_matrix + $temp$
7: end for
```

Algorithm 3 Measure the curvature value

```
1: Search the $r-$nearest neighbors as we have discussed in outlier removal
   algorithm
2: Simultaneously calculate curvature value for multiple points
3: Copy the input voxels and their $r-$nearest neighbors from CPU to GPU
   global memory
4: Assign N threads on GPU which is equal to total number of 3D points and
   launch a cuda kernel
5: for all every voxel in parallel do
6:     Define the local plane by fitting the bi-variate polynomial to the $r-$nearest
       neighbor of each point.
7:     Determine the Weingarten gaussian matrix by calculating the first and
       second derivative of polynomial.
8:     Determine the eigen-values and eigen-vectors of the Weingarten gaussian
       matrix.
9:     Find the root mean square value of the eigen-values which is the mean
       curvature value
10:    Return the curvature value
11: end for
12: Copy the curvature value of all voxels from GPU to CPU memory
```

allel task. We have done the batch processing for computing the probability and pseudo code for probability computation are given in algorithm 4.

Algorithm 4 Stochastic point classification

```
1: Decompose the data in an octree data structure using pcl library
2: for $j = 0 \rightarrow scale$ do
3:     Search the $r-$nearest neighbor for all voxels using pcl library
4:     Copy all voxels and their $r-$nearest neighbor to GPU global memory
5:     Assign $N$threads where $N$ = total number of voxels and launch a cuda kernel
6:     for every voxel in parallel do
7:         Build the covariance matrix
8:         Spectral decomposition of covariance matrix
9:     end for
10:    Assign $N$threads where $N$ = total number of voxels and launch a cuda kernel
11:    for every voxel in parallel do
12:        Measure the probability and feature strength value using the following formula:
13:            $L_{cp}(p) = \begin{cases} 1, & \text{if } \lambda_0 \geq \varepsilon\lambda_2 \\ 0, & \text{otherwise} \end{cases}$
14:            $L_p(p) = \begin{cases} 1, & \text{if } \lambda_1 < \varepsilon\lambda_2 \\ 0, & \text{otherwise} \end{cases}$
15:            $L_{cp}(p) = \begin{cases} 1, & \text{if } \lambda_0 < \varepsilon\lambda_2 \\ 0, & \text{otherwise} \end{cases}$
16:    end for
17:    Assign $N$threads where $N$ = total number of voxels and launch a cuda kernel
18:    for every voxel in parallel do
19:        Measure the curvature value
20:        Measure the feature strength
21:    end for
22:    Copy the result from GPU to CPU memory
23: end for
```

4.3.3 Edge Propagation

Edge propagation depends on using the major eigen vector of the covariance matrix of a point neighborhood as the propagation direction. Determination of the largest eigen value can be done simultaneously for multiple points. Therefore, we have determined the propagation direction simultaneously for all points using parallel algorithm as given in 5. This has speed up the construction of the feature graph considerably.

Algorithm 5 Edge propagation

```
 1: Decompose the data in an octree data structure using pcl library
 2: Perform the batch processing to search the $r-$nearest neighbors for seed
    and critical points only, using pcl library
 3: Copy the seed and critical points, and their $r-$nearest neighbor from CPU
    to GPU global memory
 4: Assign $N$ threads where $N$ = total number of voxels and launch a cuda
    kernel
 5: for every voxel in parallel do
 6:     Build a covariance matrix
 7:     Spectral decomposition of a covariance matrix
 8: end for
 9: Copy the eigen values of all voxels from GPU memory to CPU memory
10: for every seed points do
11:     Connect to a node based on relations given in algorithm
12:     Delete all the nodes which are lying closer than a connected node
13:     Delete all nodes forming acute angle with seed node
14: end for
```

4.4 Mesh Based Region Growing

The method known as fast triangulation of unordered point clouds is used for construction of mesh. It is an already established method that seems to work pretty much well for a variety of different point cloud datasets without making much changes to the default parameters. We have, therefore, utilized the same functionality.

