EFFICIENT RENDERING OF REAL-WORLD ENVIRONMENTS IN A VIRTUAL REALITY APPLICATION, USING SEGMENTED MULTI-RESOLUTION MESHES

by

Tanaka Alois Chiromo

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Electronic Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

May 2020

SUMMARY

EFFICIENT RENDERING OF REAL-WORLD ENVIRONMENTS IN A VIRTUAL REALITY APPLICATION, USING SEGMENTED MULTI-RESOLUTION MESHES

by

Tanaka Alois Chiromo

Supervisor(s):	H. Grobler
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Electronic Engineering)
Keywords:	segmentation, surface reconstruction, texturing, virtual reality, 3D point-cloud, 3D mesh

Virtual reality (VR) applications are becoming increasingly popular and are being used in various applications. VR applications can be used to simulate large real-world landscapes in a computer program for various purposes such as entertainment, education or business.

Typically, 3-dimensional (3D) and VR applications use environments that are made up of meshes of relatively small size. As the size of the meshes increase, the applications start experiencing lagging and run-time memory errors. Therefore, it is inefficient to upload large-sized meshes into a VR application directly. Manually modelling an accurate real-world environment can also be a complicated task, due to the large size and complex nature of the landscapes. In this research, a method to automatically convert 3D point-clouds of any size and complexity into a format that can be efficiently rendered in a VR application is proposed. Apart from reducing the cost on performance, the solution also reduces the risks of virtual reality induced motion sickness.

The pipeline of the system incorporates three main steps: a surface reconstruction step, a texturing step and a segmentation step. The surface reconstruction step is necessary to convert the 3D point-clouds into 3D triangulated meshes. Texturing is required to add a realistic feel to the appearance of the

meshes. Segmentation is used to split large-sized meshes into smaller components that can be rendered individually without overflowing the memory.

A novel mesh segmentation algorithm, the Triangle Pool Algorithm (TPA) is designed to segment the mesh into smaller parts. To avoid using the complex geometric and surface features of natural scenes, the TPA algorithm uses the colour attribute of the natural scenes for segmentation. The TPA algorithm manages to produce comparable results to those of state-of-the-art 3D segmentation algorithms when segmenting regular 3D objects and also manages to outperform the state-of-the-art algorithms when segmenting meshes of real-world natural landscapes.

The VR application is designed using the Unreal and Unity 3D engines. Its principle of operation involves rendering regions closer to the user using highly-detailed multiple mesh segments, whilst regions further away from the user are comprised of a lower detailed mesh. The rest of the segments that are not rendered at a particular time, are stored in external storage. The principle of operation manages to free up memory and also to reduce the amount of computational power required to render highly-detailed meshes.

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
API	Application Programming Interface
Bench	Benchmark
CAD	Computer-Aided Design
COE	Core Extraction
CD	Cut Discrepancy
CE	Consistency Error
FP	Fitting Primitives
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HD	Hamming Distance
HSL	Hue Saturation Lightness
HSV	Hue Saturation Value
KM	K-Means
LIDAR	Light Detection and Ranging
LOD	Level-of-Detail
NC	Normal Cuts
RC	Random Cuts
RGB	Red Green Blue
RGBD	Red Green Blue Depth
RI	Rand Index
RW	Random Walks
SD	Shape Diameter
SDF	Shape Diameter Function
TPA	Triangle Pool Algorithm
VR	Virtual Reality

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 CHAPTER OVERVIEW

This chapter provides context and an overview of the research conducted. Initially, the background of the problem is presented. The background focuses on how Virtual Reality (VR) can be applied as a visualisation tool for three-dimensional (3D) virtual environments. The background includes a brief description of how the virtual environments can be generated using 3D point clouds and meshes [1].

The problem statement then gives insight into the constraints associated with loading large meshes into 3D rendering engines that are used to develop VR applications. The constraints lead to the identification of the research gap and subsequent specification of the research objectives and questions. A description of how the research was conducted is given, including, the research hypothesis and the evaluation procedure. The chapter concludes by detailing the contribution to research.

1.2 VIRTUAL REALITY

1.2.1 Virtual reality applications for real-world simulations

A VR computer application simulates a real or imagined world environment [2]. The application may allow users to interact with the simulated environment as if they were in the actual environment. VR finds application in areas such as scene visualisation, ergonomics in the workplace, simulated driving lessons and real-world process simulations. For instance, in the mining industry, VR is used to visualise scenes from mines and to simulate mining processes [3]. Such simulated mining processes

may include drilling rig training, underground hazards aversion training, real-time process monitoring and rock testing [4].

1.2.2 Creating virtual reality environments from point clouds and meshes

3D rendering computer applications can be designed using 3D rendering engines such as Unity [5] and Unreal [6]. Even though the engines are mainly used to develop 3D games, they can also be used to develop other graphic applications that require 3D rendering. Such computer applications can be further developed into VR applications [7].

Designing the meshes of the 3D environments using third-party software is ideal when there is no need to produce accurate details of the environment, when the environment is not large and complex or if there are pre-existing CAD models of the environment that can be extended. Producing an accurate model of a large complex real-world environment, using such an approach, is not feasible due to human error and the laborious nature of the task. If a realistic model of the real-world environment is required, a method to extract an accurate 3D model of that environment needs to be adopted.

A possible solution involves using point cloud representations of real-world environments. A 3D point cloud is a grouping of points in a 3D (x, y, z) coordinate reference system, where each point is represented by (x, y, z) coordinates relative to a reference frame [1]. Applications that render meshes, generated from point clouds, can be designed using the Unity [5] and Unreal [6] game engines.

1.3 PROBLEM STATEMENT

1.3.1 Context of the problem

The performance of 3D rendering applications is affected by the size and number of meshes rendered at a particular time; an increase in any one of these parameters negatively affects the process' performance [5]. Using powerful computers could potentially solve the problem, however, they may be too expensive.

A second possibility would be to reduce the size of the mesh by point cloud simplification [8]. Simplification reduces the size of the point cloud by discarding a sample of the points. A successful simplification operation would create smaller-sized input meshes that would allow for improved processing performance. However, in cases where the real-world data associated with the landscape needs to be preserved, simplifying the point cloud could lead to the loss of valuable data. For instance, in the case of mining landscapes, vertices may hold more than just spatial, normal or colour information. The vertices could also be encoded with extra information to assist the mining processes. For instance, the geological data of a given landscape could be encoded into the point cloud's vertices, such that removing a large sample of the points by simplification would lead to the loss of the valuable data.

Rendering large terrains is a well-studied problem and state-of-the-art techniques use out-of-core algorithms [9] to manage the rendering system. Such approaches perform well in standard 3D applications, but certain aspects of their operation lead to virtual reality induced sickness [10] in VR applications, as will be discussed in Chapter 2.

1.3.2 Research gap

The discussion provided in the previous sections leads to a possible research gap to find a way to efficiently render large meshes of real-world environments in a VR visualising application. The research attempts to solve the problems that concern memory management when rendering large mesh data, as well as, problems of VR induced sickness that arise due to improperly designed VR applications.

The focus of the research is on building a preprocessing pipeline to transform the input data from real-world landscapes, into a format that is compatible with the efficient operation of a virtual environment visualising application. Since the input data to the preprocessing pipeline will be in the form of 3D point clouds, the research will address the problems that are associated with:

- Converting the point clouds into large textured meshes that are suitable for rendering in the VR application;

- Converting the resulting meshes into a format that facilitates the efficient operation of the VR application; efficient in terms of optimising the memory and run-time costs.

1.4 THE RESEARCH OBJECTIVES AND QUESTIONS

1.4.1 The research objectives

The objectives of the research are:

- To design a step by step pipeline that transforms a raw point cloud into a scene in a VR application.
- To develop new algorithms to solve the problems along the pipeline if no other existing algorithms can efficiently solve the problems.
- The algorithms in the pipeline should specifically solve surface reconstruction, texturing and mesh segmentation problems.

1.4.2 The research questions

For a point cloud to be used to represent a real-world environment in a VR visualising application, the point cloud needs to be converted into a suitable 3D model to be rendered with minimal cost on computational resources. The research aims to solve the problem by providing answers to the following questions:

1. How can the point cloud be converted into a 3D mesh that is suitable for VR rendering?
2. How can a texture be mapped onto the mesh's surface to make it visually realistic in VR?
3. How can the mesh be optimally rendered to reduce computational overhead during run-time?

1.5 HYPOTHESIS AND APPROACH

1.5.1 The hypothesis

A large point cloud of a real-world environment can be transformed into a mesh that can be used to simulate the environment in a VR application, by loading multiple segments of the mesh within proximity of the VR user.

1.5.2 The evaluation procedure

The evaluation procedure will focus on three areas, mesh generation, mesh texturing and mesh segmentation. The procedure will be as follows:

1. Generate meshes from point clouds of typically large landscapes, by using state-of-the-art surface reconstruction algorithms and identify the most appropriate method to adopt for the preprocessing pipeline.
2. Identify the different approaches to texture map creation and select a suitable approach to generate texture maps for the meshes.
3. Design a segmentation algorithm to partition the mesh into multiple segments. Show how the algorithm compares to state-of-the-art segmentation algorithms and motivate why the new algorithm should be used in the pipeline.

1.6 RESEARCH CONTRIBUTION

Contributions were made in two main areas; rendering of large meshes and 3D mesh segmentation. The research provided an architecture that efficiently renders large real-world environments. In contrast to other rendering architectures, the proposed architecture was designed taking into consideration the constraints introduced by VR applications; constraints including memory efficiency, processing power and health risks.

The research also contributed a novel 3D mesh segmentation algorithm. The algorithm was designed to accurately segment meshes of large landscapes. The algorithm can also be used to accurately segment

other types of mesh objects. The algorithm exploits the similarities between the triangular faces of the mesh and their contiguity, allowing the algorithm to avoid the need for using geometric attributes as features. The novel algorithm is, therefore, invariant to geometrical complexities that potentially exist in meshes of real-world landscapes. An article of the proposed work, titled 'Mesh segmentation-based rendering of real-world terrains for Virtual Reality visualizing applications', has been submitted to the journal, 'The Visual Computer'.

1.7 SUMMARY

This chapter introduced the research that was carried out and presented in the dissertation. The rest of the dissertation provides a comprehensive literature review in Chapter 2, the proposed solution in Chapter 3, the evaluation procedure in Chapter 4, the experimental work in Chapter 5, a comprehensive discussion in Chapters 6 and a conclusion in Chapter 7.

CHAPTER 2 LITERATURE STUDY

2.1 CHAPTER OVERVIEW

The initial discussion in this chapter introduces the concept of 3D rendering. The Unity [5] and Unreal [6] game engines are also introduced since they were used to design the VR application. The out-of-core rendering technique [9] is described and shown why it is not relevant to the research.

Since 3D polygonal meshes are used to model the environments in VR, a background on 3D point clouds and meshes [1] is given. A discussion on how the meshes are generated from real-world data then follows. The discussion includes the first step of generating a point cloud of the real-world, followed by how the point cloud can be transformed into a textured mesh by surface reconstruction and texture mapping methods.

In the research, 3D mesh segmentation is used to facilitate the rendering of the meshes at different levels-of-detail (LOD). The chapter, therefore, concludes with a discussion on segmentation techniques.

2.2 RENDERING REAL-WORLD ENVIRONMENTS IN VR

2.2.1 Rendering using the Unity and Unreal game engines

The process of synthesising 2D images by simulating how light behaves on 3D surfaces is known as rendering [11]. Rendering allows 3D representations to be visualised in computer graphics applications.

Unity [5] and Unreal [6] are two game engines that can be used to render 3D environments. Besides rendering capabilities, Unity and Unreal provide features such as physics, scripting, collision detection and artificial intelligence. Combining these features with 3D meshes, allows software developers to create computer applications that simulate real-world environments. The engines also have the capability of extending the 3D rendered environments into VR environments, hence, they can be used to create computer applications that simulate real-world behaviour in VR.

Pre-existing 3D meshes can be made available for rendering in an application before run-time. Such meshes can be created using the engines themselves or using third-party programs. During run-time, 3D meshes can also be made available to the engines by providing them with input such as:

- the vertices of the mesh;
- the per-vertex colours of the mesh;
- the per-vertex normals of the mesh;
- the image coordinates (UV) of the mesh's texture;
- the mesh's triangular faces.

Generating meshes in such a way is known as procedural mesh generation [5], [6] and it forms the basis of how the meshes are loaded into the VR application, as described in Chapter 3.

The performance of a rendering application can be affected by the size of a mesh; the larger the mesh, the larger the negative impact on performance [5]. A negative impact on performance potentially leads to:

1. Lagging; a slow response to input commands.
2. Loss of sync between the application program and the VR hardware, leading to the sporadic disappearing of VR visualisation.

A method to load large meshes with minimal cost to the application's performance, therefore, has to be identified.

2.2.2 Rendering large terrains using out-of-core methods

Section 2.2.1 concludes that a method needs to be introduced to facilitate the loading of large meshes into rendering applications. The following section provides an overview of how large terrain rendering management has traditionally been carried out.

Due to the typically large size of terrain data, powerful processors with large amounts of internal memory would potentially be required to directly load all the data into the game engines. Out-of-core or external storage algorithms use external storage to store data that is too large to store in internal memory [9]. The general work-flow of such techniques involves:

- Storing, in external storage, multiple versions of the terrain at different levels-of-detail (LOD);
- Determination of the field-of-view at each frame;
- Refreshing the scene by loading the stored version of the terrain, based on the field-of-view and position of the viewer.

The result is that, as the viewpoint changes, the application notes the position of the viewer, queries a hierarchical tree data structure for the LOD and performs view-frustum-culling. View-frustum culling is the selection of only the parts of the terrain that are in the field-of-view [9]. Such a work-flow ensures that when the viewer is close to a surface, the LOD of the terrain is higher than when the viewer is further away. The work-flow also ensures that only the regions of the terrain within the field-of-view are rendered at a particular time.

The work-flow of out-of-core algorithms could potentially be used to facilitate the loading of large meshes into the VR applications, however, the work-flow leads to unwanted VR induced sickness. VR induced sickness is a known phenomenon that may occur when using VR applications [10] and [12]. The effects of VR induced sickness are similar to the effects of motion sickness, such as dizziness, nausea and headaches. Experimental evidence [10] suggests that the causes of VR induced sickness include:

- Sensory mismatch; for instance, when the virtual world moves away from the user.

- Orientation mismatch; when the user is not oriented in the natural upright position relative to the virtual environment.
- Frequent updates of the environment; when the scene frequently changes.

The operation of out-of-core methods includes zooming and changing the LOD of the scene when the field-of-view changes. In a VR application, any head movement leads to changes in the field-of-view. If out-of-core methods were implemented, the frequency of users' head movements, which is typical in VR simulations, would lead to VR induced sickness.

Large terrain management systems that are used in a VR environment, therefore, need to avoid frequent changes of the LOD of the environment. Hence, to reduce the risk of VR induced sickness, the proposed solution does not trigger a change in LOD of the environment when a user's head is moved. Changing of the LOD is triggered only when the user teleports from one location to another as is discussed in Chapter 3.

2.3 AN OVERVIEW OF 3D POINT CLOUDS

3D meshes are generated using 3D point clouds. A point cloud is a sample of points, in a 3D coordinate reference system, that represents a real-world object or environment [1]. If a real-world environment can be treated, hypothetically, as being made up of spatial points, point clouds can be interpreted as a sample of the total points that make up that real-world environment.

Point clouds can be obtained by using various sensors, including RGB cameras and depth sensors. Various algorithms can be used to generate the final point cloud, depending on the type of sensor used. Methods that can be used to generate point clouds include:

1. Stereo vision [13]
2. Structure from motion (photogrammetry) [14]
3. RGBD sensor-based generation [1]
4. Light detection and ranging (LIDAR) sensor-based generation [15]

Sections 2.3.1 to 2.3.4 outline how the methods work.

2.3.1 Generation by stereo vision

Generating point clouds by stereo vision [13] involves using two or more cameras that:

- Have a known separation d;
- Have known internal camera parameters;
- Have identical focal lengths f;
- Have a geometrical set-up such that their optical axes are in parallel.

Suppose that the 3D coordinate of a point A, that is found in both images of the cameras, is to be calculated. A triangle can be formed by joining point A, the first camera's optical centre and the point at $\frac{d}{2}$. Similarly, another triangle can be formed for the second camera. Let the x coordinate of A be A_1 in the first camera's image plane and A_2 in the second camera's image plane. By using the similarity of the triangles, the depth, z, of A is given by (2.1),

$$z = \frac{df}{A_1 - A_2}. \tag{2.1}$$

By finding the 3D coordinates of all the corresponding points in the images produced by the two cameras, a point cloud that consists of such corresponding points can be formed.

2.3.2 Structure from motion

Point clouds can be generated by a photogrammetric process that uses multiple images of real-world surfaces [14]. Given a set of images, the steps taken to generate a point cloud are as follows:

1. Feature points between the images are detected;
2. The feature points are matched;
3. The matched features are triangulated by bundle adjustment [16], a non-linear technique that performs least-squares optimisation, to form a 3D point cloud.

The steps described above lead to the generation of a sparse point cloud as only the matched feature points are triangulated. An alternative approach involves matching every point in an image to a point in the image sequence before going ahead with the triangulation.

2.3.3 Generation by colour and depth sensors

Red-Green-Blue-Depth (RGBD) sensors are cameras that provide both a colour RGB image and a depth image [1]. An example of such a sensor is Microsoft's Kinect that was released in November 2010 [17]. The sensor uses an RGB camera to collect colour images and an infrared camera to collect the corresponding depth images. In a depth image, the values of the pixels represent the relative distance between the real-world positions and the camera that retrieved the image.

By combining a pixel's (x, y) coordinates in the depth image with its depth value z, a 3D point is formed. Iterating the process for all pixels in the depth image, a grouping of such points is formed, resulting in a point cloud. If multiple RGBD images of the same scene are taken from multiple different angles, multiple point clouds can be produced. By using the camera-positions of each of the RGBD images, the point clouds can be aligned and combined into one full point cloud through the process of registration [18].

2.3.4 Generation by LIDAR

LIDAR sensors include a laser transmitter and receiver pair [15]. To retrieve the 3D coordinates of a point on a surface, the transmitter sends a laser pulse towards a surface; when the laser pulse hits the surface, it is reflected to the receiver. The distance s between the sensor and the hit-point is given by (2.2),

$$s = \frac{ct}{2}, \tag{2.2}$$

where c is the speed of light and t is the time elapsed between transmission and reception of the laser pulse.

When integrated with a Global Navigation Satellite System (GNSS), the LIDAR sensor has access to the Global Positioning System (GPS) coordinates of the hit-point. Combining the GPS coordinates and the depth information results in the 3D coordinate of the hit-point. A point cloud is formed by combining the 3D coordinates of multiple hit-points from different parts of the real-world surface.

2.3.5 Additional information contained in the point cloud

Additional information can further be added to a point cloud's spatial information. Such information can either be retrieved directly from the sensors themselves or can be calculated from other attributes of the point cloud.

For instance, whilst the depth image provides the spatial data of a point, the corresponding pixel in the RGB image provides the colour information [19]. Therefore, if both the RGB and depth images are available, the colour can be retrieved and added to the point cloud. The RGB images can also be used to map textures to the point cloud [20].

The surface normal attributed to a point can be computed from the spatial data of its neighbours [21]. Such information can also be incorporated into the point cloud. Surface normals provide the physical orientation of each point and can thus be used when modelling various features of the point cloud in rendering engines. For instance, surface normals can be used to model how light affects the surface of the point cloud when placed in a lit environment.

2.4 POINT CLOUD SURFACE RECONSTRUCTION (MESHING)

2.4.1 Overview

Even though the VR scenes can be visualised in the form of point clouds, the point clouds need to be converted into meshes if they are to produce realistic scenes in 3D rendering engines. Meshes are formed when polygons, known as faces, are formed by connecting the vertices of a point cloud by edges. Faces add a realistic feel to the point cloud as it is transformed into a solid surface. Such a step is necessary since, as just point clouds, the scenes will be undesirably visualised as a collection of coloured points suspended in space.

Surface reconstruction methods infer a best-fitting polygonal mesh surface for a set of points. Since point clouds are a group of spatially sampled points, surface reconstruction methods can, therefore, be used to transform point clouds into meshes [22].

To achieve their task, surface reconstruction algorithms may require extra input information, other than the spatial information of the sampled points, such as, the points' surface normals, the scanner's specifications and the original images that the point clouds were extracted from.

2.4.2 Point cloud surface reconstruction algorithms

In the following sections, state-of-the-art surface reconstruction algorithms are discussed; providing a detailed description of how the algorithms work, their advantages and their limitations. The details of the particular algorithm that was selected for implementation in the preprocessing pipeline of the VR application are given in Chapter 3.

2.4.2.1 Delaunay triangulation

Delaunay triangulation involves joining a set of points to form triangles with maximised minimum angles [23]. The triangulation is performed such that no point lies in the circumcircle of each triangle, in the case of two dimensional (2D) spaces, or inside the circumsphere of the triangle, in the case of 3D spaces.

Delaunay triangulation is an effective way to create triangular meshes of point sets. However, it is rarely used as a surface reconstruction algorithm by itself, but rather as a triangulation tool during the implementation of other surface reconstruction algorithms. In contrast to other surface reconstruction algorithms, Delaunay triangulation acts on the raw vertices, whilst, other surface reconstruction algorithms attempt to infer the best fitting surface beyond triangulation [24].

Various approaches have been proposed to carry out Delaunay triangulation. In the flipping algorithm [25], for two triangles sharing an edge, the sum of the opposite angles that are not cut by the common edge should be less than or equal to 180° for the triangles to be Delaunay triangles. The property can be exploited to perform Delaunay triangulation over a set of points in what is known as a flipping algorithm. The steps that are taken by the algorithm are as follows:

1. The points are joined together to form triangles;

2. For two triangles that share an edge, the sum of the two opposite angles that are not cut by the edge should be less than or equal to 180°;
3. If their sum is more than 180°, the common edge is flipped such that it cuts through these angles. Since the angles of a quadrilateral add up to 360°, the two opposite angles that would not be cut by the common edge, would then be less than or equal to 180°, and the triangles would have transformed to Delaunay triangles;
4. The process is repeated for all the triangles in the mesh until all the triangles are Delaunay triangles.