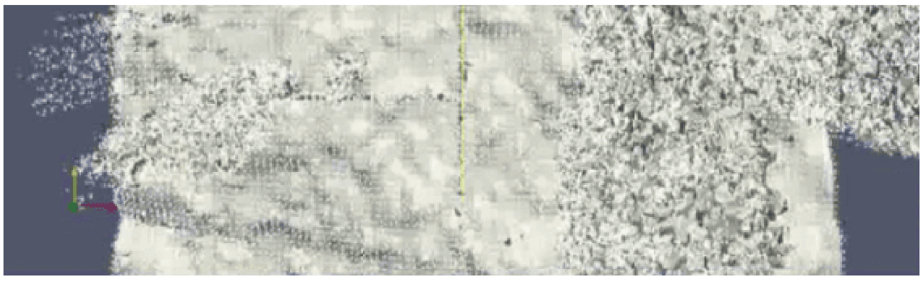


Figure 4.8: Mesh Generation On A Point Cloud Data

This method is also popularly known as greedy surface triangulation algorithm as it performs a triangulation on a point cloud (native data type in pcl) with normals, to obtain a triangle mesh based on projections of the local neighborhood. It starts by keeping a list of points from which the mesh can be grown, sequentially. These points are termed as 'fringe points' and starting from here, it keeps extending until all the points are covered. This algorithm has been found to perform well in case of unorganized data obtained from possibly multiple scans and/or containing multiple connected paths. However, just like others, it does its best when the surface is locally smooth and where the transition between areas of different densities is also gradual.

4.5 Implementation

Our application runs on server-client architecture where server performs all high-duty computations and clients are used as a thin device for display purpose and send feedbacks of user interactions on the network. Local Area Networking has been implemented using a product called ThinLinc [1], completely based upon open-source projects like VirtualGL, TigerVNC and many others creating a wholesome feature-packed server-client distribution package. The server-client architecture and implementation and usefulness of ThinLinc have been explained in the Chapter 7 and 6 respectively.

In this section, we will explain the implementation and design flow of the algorithm used in our application. Figure 4.9 shows the design flow for serial implementation of Keller et al. [15]. The algorithm mainly consists of five major block: storage of point cloud, outliers removal, feature detection, filtering and point classification and construction of feature graph. Excepts for first block, other blocks are computationally very intensive and therefore, we have developed a parallel algorithm. Design flow for parallel implementation has been given in the Figure 4.10. GPU power has been used to parallelize the independent tasks and implementation has been done using PCL library and CUDA.

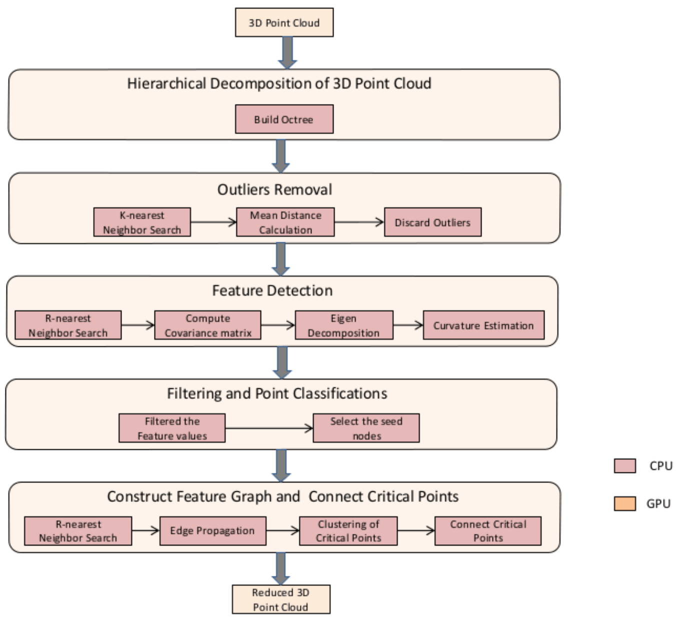

3D Point Cloud
Hierarchical Decomposition of 3D Point Cloud
Build Octree
Outliers Removal
K-nearest Neighbor Search
Mean Distance Calculation
Discard Outliers
Feature Detection
R-nearest Neighbor Search
Compute Covariance matrix
Eigen Decomposition
Curvature Estimation
Filtering and Point Classifications
Filtered the Feature values
Select the seed nodes
Construct Feature Graph and Connect Critical Points