The flipping algorithm has a long execution time in the order of $O(n^2)$ and has no guarantee of convergence. It can, however, be used in dimensions higher than 2D.

Incremental algorithms [24] are a variation of the flipping algorithm. Instead of initially triangulating the whole set of points at once, the triangles are formed incrementally with one vertex being introduced to the mesh at a time. Flipping of the edges is performed after each vertex addition. These algorithms have execution times in the order of $O(n)$. The algorithms can be used in dimensions higher than 2D.

In the divide and conquer algorithm [24], the point set is recursively split into two parts. Triangulation is then performed separately between the two sets before they are merged. The algorithm has an execution time of $O(n\log(n))$. The algorithm, however, only works in 2D space.

2.4.2.2 Marching cubes

The Marching cubes algorithm [26] reconstructs the surface using a voxel grid of the point cloud. It uses a hypothetical cube that moves throughout the voxel grid of the point cloud, using a lookup table to determine which triangular face should be placed in a particular region.

The voxel grid is made up of adjacent same-sized cubes that cover the area in which the point cloud is located. Each vertex of the cube is labelled as occupied or as unoccupied, depending on the occupancy of the point cloud.

The lookup table is comprised of 256 possible triangle combinations in 3D space. The triangles in the lookup table are structured in such a way that they correspond to every possible combination that the marching cube could have encountered. Each combination that the eight vertices of the marching cube exhibit when placed in a cube of the grid, determines which triangle is going to be selected from the lookup table. Every cubic area in the voxel grid is replaced by the corresponding triangle selected from the lookup table.

The algorithm is easy to implement. However, to produce surfaces with a high resolution, the size of the cube that forms the grid needs to be small, which corresponds to the need for more processing power.

2.4.2.3 Voronoi filtering

Voronoi filtering [27] is a surface reconstruction algorithm that exploits Delaunay triangulation and Voronoi diagrams [28]. Given a set of points, if a network of regions is formed such that each point is enclosed by a region closest to it in space, the representation is called a Voronoi diagram, and each point is referred to as a Voronoi vertex. The Voronoi diagram of a set of points is the dual graph of the Delaunay triangulation of the points.

If the points in a point cloud are taken as both the Voronoi vertices of a Voronoi diagram and also as the vertices of a Delaunay triangulation network, the algorithm attempts to reconstruct the surface by eliminating the Delaunay triangles that distort the required surface. The algorithm takes the following steps:

1. A 3D Voronoi diagram of the point cloud is created;
2. For each Voronoi vertex:

   (a) If the Voronoi vertex does not lie on the convex hull of the point cloud, the furthest point from the Voronoi vertex in its Voronoi cell, the pole p^+, is defined;

   (b) If the Voronoi vertex lies on the convex hull, the pole p^+ is a point at infinite distance in the direction equal to the average of the outward normals of the hull faces, meeting at the Voronoi vertex;

(c) p^- is defined as the furthest point from the Voronoi vertex in the opposite direction to that of p^+.

3. A Delaunay triangulation is created for the new points, consisting of the Voronoi vertices and all the poles except for those poles, at infinite distances.
4. Any triangle that has a pole as any of its vertices, is discarded from the network.
5. The triangles and vertices that remain in the network, form the reconstructed surface.

The algorithm is efficient for practical use and does not depend on user-specified parameters. However, if the point cloud density is low, holes may appear in the reconstructed surface.

2.4.2.4 The Ball Pivoting algorithm for surface reconstruction

The Ball Pivoting algorithm for surface reconstruction [29] reconstructs a surface from a point cloud by traversing an imaginary sphere (ball) across the point cloud. If three points in the point cloud touch the edge of the sphere whilst no other point is contained within the sphere, the points become vertices of a valid triangular face of the surface.

Initially, an arbitrary triangle made up of three points from the point cloud is selected. A sphere is then revolved around one of the edges of the triangle, ensuring that two points from the triangle are kept on the edge of the sphere. If during a revolution, a point is touched by the surface of the sphere and no other points lie inside the sphere, a valid triangle is formed. The process is repeated until all applicable edges have been considered. A new seed triangle is selected and the process is repeated until the whole point cloud has been traversed.

To select a seed triangle, a point that would not yet have been involved in the reconstruction algorithm is considered. Two other points in its neighbourhood are selected to complete the triangle. These three points should have the same normal vector.

The algorithm can reconstruct surfaces of large point clouds efficiently. However, a correct selection of the ball radius has to be done manually by the user, since the radius of the sphere has to be kept small for evenly dense point clouds and large for uneven point clouds.

2.4.2.5 Poisson surface reconstruction

The Poisson surface reconstruction algorithm [30] attempts to infer a surface, S, of a point cloud by solving a Poisson equation. Suppose that an indicator function, x, is defined with values 1 inside of S and 0 outside of S. Since x is a piecewise continuous function, its gradient is a vector field [14]. Since x is 0 outside of S and 1 inside of S, changes in x only occur on the surface S; meaning that the gradient of x, ∇x, is 0 everywhere else, except on S. The values of ∇x on S are equal to the corresponding values of the surface normal vectors acting on S [14]. Equation (2.3) shows the relationship.

$$\nabla x = \mathbf{V}, \tag{2.3}$$

where $\mathbf{V}$ is the vector field of the surface normals.

Applying the divergence operator, (2.3) reduces to (2.4),

$$\therefore \Delta x = \nabla \cdot \mathbf{V}, \tag{2.4}$$

where Δ is the Laplacian operator and $\nabla \cdot \mathbf{V}$ is the divergence of $\mathbf{V}$.

From (2.4), the function x can be found given the normal vector field, $\mathbf{V}$, of the point cloud. By subdividing the space into a voxel grid, the function x can be approximated discretely within each voxel, by finding the function x from the normal vectors of the points within each voxel. The function x is stored in an octree data structure and the Marching cubes algorithm [26] is used to approximate the surface S from each of the functions stored in the octree. Increasing the octree size will increase the LOD of the surface but is memory and computationally expensive.

2.4.2.6 Fast surface reconstruction

The Fast surface reconstruction algorithm [31] is designed to triangulate large point cloud datasets in a short amount of time. A seed triangle is initially identified. The seed is an arbitrary triangle with vertices represented by the coordinates of three points in the point cloud. To construct the triangles of the surface: for a point x, a k neighbourhood of points is constructed by selecting the k nearest neighbours contained in a hypothetical sphere centred on x. The radius of the sphere is dependent on the local point cloud density. For a point cloud with non-uniform densities, the regions of the point cloud with lower densities are sampled to generate more points to make the region uniform. The neighbourhood is then projected onto a plane that is tangential to the neighbourhood's surface

and ordered around the x. Triangles are then formed by point pruning [32]. A seed is selected and triangulation proceeds from the seed triangle's points to the rest of the points, with the process being iterated until each point has been processed and no triangles can be formed. The method can handle both uniform and non-uniform point clouds. However, the radius of the sphere in the triangulation algorithm has to be manually selected by the user for optimum surface reconstruction results.

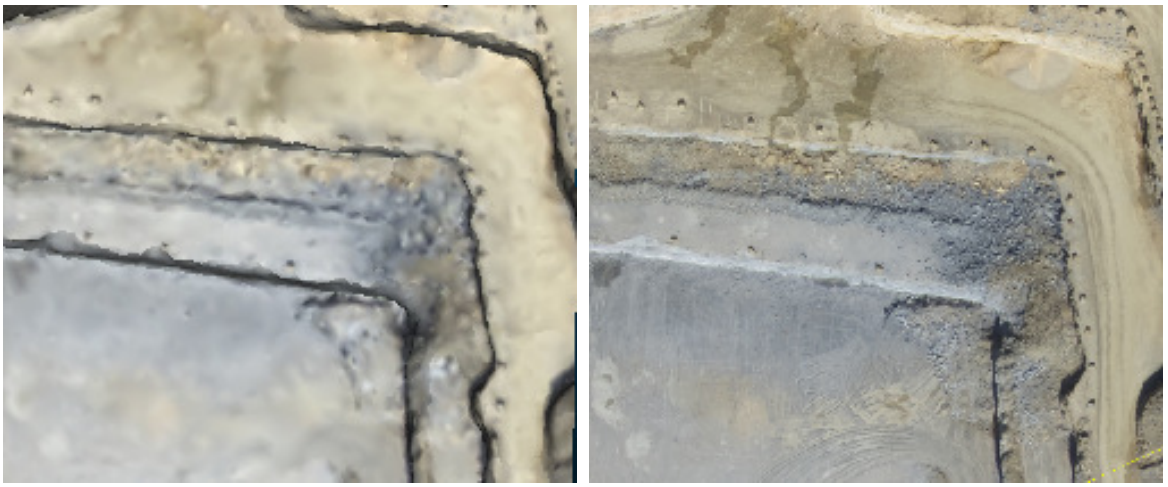
2.5 TEXTURING

2.5.1 Overview

The term texture may either be used to refer to the tactile feel of the surface or the visual appearance of the surface [33]. In the context of the research, texture only refers to the visual appearance of the surfaces.

The reconstructed mesh gives an approximation of the true shape of the actual environment; however, extra detailing of the mesh can be achieved by the addition of a texture onto the mesh [5]. In most computer graphics applications, assigning a texture to a mesh is the only way that high details are represented [34]. Texturing has the advantage of increasing the visual richness of the mesh without a significant increase in the required computational power [33]. Figure 2.1 shows a comparison of a vertex-coloured mesh with a textured mesh generated by Pix4Dmapper [35], using the Swiss quarry dataset [36].

Both meshes shown in Figure 2.1 are identical in all aspects except for their textures. The mesh shown in sub-figure 2.1(b) gives an impression of having a higher LOD than the mesh shown in sub-figure 2.1(a). Texturing, therefore, adds an aesthetic appeal to the reconstructed surface. Texture creation methods create 2D texture map images from 3D polygonal surfaces that form a mesh. The process of mapping a 3D object space onto a 2D texture space is known as UV mapping, where (u, v) coordinates are used to represent the 2D texture map's coordinates [33].



(a) A vertex-coloured mesh **(b)** A textured mesh

Figure 2.1. A comparison of a vertex-coloured mesh with a textured mesh generated by Pix4Dmapper [35], using images from the Swiss quarry dataset [36].

2.5.2 UV mapping

Texture mapping for complex 3D meshes is too complex to be carried out by hand; hence, various methods have been attempted to automate the process [37]. Creating texture maps involves a UV mapping or parameterisation step that maps a 2D texture surface onto a 3D texture [38]. Equation (2.5) shows an affine mapping of 2D (u,v) texture coordinates to 3D (x,y,z) mesh coordinates.

$$\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \\ a_3 & a_6 & a_9 \end{bmatrix} \tag{2.5}$$

Specifying the triangle vertices x_i, y_i, z_i and their corresponding texture coordinates u_i, v_i for $i = 1,2,3$ is sufficient to compute the mapping parameters $a_1 ... a_9$. To avoid errors due to distortions of the camera, (2.5) can be modified to (2.6).

$$\begin{bmatrix} xw & yw & zw & w \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \\ a_3 & a_6 & a_9 & 1 \end{bmatrix} \tag{2.6}$$

Parameterisation provides 3D rendering engines with information of how to map the texture image onto the 3D mesh during rendering.

2.5.3 Texture map creation

In Blending images for texturing 3D models [39], the input is a 3D mesh and its corresponding images, taken by a camera with known intrinsic and extrinsic parameters. The output comprises a texture map and the UV coordinates of each vertex in the triangular mesh. Each image is divided into different frequency bands that are projected onto the texture map. Bands with a low frequency are passed through an averaging filter, whilst higher frequency bands are passed through a non-linear filter. When all the images have been processed, the frequency bands are recombined into one final texture map.

The TextureMontage algorithm [37] creates multiple texture maps for a single 3D mesh. Given multiple images of a 3D mesh, the user initially specifies feature correspondences that match the vertices of the mesh. An optimisation procedure is then used to compute the texture coordinates of the mesh's triangles. For regions that are not mapped during the optimisation process, due to failure of feature matching, for instance, a Poisson-based interpolation technique is used to fill in the missing sections.

Mesh segmentation can also be used to create the texture map by initially parameterising the mesh by mapping its 3D surface to a 2D texture surface in a one-to-one relationship [40]. The parameterised surface is output in the form of patches. From multiple input images of the 3D mesh, regions are identified that can be blended onto each patch of the parameterised 3D surface. The mapped patches are extracted and merged into the texture map to form a texture atlas.

Another method associates each polygon from the mesh with one image from a set of multiple images [38]. The boundaries of the triangles created on the texture map may have distortions, since they may have been assigned to different images. A filtering technique that involves averaging the texture between adjacent triangles, is used to remove the edge distortions.

2.6 POINT CLOUD SEGMENTATION

The preprocessing pipeline that is discussed in Chapter 3 incorporates a mesh segmentation step. The segmentation step breaks up a large mesh into multiple smaller segments. The smaller segments are used to facilitate the refreshing of the LOD in selected areas of the virtual environment during run-time

of the VR application. The following section discusses point cloud segmentation, and Section 2.7 discusses mesh segmentation.

2.6.1 Overview

Point cloud segmentation involves grouping the points of a point cloud into multiple meaningful similar regions [41]. Specific sets of features that characterise the point cloud are used to distinguish one segment from another. Segmentation can be modelled by (2.7),

$$S = S_1 \cup S_2 \cup S_3 \cup ... \cup S_n, \tag{2.7}$$

where S is the full point cloud and $S_1 - S_n$ are the disjoint subsets (the segments) of S.

The goal of segmentation algorithms is to extract the subsets $S_1 - S_n$ from S. Segmentation is an important topic in computer vision since it is an important first step in other algorithms such as object recognition. In such an application, the segmentation algorithm is responsible for breaking down a point cloud into multiple parts that are subsequently set as the input to a recognition algorithm.

Segmentation algorithms include two main tasks: The first, to identify the type of similarity features which should be used to separate the segments and the second involves the actual grouping of the points, based on the features. Various approaches have been proposed to achieve these two tasks.

Edge detection methods identify the borders of the segments that make up the point cloud [41], [42] and [43]. The principle of the algorithms is the assumption that the feature representations of each point on the edge of a segment are similar. Different kinds of similarity features are used for edge detection algorithms. By fitting 3D lines to groups of points in the point cloud, the gradient and hence, the unit normal vectors of the points, can be calculated. The changes in orientation of the unit normal vectors signify the presence of boundaries between the points [44]. The variation in the curvature of the points can also be used to detect boundaries of the point cloud [45]. Edge detection methods have the advantage of being fast but are affected by noise and uneven density in point clouds [41].

Region growing methods exploit the similarities of features amongst neighbouring points to subdivide the point cloud [46]. There are two types of the method: seeded region growing and unseeded region growing [41]. In seeded region growing, an initial set of seed points is selected. Regions are

grown from these seed points by adding the seeds' neighbouring points that have similar properties [42]. Region growing is eventually terminated when a specific criterion has been met [41] and [43]. The seeds need to be accurately selected since incorrectly selecting them may lead to over or under segmentation. In unseeded region growing methods, the point cloud as a whole is initially set as one segment. The segment is further subdivided into smaller regions by grouping points according to a similarity property [41]. The methods handle noisy point clouds well; however, they manage object border regions poorly.

Model-based segmentation methods subdivide the point clouds according to the shapes that they mathematically represent [41]. Points that conform to the same mathematical representation are grouped into the same segment. Similar to edge-based methods, model-based methods are also fast.

Graph-based methods transform the point cloud into a graph representation. Each point in the point cloud is represented by a node and is connected to other nodes in the graph by a set of edges [41]. The algorithms that are based on such a principle determine valid and invalid edges between the nodes. The validity of an edge is dependent on the similarity between the connecting nodes. Nodes that have valid edges are placed in the same segment. Algorithms that determine the validity and invalidity of the edges have at times been implemented by well-known algorithms, such as Frequency Hopping [47], k-nearest neighbours [48] and inference methods, such as Conditional Random Fields [49].

2.6.2 Similarity metrics used in segmentation

Graph-based methods and region growing methods exploit the local similarity of the points of the point cloud to perform segmentation. The proximity of the points combined with their planarity can be used as the similarity metric [46]. In such an approach, seed points are chosen based on a physical characteristic, such as their curvature, and the regions are grown, based on a nearest neighbour search of their planar similarity. An equation of a plane is solved for each point's coordinates, and points falling within the same values are grouped into one segment. The method does not handle noise very well and is slow [41]. Such an approach also poorly segments objects that are not man-made since they do not have clearly defined planes [50].

The normal orientation attribute of the points can be used as a similarity metric in point cloud segmentation [51]. Points are segmented based on their orientation to one other. The attribute works best as a similarity metric when segmenting geometrically planar surfaces.

Other approaches use vertex colours as the segmentation's similarity metric [52] and [53]. If the vertex colours are present in a point cloud, the similarity in their colours can be used to group them into similar segments. Colour-based segmentation methods will be looked at in more detail in the next sections.

2.6.3 Colour-based segmentation

Colour-based segmentation methods use the colour property of the points as the primary component of the similarity metric [52] and [53]. The motivation for using the colour property is to avoid using geometrical properties such as normals or curvature, that need to be calculated initially from the point cloud data itself, usually in a time-consuming process. Geometrical properties tend to work well in segmenting point clouds of regularly shaped planar surfaces but perform poorly in segmenting irregular complex structures [52].

2.6.3.1 Colour spaces

Different colour-based segmentation algorithms may use colour values that are represented in different colour spaces. A colour space defines how colours are organised and defined [54]. There are various colour spaces and common ones include:

- Red-Green-Blue (RGB);
- Hue-Saturation-Value (HSV);
- Hue-Saturation-Lightness (HSL);
- The International Commission on Illumination's LAB colour space, where L represents lightness, A represents the range of colours from green to red and B represents the range from blue to yellow.

The RGB colour space defines all the colours that can be formed by mixing any of red, green or blue. All the other colour spaces can be derived from the RGB colour space.

The HSV [54] colour space defines colours using three properties: hue, saturation and brightness. Hue represents what is commonly called colour. For instance, all blues have the same hue value with disregard to their shade. Saturation refers to the colour's intensity. The brightness defines how light or dark a colour is and a brightness value of zero will always produce a black colour. The HSL [54] colour space is similar to HSV, bar that, instead of the brightness, it considers the luminance of a colour. The luminance takes into consideration the sensitivity of the human eye to certain colours. For instance, the human eye is more sensitive to yellow than to blue.

The CIELAB colour space represents colour values in accordance with the human perceptual vision. Colours that look alike, according to the human eye, are numerically close together. It is represented by its lightness (L), green-red component (A) and its blue-yellow component (B) [54].

2.6.3.2 Colour-based segmentation of point clouds (The Zhan algorithm)

The algorithm has two main stages: a region growing stage and a region merging stage [52]. During the region growing stage, initially, all the points are unlabelled, where a label represents a particular segment. The first unlabelled point is assigned a new label; the point acts as the seed of the new region. The k-nearest neighbours of the seed are retrieved, where k, the number of nearest neighbours, is user-defined. All the nearest neighbours that are similar to the seed are assigned the seed's label. The process is iterated until there are no more unlabelled points. An over-segmented point cloud is returned.

The second stage is similar to the first stage; however, instead of iterating through points, the iteration is through the regions that were formed in the first stage. If the sizes of similar neighbouring regions are below a specified size, they are assigned to the same label, resulting in the merging of the over-segmented regions that were produced in the first stage.

The similarity metric that is used in the algorithm is represented by (2.8),

$$SM(C_i, C_j) = \sqrt{(R_i - R_j)^2 + (G_i - G_j)^2 + (B_i - B_j)^2}, \tag{2.8}$$

where SM is the similarity metric function, C_i is the RGB colour value of the i^{th} point being compared, and R_i, G_i and B_i are the red, green and blue values of C_i, respectively. The similarity metric computes a Euclidean distance between the RGB colour values of the points. If the distance is larger than a set threshold, the points are deemed not to be in the same segment.

The algorithm was tested on a coloured point cloud of an ancient Chinese building. The results showed that it performed better than a geometrically-based algorithm. The algorithm managed to separate regions of different colours well, and over-segmentation effects were minimal. The algorithm struggled to achieve a satisfactory global segmentation effect, due to the non-conformity of the RGB colour space to the human eye's perception of colour.

2.6.3.3 Graph-based segmentation for coloured 3D laser point clouds

A graph-based approach to 3D point cloud segmentation is proposed in [53], where the points of the point cloud are the vertices and the edges are weights that measure the similarity between two connected vertices.

Initially, each point is assigned to its own segment. For each point, a graph is formed between itself and the eight nearest neighbouring points. If any of the weights in the graph are below a predefined threshold, the segments holding those two points are merged. The result is a segmented point cloud.

Two similarity metrics were used with the algorithm, namely, the colour and the unit normal vector of each point. Equation (2.8) gives the colour similarity metric function that was used for the algorithm and (2.9) gives the second similarity metric, based on the unit normal vectors of the points.

$$SM(N_i, N_j) = arccos(n_i \cdot n_j), \tag{2.9}$$

where $SM()$ is the similarity metric function, N_i is the i^{th} vertex being compared and n_i is the unit normal vector of the i^{th} point being compared.

The similarity metric shown in (2.9) compares the similarity between the directions of the unit normal vectors of the points. If the angle between them is over a certain threshold, the points do not belong to the same segment. Since two similarity metrics were involved in the algorithm, two segments could be merged if, and only if, both weights were below their respective thresholds.

The algorithm performed well on indoor scenes. The introduction of the second similarity metric of unit normal vectors improved the algorithm. The algorithm, however, struggled with outdoor scenes due to the irregular nature of such environments, where vertices in the same segment could have an irregular unit normal vector distribution.

2.7 MESH SEGMENTATION

2.7.1 Overview

Point cloud segmentation algorithms are not able to segment meshes. However, since a mesh segmentation step is part of the system's pipeline in the VR application, an algorithm that can segment meshes of landscapes needs to be identified or designed. If a point cloud has been successfully reconstructed into a mesh, it can be segmented, using not only the mesh's vertices but also its polygonal face structure. The following section gives an overview of mesh segmentation techniques.

Various approaches to mesh segmentation adopt techniques that have already been discussed for point cloud segmentation; the major difference being that, instead of using only the vertices, the faces are also used. Region-growing methods extended for 3D meshes are proposed in [55], [56], [57] and [58].