R-nearest Neighbor Search
Edge Propagation
Clustering of Critical Points
Connect Critical Points
Reduced 3D Point Cloud
CPU
GPU

Figure 4.9: Algorithmic design flow of serial implementation

The main application will run on CPU and it will use the GPU to run the parallel tasks. To perform the computation on GPU, data has to be copied from CPU memory to GPU memory, then performs the tasks and copy back the results from GPU memory to CPU memory. We have done the parallel processing on GPU using NVIDIA's CUDA. For batch processing on GPU, a cuda kernel has to launched where each independent task is assigned to a separate thread so that batch processing can be performed to speed up the

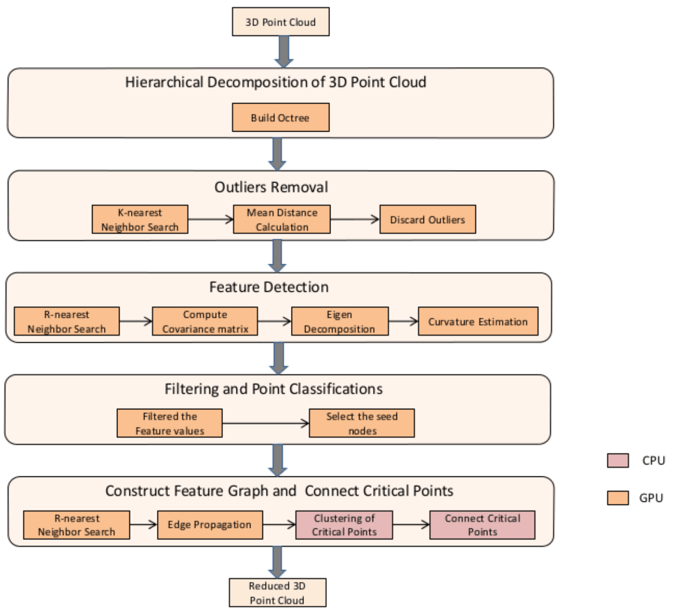

3D Point Cloud
Hierarchical Decomposition of 3D Point Cloud
Build Octree
Outliers Removal
K-nearest Neighbor Search
Mean Distance Calculation
Discard Outliers
Feature Detection
R-nearest Neighbor Search
Compute Covariance matrix
Eigen Decomposition
Curvature Estimation
Filtering and Point Classifications
Filtered the Feature values
Select the seed nodes
Construct Feature Graph and Connect Critical Points
R-nearest Neighbor Search
Edge Propagation
Clustering of Critical Points
Connect Critical Points
Reduced 3D Point Cloud
CPU
GPU

Figure 4.10: Algorithmic design flow of parallel implementation

computations. A cuda kernel is a program written by the programmer that gets executed on many threads simultaneously on GPU. The pseudo code for complete algorithm is listed in algorithm 6.

Algorithm 6 Skeleton code of complete algorithm for CPU

```
1: Read the input file and load the input data
2: Scale the input data between [-1 1]
3: Perform the hierarchical decomposition of input data and store in an octree
   data structure. Octree has been built in-parallel using PCL library.
4: Outlier removal has been done. {perform batch processing for multiple
   points}
5: Reload the outlier removed data in octree.
6: Point classification has been done { Done in-parallel for multiple points}
7: Feature filtering has been done { Done in-parallel for multiple points}
8: Seed selection for curve and edge graph has been performed.
9: Curve Graph has been constructed { r-nearest neighbor search and eigen
   decomposition are performed in-parallel}
10: Curve critical nodes are merged in curve graph
11: Disc graph has been constructed { r-nearest neighbor search and eigen de-
    composition are performed in-parallel}
12: Disc critical nodes are merged in disc graph
13: Curve and disc graph are merged together
14: Display the reduced points using openGL pipeline
```