Another method that is extended from the model-based segmentation of point clouds, models the mesh as a distribution of electrical charge [59]. Areas with a low charge density represent areas with deep concavities and areas with a high charge density represent areas of convexity. The algorithm then extracts the segments by tracing the areas with a low charge density. Such an approach has an advantage in that it avoids local characteristics of the mesh, such as the curvature, therefore, reducing its sensitivity to noise. However, if concavities exist in other areas of the mesh apart from the segmentation areas only, false segments will be created. The curvatures can be used to explicitly extract the boundaries of a mesh by using the minimum negative curvature [60].

Watershed-based algorithms imitate the way water fills up a geographical basin [43]. The segmentation regions are represented by the areas where flood points meet. The extraction of basins is achieved by finding minimum points of functions that represent the mesh and denoting them as the base of the basins where the hypothetical water can be filled from. Usually, the curvature is used as the function to represent the surface [43].

Mesh segmentation can also be carried out by clustering techniques [43]. Such methods initially cluster the faces of the mesh into separate regions; followed by grouping together similar faces in each region. The *k*-means [61] and hierarchical clustering [62], [63] algorithms are clustering-based mesh segmentation methods.

Spectral analysis methods segment the mesh by using the eigenvalues of matrices, constructed from the connectivity of a graphical model of the faces [43]. An affinity matrix can be formed with elements $w_{i,j}$ that represent the likelihood that polygonal faces i and j are in the same segment [64]. A second matrix is created, consisting of the largest eigenvectors of the normalised version of the first matrix. *k*-means clustering is then used to group the row vectors of the second matrix, based on a Euclidean distance metric.

2.7.2 State-of-the-art algorithms in mesh segmentation

The following section discusses, in more detail, seven state-of-the-art algorithms in 3D mesh segmentation. These algorithms are discussed since they were included in a mesh segmentation benchmark evaluation procedure [65] that was used to evaluate the research's proposed segmentation algorithm. The evaluation will be further discussed in Chapters 3 and 4.

The *k*-means algorithm [61] segments the 3D mesh by iteratively clustering the faces. Initially, the size of k, the number of segments, is defined by the user and a set of k seed faces is selected from the faces. During an iteration, each face in the mesh is assigned to a cluster containing its nearest seed. The distance is calculated using the dihedral angle between faces and the physical distance between them. After an iteration, the seed is adjusted to be the face in the centre of each cluster. The process is iterated until convergence.

For the Fitting Primitives algorithm [62], every face is initially assigned to its own segment. For every pair of adjacent segments, a geometric primitive, sphere, plane or cylinder is approximated and a merger is carried out on the best-fitting segments. The merging of adjacent segments is repeated until a user-defined number of segments are reached.

The Normalised Cuts algorithm [66] also performs hierarchical clustering to achieve segmentation. Each face is initially placed in its own segment before segments are merged by a normalised cut cost, defined by the sum of each segment's perimeter and divided by each segment's area. Merging occurs until a predefined number of clusters are reached.

For the Random Cuts algorithm [66], the mesh is initially reduced to a predetermined number of triangles. By starting with every face being contained in one segment, the algorithm iteratively splits the faces, computing the random cuts for each segment and selecting the cut that minimises the normalised cut cost. Iteration proceeds until a predefined number of segments are reached.

For the Random Walks algorithm [67], initially, a set of seed faces is chosen from the faces of the mesh. The size of the set of faces determines the final number of segments that will be produced. Every face is then assigned to the seed face to which it has the highest probability of reaching through a random walk. The segmentation is refined by combining adjacent segments, based on their lengths and perimeter.

The Core Extraction algorithm [68] transforms the mesh into a pose insensitive representation by using multi-dimensional scaling. The algorithm proceeds by iteratively extracting the main features in the segment until there are no more features to extract. The faces are then grouped based on the features.

The Shape Diameter algorithm [69] proposes a Shape Diameter Function (SDF), a metric that represents the diameter of an object's volume surrounding a point on the surface of the mesh. By computing the SDF for each face, a Gaussian mixture model is fitted on the histogram of the faces' SDF values. The Gaussian mixture model fits a set of Gaussians of size n. Each face is represented by a vector that stores the probability of each face belonging to any of the n clusters. By combining the vectors with the local curvature values of the mesh's faces, a graph cut algorithm is used to minimise an energy function; thus creating the different segments.

2.8 SUMMARY

The literature presented in this chapter provides a comprehensive analysis of the content associated with the proposed research. The state-of-the-art approaches that are used to reconstruct point clouds, texture meshes and segment meshes have been identified. Chapter 3 provides insights into the selection procedure of the methods that were used to build the preprocessing pipeline of the VR application. A novel algorithm, provided as a solution to 3D mesh segmentation, is also presented.

CHAPTER 3 APPROACH

3.1 CHAPTER OVERVIEW

This chapter provides the approaches that were taken to fulfill the research objectives. The research objectives involve designing a preprocessing pipeline of a VR application that allows a user to virtually visualise and interact with real-world environments. The purpose of the preprocessing pipeline is to convert large point clouds that represent real-world environments into a format that is suitable for rendering in the VR application. Since the virtual environment is generated from meshes of real-world environments, the point cloud needs to be converted into a textured mesh.

The Unity [5] and Unreal [6] rendering engines were used to create the VR application. Due to performance issues that arise with rendering large textured meshes using Unity and Unreal, this chapter also presents the approach that was taken to handle performance overhead issues. The approach led to the design of a novel mesh segmentation algorithm.

3.2 OPERATION OF THE VR APPLICATION

The environments that are rendered in the VR application are made up of polygonal meshes. In an ideal case, a large mesh of a real-world environment with a high LOD can be rendered directly into the VR application. However, if the mesh's size is almost as large as the host system's internal memory, the performance of the application will be negatively affected [5]. The negative impact on the performance is characterised by poor processing speeds that lead to lagging during run-time and will eventually lead to VR induced sickness. If the mesh's size is greater than the internal memory, the mesh loading will fail. Sections 3.2.1 to 3.2.4 illustrate the work-flow of the proposed VR application, specifically on how it maintains a smooth operation, whilst rendering meshes with large polygonal face counts.

3.2.1 Loading the low-resolution mesh

Suppose that two textured versions of the mesh that forms the virtual environment exist; the original (high-resolution) mesh with a higher LOD and a low-resolution mesh, a version of the original mesh with a lower LOD. Initially, both meshes are stored in external storage; when the VR application starts, the low-resolution mesh is loaded into the scene. The low-resolution property means that the mesh has a lower polygon face count than the original mesh. The small size of the mesh means that it can be directly loaded into the VR application without negatively affecting the performance of the application. The mesh is loaded into the program by the following sequence of steps:

1. Retrieve the spatial data from the external storage;
2. Retrieve the normals data from the external storage;
3. Retrieve the per-vertex UV coordinates from the external storage;
4. Retrieve the reference to the texture map from the external storage.
5. Load the data to a *procedural mesh component* of the procedural mesh API of either Unreal or Unity;
6. Render the mesh in the engine.

Even though the mesh is textured, the low polygonal face count means that the LOD of the mesh is still relatively low.

3.2.2 Increasing the LOD

A VR user is simultaneously placed into the scene when the application starts. To make the scene more realistic, the VR user needs to be exposed to a detailed view of the mesh. To achieve such a condition without affecting the performance and compromising the speed of the VR application, only the areas that are close to the user can be rendered with a higher LOD, whilst the rest of the environment is rendered at a lower LOD. The following steps show how such a process can be done:

1. When a VR user is initially placed at a starting position within the VR environment, segments extracted from the original high-resolution mesh that are within a radius r of the user's location, are rendered into the program from external storage;

2. The section of the low-resolution mesh falling within the radius r of the starting location is changed to a hidden state.
3. When a user teleports from one point to another within the low-resolution mesh, the landing location is noted;
4. The load is repeated, segments extracted from the original high-resolution mesh that are within a radius r of the landing location, are rendered into the program from the external storage;
5. The section of the low-resolution mesh falling within the radius r of the landing location is changed to a hidden state.
6. The segments that had been loaded in the previous location are deleted from the system and the corresponding hidden section of the low-resolution mesh is returned to a visible state.

The result is that a region of radius r from where a user is located within the VR scene will have a higher LOD than the rest of the scene. By adjusting r, the region with the higher LOD can be restricted to within a desirable field of view of the user. The net effect is that areas that are closer to the user have a higher LOD and those further away have a lower LOD. The total number of segments that can be rendered at a particular time is restricted by r, hence r has to be set in such a way as to prevent the loading of too many segments.

As discussed in Chapter 2, other out-of-core methods update the LOD when the field-of-view changes; if such an approach were to be taken for the VR application, two problems would arise:

1. The field-of-view of the VR user would include surfaces of the environment that were far away from the user such that it would not be necessary for those areas to have a higher LOD. Effectively, the memory space of the system would be used unnecessarily.
2. If a user were to constantly look around the environment (which is typical), the field-of-view would constantly change, triggering an update of the environment for every head movement; placing the user at risk of experiencing VR induced sickness.

3.2.2.1 The approach to load multiple segments

The segments of the high-resolution mesh reside in external storage since external storage is typically large enough to store large sums of data. When the VR application starts, a concurrent background

application is also started. The background application is in constant communication with the VR application through a network socket and is responsible for sending details of the multiple segments to be loaded at a particular time.

When a user teleports in the VR application, a message, containing the teleport-location, is sent to the background application. The background application selects the segments that are within a predefined radius r of the teleport-location and responds with the details of all the segments to be loaded. The segments are selected and loaded from external storage.

Such an architecture ensures that the multiple segments are not stored in the VR application's internal memory; hence, increasing the amount of memory available for the VR application's operations. The segments to be loaded are also selected by the background application, instead of the VR application, reducing the computations that the VR application needs to do.

Since the segments are stored and selected by the background program from external storage, the operations of the background program do not lead to internal memory congestion. In essence, the segments of the high-resolution mesh can take up space that is larger than the internal memory of the system since only a fraction of the segments is loaded into internal memory at a given time. As a result, the architecture allows the rendering of high LOD meshes that would not have been possible to render if they were supposed to be loaded directly.

The architecture allows the high-resolution segments to reside in a central server that can be accessed by multiple client computers on a network. This provides the benefits of:

1. Easing the external storage requirements for the client;
2. Multiple VR applications using the same real-world data;
3. Remotely accessing the real-world data.

However, a loading overhead can result if large data is transmitted over the network. The amount of data that is sent over the network is controlled by setting r to an optimal value that restricts the size of the data that is provided in a single transmission.

3.2.3 How the operation reduces risks of VR induced sickness

The low-resolution mesh provides a constant ever-present landscape in the scene. Such a scenario ensures that every time a user moves their head and enters a new field-of-view, the LOD of the scene is not changed, reducing the risk of VR induced sickness. Rendering of new meshes only occurs when the user teleports from one location to another.

3.2.4 Requirements to achieve the operations of the VR application

The steps that are taken to transform the point cloud of a real-world scene into a format compatible with the VR application, are as follows:

1. Point cloud surface reconstruction: The point cloud has to be transformed into two meshes that accurately represent the real-world landscape. A low-resolution mesh (with fewer vertices and faces) and a high-resolution mesh;
2. Surface texture mapping: Both meshes have to be textured to ensure that the meshes are visually detailed;
3. Segmentation: The high-resolution mesh of the landscape has to be segmented so that only parts of it can be loaded into the scene at a given time. A semantic segmentation approach is taken to allow the tracking each segment's identity. This will allow for object recognition (classification) during run-time of the VR application.

Sections 3.3 to 3.5 provide an analysis of the methods that can be used to build the preprocessing pipeline that satisfies the aforementioned requirements.

3.3 SURFACE RECONSTRUCTION

Three criteria need to be met by a surface reconstruction algorithm to make it suitable for the preprocessing pipeline:

1. It should preserve RGB colour information since the segmentation algorithm that will be introduced in Section 3.5 depends on the RGB vertex-colours;

2. It should produce a surface that has no holes to ensure that the resulting VR environment has a natural resemblance to the real-world;
3. It should produce a surface that accurately and visually represents the point cloud.

Six state-of-the-art surface reconstruction algorithms were discussed in Chapter 2, these include:

1. Delaunay triangulation.
2. Marching cubes.
3. Voronoi filtering.
4. Ball Pivoting algorithm.
5. Poisson surface reconstruction.
6. Fast surface reconstruction.

The selection of the surface reconstruction algorithm to be implemented in the preprocessing pipeline was carried out by an elimination process. Each of the state-of-the-art algorithms was eliminated if it failed to meet a requirement of the pipeline.

Sections 3.3.1 to 3.3.3 provide qualitative experiments that were conducted to select a suitable surface reconstruction algorithm that could be used in the system's pipeline. It should be noted, however, that the following experiments were not the formal evaluation of the research implementations, but were experiments that were used to select the best approach that would then be used in the formal evaluation in Chapter 5. The Marching cubes, Voronoi filtering and Ball Pivoting algorithms were evaluated using Meshlab [70]. The Delaunay triangulation and Poisson surface reconstruction algorithms were evaluated using CloudCompare [71]. To evaluate the Fast surface reconstruction algorithm, the Point Cloud Library [72] was used.

3.3.1 Test for colour preservation

In the following experiment, the ability of the surface reconstruction algorithms to preserve the colour property in the resulting meshes was tested. During reconstruction, a sample of the points may be lost, for instance, during voxelisation in the Marching cubes algorithm [26] or when discarded in the Voronoi filtering algorithm [27]. If there is such an occurrence, the colour patterns of the mesh may

become distorted and the colour will not be properly preserved. The preprocessing pipeline requires the colour property of the meshes to be preserved, therefore, algorithms that failed to preserve the colour property were immediately eliminated from consideration. The algorithms that passed the test, proceeded to the next test.

The results of the test were evaluated qualitatively since the presence of colour in the meshes could be observed by in. The point cloud was spection. Figure 3.1 shows the coloured point cloud that was used to perform the test.

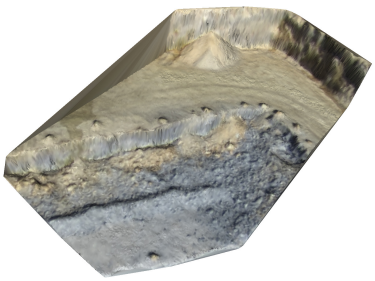
Figure 3.1. The point cloud that was used in the colour preservation test; generated using images from the Swiss quarry dataset [36]

Sub-figures 3.2(a) to 3.2(f) show the surfaces reconstructed by the Delaunay triangulation, Marching cubes, Voronoi filtering, Ball Pivoting, Poisson surface reconstruction and the Fast surface reconstruction algorithms, respectively.

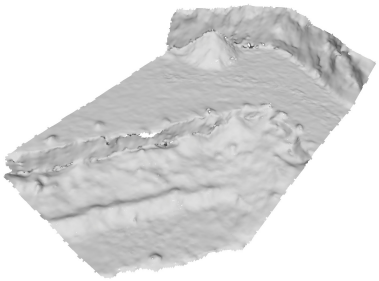
The results of the test are presented in Table 3.1. The selection verdict given to each algorithm is also shown. Algorithms that failed the test were immediately eliminated, making them ineligible for the preprocessing pipeline. From the verdicts presented in Table 3.1, the algorithms that remained eligible for selection were the Delaunay triangulation, Ball Pivoting and the Poisson surface reconstruction algorithms.

3.3.2 Test for visual accuracy

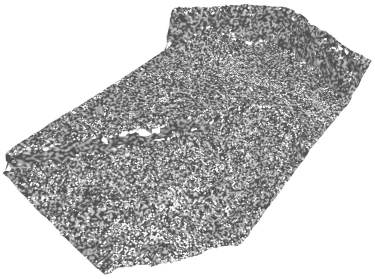
Having identified the algorithms that could preserve the RGB colour property of the point cloud in the resulting mesh, the next test involved identifying the algorithms that produced meshes that best visually represented the point cloud. The task involved testing two key aspects:



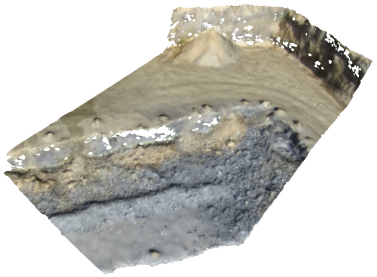
(a) Delaunay triangulation



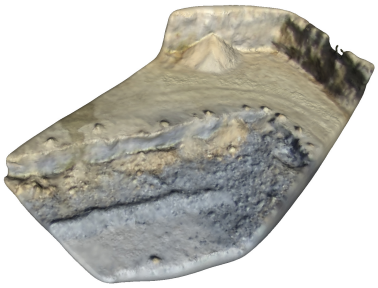
(b) Marching cubes surface reconstruction



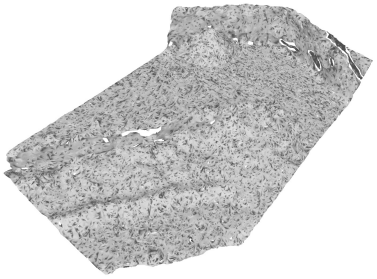
(c) Voronoi filtering surface reconstruction



(d) Ball Pivoting surface reconstruction



(e) Poisson surface reconstruction



(f) Fast surface reconstruction

Figure 3.2. Surface reconstruction results on the point cloud shown in Figure 3.1.

1. the absence of holes in the resulting mesh;
2. the visual similarity between the original point cloud and the resulting mesh.

An algorithm that passed both tests would be suitable for implementation as the surface reconstruction algorithm in the system's pipeline.

To carry out the test, a point cloud was extracted from a known CAD model. Each of the remaining eligible surface reconstruction algorithms was used to reconstruct the surface from the extracted point cloud. Since the original CAD model was available, the presence or absence of holes in the resulting

Table 3.1. The test results and the verdict

Algorithm	RGB Preservation	Verdict
Delaunay triangulation	Yes	Proceed
Marching cubes	No	Eliminated
Voronoi filtering	No	Eliminated
Ball Pivoting	Yes	Proceed
Poisson surface reconstruction	Yes	Proceed
Fast surface reconstruction	No	Eliminated

reconstructed surfaces could be checked qualitatively against the known CAD model. The visual similarity of the reconstructed surface could also be compared qualitatively with the original CAD model. The CAD models that were used for the test were retrieved from the Stanford University repository of CAD models [73]. The dataset has permissions that allow for educational use [73].

3.3.2.1 Armadillo man

Sub-figure 3.3(a) shows the original CAD model of the Armadillo man and sub-figures 3.3(b) to 3.3(d) show the reconstruction results of the Delaunay triangulation, Ball Pivoting and the Poisson surface reconstruction algorithms, respectively.

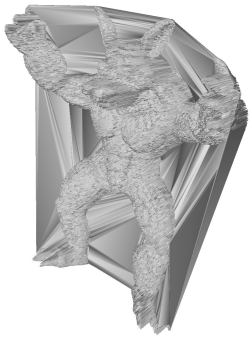
3.3.2.2 Bunny

Sub-figure 3.4(a) shows the original CAD model of the Bunny and sub-figures 3.4(b) to 3.4(d) show the reconstruction results of the Delaunay triangulation, Ball Pivoting and the Poisson surface reconstruction algorithms, respectively.

3.3.2.3 The results

Table 3.2 gives the test results and the verdict after considering all the surface reconstruction results obtained from the point clouds of the four CAD models. Table 3.2 shows that the Poisson surface reconstruction algorithm was the only algorithm that passed the test.

(a) Original CAD model



(b) Delaunay triangulation

(c) Ball Pivoting algorithm

(d) Poisson surface reconstruction

Figure 3.3. Surface reconstruction results of a point cloud of the Armadillo man, a CAD model in the Stanford University dataset [73].

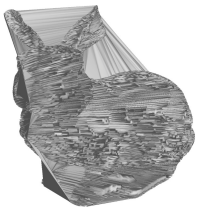
Table 3.2. The final test results and the final verdict

Algorithm	Holes present	Similar to original	Verdict
Delaunay triangulation	No	No	Eliminated
Ball Pivoting	Yes	Yes	Eliminated
Poisson surface reconstruction	No	Yes	Proceed

3.3.3 The final selection of a suitable surface reconstruction algorithm

As previously seen, the Poisson surface reconstruction algorithm was the most suitable algorithm that met the criteria required for the surface reconstruction algorithm. The algorithm was therefore selected

(a) Original CAD model



(b) Delaunay triangulation

(c) Ball Pivoting algorithm

(d) Poisson surface reconstruction

Figure 3.4. Surface reconstruction results of a point cloud of the Bunny, a CAD model in the Stanford University dataset [73].

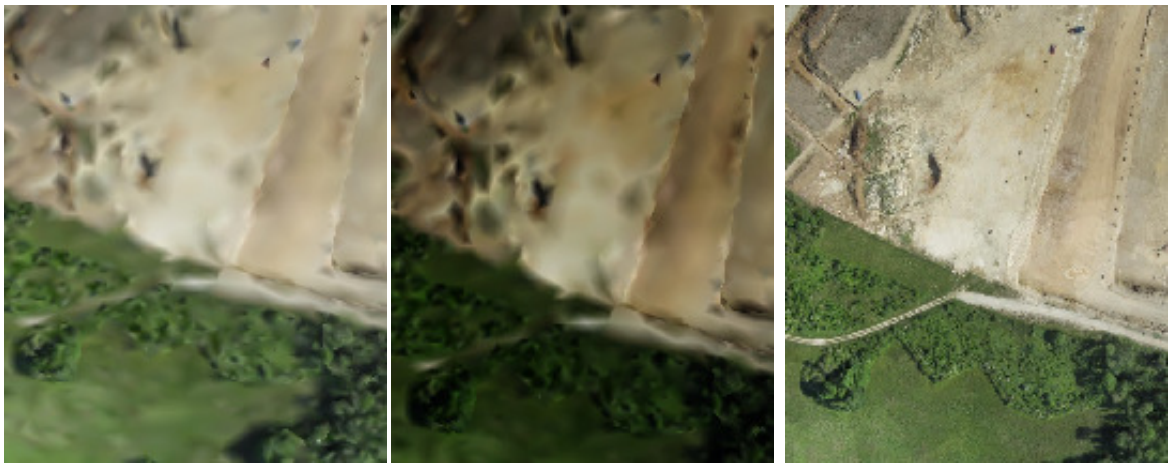
as the surface reconstruction algorithm that would be used in the preprocessing pipeline.

3.4 TEXTURE MAPPING

3.4.1 Comparison of image-based texture mapping and vertex-colour-based texture mapping

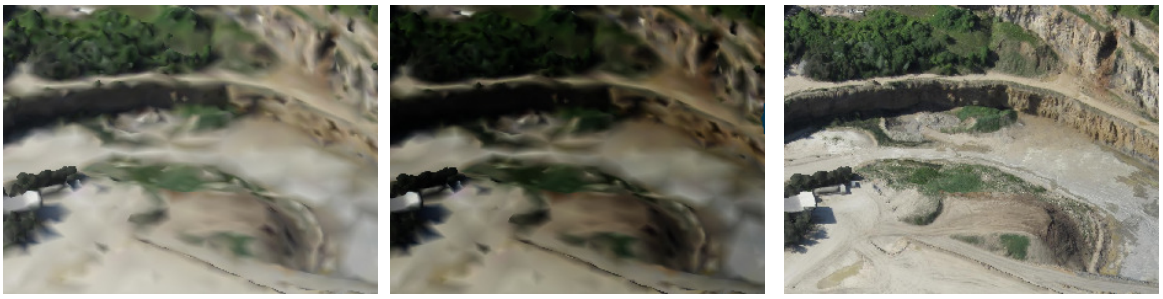
Creation of texture maps for the reconstructed meshes can be facilitated by the use of the original images of the mesh [39], [37], [40] and [38], or by the use of vertex-colours extracted from the point cloud [74]. The following section investigates which of the two approaches is appropriate for the VR application. Figures 3.5, 3.6 and 3.7 show the results of texture mapping a mesh, using both methods, on three scenes of the Swiss quarry dataset [36]. Meshlab [70] was used to texture map the mesh using the mesh's vertex-colours and Pix4Dmapper [35] was used to texture map the mesh using the original images.

Observations of the results showed that:



(a) Untextured **(b)** Vertex-colour textured **(c)** Textured using images

Figure 3.5. A comparison of textures generated from the mesh's vertex-colours and textures generated from the original images, using meshes generated from the Swiss quarry dataset [36].

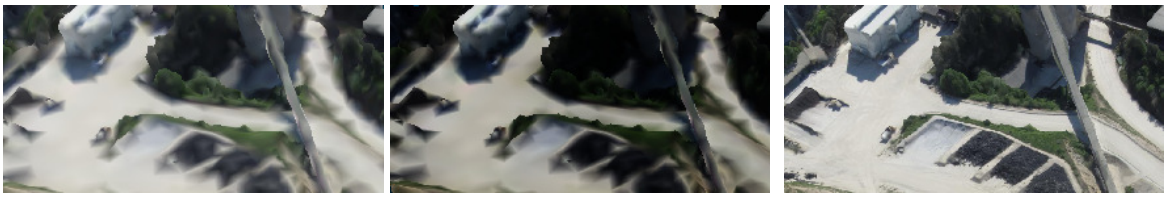


(a) Untextured **(b)** Vertex-colour textured **(c)** Textured using images

Figure 3.6. The second comparison of textures generated from the mesh's vertex-colours and textures generated from the original images, using meshes generated from the Swiss quarry dataset [36].

1. Texture mapping using vertex-colours enhanced the colours of the untextured mesh but did not add any extra details to the appearance of the mesh.
2. Texture mapping using images created a detailed texture which resembled the real-world landscape.

A point cloud is a sample of an actual surface's representation, meaning that, not all points making up a surface are contained in the point cloud. Furthermore, when a surface is reconstructed, a small sample of the points are lost along with their vertex-colour information. Consequently, using the vertex-colours



(a) Untextured **(b)** Vertex-colour textured **(c)** Textured using images

Figure 3.7. The third comparison of textures generated from the mesh's vertex-colours and textures generated from the original images, using meshes generated from the Swiss quarry dataset [36].

for texture mapping will lead to textures with relatively lower LOD due to the lost information.

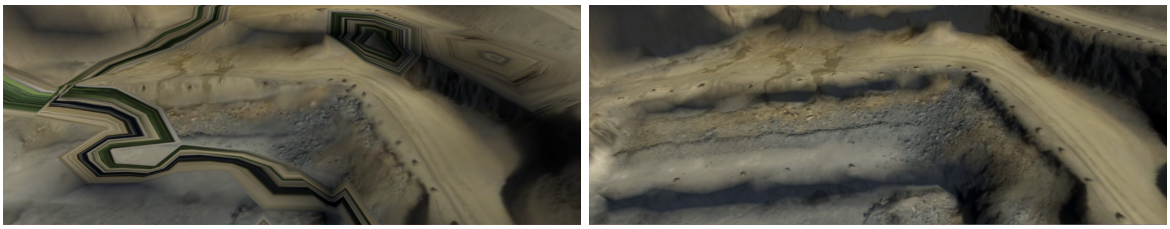
On the other hand, texture mapping, using the original images, is independent of the lost information since the texture map is derived directly from the original images from which the point cloud was sampled. The original images contain all the visual details of the landscapes; hence, their texture maps add extra details to the mesh, to make them resemble the real-world landscapes properly. The texture maps used in the VR application were therefore created using multiple images of the real-world environment.

3.4.2 Texture map correction for Unity and Unreal engines

Since the meshes are to be rendered in the Unity and Unreal engines, the data coming from Pix4Dmapper must be compatible with the texture mapping methods of the engines. The UV mapping data format that is produced by Pix4Dmapper takes the form of a face to texture mapping, however, the texture mapping methods in the engines require a vertex to texture mapping format instead.

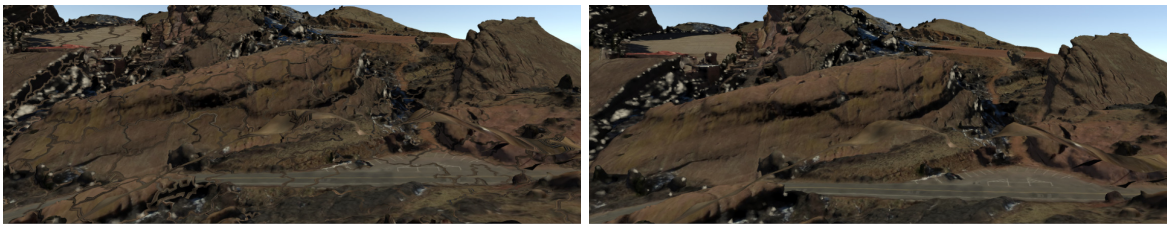
To counter such an anomaly, the texture map that is received from Pix4Dmapper is processed by the Meshlab routine, *Convert PerWedge UV into PerVertex UV*, before it is passed on to the engine. The routine converts the texture map from a per-face format to a per-vertex format. Figures 3.8 and 3.9 show two comparisons of how the two types of mapping formats are rendered in the engines.

Figures 3.8 and 3.9 show that the per-vertex format produces a distorted texture whilst the per-face format achieves the required undistorted texture; justifying the introduction of the correction step into the pipeline.



(a) Per-face mapping **(b)** Per-vertex mapping

Figure 3.8. Comparison of a mesh textured using per-face mapping with a mesh textured using per-vertex mapping. The meshes were generated using images from the Swiss quarry dataset [36].



(a) Per-face mapping **(b)** Per-vertex mapping

Figure 3.9. The second comparison of a mesh textured using per-face mapping with a mesh textured using per-vertex mapping. The meshes were generated using images from the Red Rocks dataset [75].

3.5 SEGMENTATION

3.5.1 The motivation to design a new segmentation algorithm

Segmentation of complex landscapes requires an algorithm that is insensitive to the complex geometries of the landscape. Ideally, the segmentation algorithm needs to avoid using metrics such as the normals or the curvature. The algorithm also needs to avoid user-defined parameters such as the expected number of segments, hence, the segmentation algorithm has to automatically determine the number of segments in a mesh.

From the state-of-the-art algorithms discussed in Chapter 2, the k-means [61], Fitting Primitives [62], Normalised Cuts, Random Cuts [66] and Random Walks [67] algorithms all require a predefined number of segments. The Core Extraction [68] and Shape Diameter algorithms [69] both require the geometric attributes of normals and curvatures respectively.

In addition, as will be seen in Chapter 5, the state-of-the-art algorithms do not perform well in segmenting meshes of real-world landscapes. Hence, to avoid using geometric properties of the mesh and to avoid the need to predefine the expected number of segments, a novel segmentation algorithm is presented in Section 3.5.2.

3.5.2 Overview of the segmentation algorithm

Consider a mesh as being made up of distinct meaningful segments of faces that share a common physical attribute. Suppose that the value of the attribute is similar for all the faces in one segment. If the value of the attribute is measurably different between neighbouring contiguous segments, the proposed segmentation algorithm can be used to distinguish the different segments.

For real-world landscapes, the mesh's colour attribute can be used to distinguish the segments, hence, for the preprocessing pipeline of the VR application, the colour attribute is utilised. However, any other property that can be used to represent the homogeneity of the faces in a single segment can be used by the algorithm. The SDF metric, presented in [69], was used to compare the algorithm with the state-of-the-art algorithms in the experiments that are analysed in Chapter 5. The segmentation algorithm's pipeline has the following stages:

1. Coloured triangular-face mesh input;
2. Data preprocessing;
3. Segments creation;
4. Segments refining.

3.5.3 The coloured triangular mesh input

The input to the algorithm is a coloured triangular mesh. In the context of the proposed algorithm, the face-colours and the contiguity of the triangular faces' vertices are the important attributes of the data structure.

3.5.3.1 The motivation for using the triangular faces

The connections of the vertices to form triangular faces add information to the data structure. The connections between the vertices determine the neighbours of a face due to their contiguity. When contiguous neighbours are known, it is possible to determine if they belong to the same segment by using their similarities.

3.5.3.2 The motivation for using the colour property

As seen in Chapter 2, the majority of the segmentation algorithms use similarity metrics that are made up of geometrical attributes. Real-world landscapes are usually geometrically irregular, and using a geometric attribute to formulate a similarity metric, would lead to poor results [52].

Even though objects lack geometrical regularity, in nature, distinct regions will generally be of uniform colour. Colour is, therefore, a good property to incorporate in formulating a similarity metric to distinguish different segments of meshes of complex landscapes.

3.5.4 Data preprocessing

The similarity metric that is used for the segmentation algorithm is comprised of spatial data and colour data. Sections 3.5.4.1 and 3.5.4.2 show how the data from the triangular mesh input is preprocessed to evaluate the similarity metric.

3.5.4.1 Preprocessing of colour data

The vertex-colours of the mesh are used to calculate the average colour of each face to form face-colours. Define $\mathbf{l}$, a vector containing attributes referred to as the colour similarity indices, given by (3.1),

$$\mathbf{l}_i = \Delta E(\mathbf{c_{white}}, \mathbf{c_i}), \tag{3.1}$$

where $\mathbf{l}_i$ is the colour similarity index of the i^{th} face, $\mathbf{c}$ is a vector of the CIELAB colour values of each face, $\mathbf{c_{white}}$ is the CIELAB colour value of the colour white and ΔE is a function to calculate the CIEDE2000 [76] LAB colour space difference between two faces' colours. The difference in the

colour similarity indices between two faces is small when the colours are similar and large when they are different.

The CIELAB colour space [54] was chosen to represent the colour values in the algorithm. The colour space, unlike other colour spaces, represents colour values conforming to human perceptual vision; similar colours as seen by the human eye are numerically closer together.

The colour space has had several comparison functions defined since 1976, each new one an improvement to the previous ones. The CIEDE2000 [76] is a function that can be used to compare the similarity between two colours. The CIEDE2000's function ranges from 0 to approximately 100, the similarity decreasing with increasing values.

Another of the CIELAB colour space's strengths is its ability to account for lightness, which is a critical factor of the research since lighting affects the images of the landscape. Therefore, the CIELAB colour space becomes a favourable option to be used in the research [52].

3.5.4.2 Preprocessing of spatial data

Each vertex in the mesh has 3D spatial coordinates that represent it in space. These coordinates can be used to calculate the spatial coordinates of each face's centre. The spatial data is preprocessed in such a way that the faces can be represented by a standard separation between them. The representation provides a standard measure of the closeness of the faces amongst each other regardless of the units of distance used by the 3D sensor in retrieving the point cloud. Define $\mathbf{d}$, a vector of attributes referred to as the standardised separation indices, given by (3.2),

$$\mathbf{d}_i = \frac{E(\mathbf{p_1}, \mathbf{p_i})}{max(\mathbf{d})}, \tag{3.2}$$

where $\mathbf{d}_i$ is the standardised separation index of the i^{th} face, $\mathbf{p_i}$ is a vector of the raw spatial coordinates of a face p and $E()$ is a function to calculate the Euclidean distance between two faces. $\mathbf{p_1}$ represents the spatial coordinates of the first face in the mesh's list of faces. The assignment of the first point is arbitrary and it should be noted that $\mathbf{p_1}$ can be selected as any of the faces in the list. Intuitively, the smaller the difference between two faces' $\mathbf{d}$ values, the closer those two faces are to each other. The range of $\mathbf{d}$ is from 0 and 1.

3.5.5 Creating segments: The Triangle Pool Algorithm (TPA)

The following sections illustrate the operation of the novel segmentation algorithm, referred to as the Triangle Pool Algorithm (TPA). Assume that the real-world model to be provided as the input to the algorithm is a coloured triangular mesh with known positional coordinates of its triangular faces' centres and known triangular faces' colours. It is assumed that the mesh's faces are contiguous. Initially, all the triangles can be imagined to be contained in a *pool of triangles* denoted as the *level 1 pool*.

3.5.5.1 Creating level 2 pools

The first step involves calculating the s values of each triangle in the pool given by (3.3),

$$s_i = \mu \mathbf{l}_i + \tau \mathbf{d}_i, \tag{3.3}$$

where s_i, $\mathbf{l}_i$ and $\mathbf{d}_i$ are the similarity index, colour similarity index and standardised separation index of the i^{th} face, respectively. μ and τ are constants with a range of 0 to 1, that control the effect of $\mathbf{l}_i$ and $\mathbf{d}_i$ on s_i. The segments of real-world environments are assumed to be more dependent on the colour than the spatial attribute. Hence, to ensure that s_i depended more on the colour than the spatial attribute, the parameters were set to, $\mu = 1$ and $\tau = 0.1$.

The second step involves grouping the triangles by clustering the triangular faces using the k-means clustering algorithm [77]. The distances of the clustering algorithm are calculated using the s values of the triangular faces.

To determine the number of clusters, k, to use in the k-means clustering algorithm, the s_i values are used. Define $\omega_{a,b}$ as the measure of similarity of two faces a and b, given by (3.4),

$$\omega_{a,b} = |s_a - s_b|, \tag{3.4}$$

where $\omega_{a,b}$ is the absolute difference between the similarity index of face f_a and the similarity index of another face f_b. The following steps are used to calculate k,

1. Initialise $k = 1$,

2. The face, f_0, with the smallest s value is selected. The quantity $\omega_{0,1}$ is calculated, for f_0 and f_1, the face with the second smallest s value. If $\omega_{0,1}$ is less than a threshold Th, the process is repeated for $\omega_{0,2}$ and so on until the condition $\omega_{0,n} > Th$ is encountered,
3. When the condition, $\omega_{0,n} > Th$, is encountered, k is incremented by 1 and all the faces f_0 to f_{n-1} are removed from consideration and step 2 is repeated until there are no more triangular faces to consider,
4. The final value of k is used as the number of clusters in the k-means clustering algorithm.

The result of the stage is that the faces are grouped into k different clusters denoted *level 2 pools*. Intuitively, each *level 2 pool* contains faces that are spatially close together and with similar colours.

The threshold value, Th, controls the size of the *level 2 pools*. A large value of Th means that more faces are considered to be in the same grouping than when the value of Th is lower; meaning that, at higher values of Th, there are fewer and larger *pools*.

3.5.5.2 Creating level 3 pools

The result of the previous stage is that spatially close and similar faces of the mesh are grouped and are separated from the rest of the other mesh's faces. The aim of creating *level 3 pools* is to group faces that fall within the same physical segment boundaries.

The wire-frame structure of the triangular mesh makes it possible to effectively group faces that are within the same boundaries. In the wire-frame, the triangular faces are contiguous as long as the mesh is watertight. Consequently, a face in a particular segment has to be contiguous to at least one other face in that segment. Using this assumption; for each *level 2 pool*, an initial seed triangle is selected. A greedy algorithm [78] is used to search for all the faces that are contiguous to the seed. For all the faces that are found, the greedy algorithm searches for their contiguous faces. The search is iterated and terminated when no contiguous faces can be found. All the faces that are found during the search, form a *level 3 pool*. A new seed is selected from the remaining unassigned faces and the process is repeated until all the faces belong to a *level 3 pool*.

Greedy algorithms attempt to find a global solution to a problem by making irreversible local decisions [78]. For a greedy algorithm to give an optimal global solution, the local decisions themselves need to provide an optimal solution. In the design, the algorithm searches through a lookup table consisting of contiguous faces; every local decision that retrieves a contiguous face pair is an optimal solution. Hence, a greedy search algorithm provides an optimal solution to the problem.

As the size of the mesh increases, it is expected that the greedy algorithm's search time will also increase since there will be more triangular faces to search through. To reduce the run-time, multiprocessing is introduced to concurrently execute the greedy algorithm on multiple instances of the *level 2 pools*. The number of processes executing at the same time depends on the available CPU cores.

3.5.5.3 Refining the level 3 pools

After the *level 3 pools* are created, the final step involves merging all the smaller *level 3 pools* with the larger *level 3 pools*. Define the normalised size, δ, of a segment, given by (3.5),

$$\delta = \frac{y}{x}, \tag{3.5}$$

where y is the number of faces in the segment and x is the total number of faces in the mesh.

Suppose that a *level 3 pool* is defined as being small when $\delta < sTh$, where, sTh is a threshold. If γ_L represents the average of the s values of a large *level 3 pool*, the small and large *level 3 pools* can be merged by minimising the function g, given by (3.6),

$$g = \sum_{n=1}^{N} \alpha + |\gamma_L - \gamma_n|, \tag{3.6}$$

where n is the n_{th} small *level 3 pool*, γ_n is the average of the s values of its faces and α is a value of 1 when none of the faces in the small *level 3 pool* is contiguous to any face in the large *level 3 pool*; or is 0 otherwise.

When g is minimised, all the large and small *level 3 pools* pairs that evaluate $\alpha = 0$ are merged. Intuitively, g is minimised when all the small *level 3 pools* are merged with the large *level 3 pools* to which they are most similar to and to which at least one contiguous pair of faces exists between them. The final *level 3 pools* form the segments of the input mesh.

3.5.6 Summary of the segmentation algorithm

The segmentation algorithm presented in this chapter uses a mesh's spatial information, colour information and its triangular faces information. The algorithm attempts to segment a mesh with disregard to its geometric properties; an advantage when segmenting meshes of real-world landscapes that have irregular geometrical properties. Another advantage of the algorithm is that the number of segments is automatically determined by the algorithm.

3.6 SUMMARY

This chapter provided details on the approach that was taken to achieve the research objectives. The operation of the VR application was described and explained. The requirements to achieve the VR application's operation were stated, specifically the need for a surface reconstruction algorithm, texture mapping algorithm and a segmentation algorithm. The three aspects were described in detail, showing how pre-existing surface reconstruction and texture mapping algorithms were to be implemented in the preprocessing pipeline. The chapter concluded by formulating a novel segmentation algorithm that was also to be implemented in the preprocessing pipeline.

CHAPTER 4 EVALUATION

4.1 CHAPTER OVERVIEW

This chapter provides the evaluation procedures that were conducted for the experiments that are presented in Chapter 5. The datasets and metrics that were used for the evaluation are all presented. Two types of datasets are identified:

1. Datasets that represent large real-world environments such as mining quarries.
2. Datasets that represent synthetic objects.

Both types of datasets were used to conduct the experiments and a combination of quantitative and qualitative analysis was used to evaluate them. The metrics that were used for the evaluation are also presented in this chapter.

4.2 DATASETS USED

Various datasets were used in the experiments associated with the research. Point clouds and meshes retrieved from real-world landscapes were used in the experiments for surface reconstruction, texturing and segmentation. A benchmark dataset of Computer-Aided Design (CAD) models was also used in experiments involving segmentation. The following sections give the details of the datasets.

4.2.1 The Swiss quarry dataset

A dataset of images of a quarry [36] was used in the surface reconstruction, texturing and segmentation experiments. The dataset contains 127 original images of the quarry, a 3D point cloud and a 3D mesh. The point cloud is made up of approximately 12 million points.

4.2.2 The Gravel quarry dataset

Another dataset of a quarry [79] was used to evaluate the experiments. The dataset comprises of 136 images, a point cloud and a mesh. The point cloud is made up of approximately 16 million points.

4.2.3 The Red Rocks dataset

The dataset is taken from the Red Rocks Open-air Amphitheatre in Colorado [75]. The dataset contains 45 images and a typical point cloud generated from the images contains approximately 4 million points.

4.2.4 The Reservoir dataset

The dataset was retrieved from a reservoir [75]. The dataset contains 48 images and a typical point cloud generated from the images contains 5 million points.

4.2.5 The Princeton dataset

The Princeton dataset was used as a benchmark for quantitatively comparing segmentation algorithms [65]. The dataset was created to quantitatively evaluate 3D segmentation algorithms since qualitative evaluations are highly subjective. Further details on how to evaluate algorithms, using the benchmark, are given in Section 4.3.2.

The Princeton dataset consists of 19 classes of different object types, each containing 20 CAD models, making the total size of the dataset equal to 380 objects. Ground-truth segmentations were created by 80 different people who submitted segmentations of the models that they had done on their own. In total, each model has an average of 11 different ground-truth samples.

4.2.6 Dataset Permissions

The Swiss quarry and Gravel quarry datasets can be used for non-commercial purposes as stated in the term and conditions [80]. The Red Rocks and Reservoir datasets can also be used for non-commercial purposes as stated in the terms of use [81]. The Princeton dataset can be used for academic research [65].

4.3 EVALUATION APPROACH

A series of experiments on surface reconstruction, texturing and segmentation were carried out. The experiments intended to evaluate the performance of the selected algorithms on large real-world data. In addition, experiments were undertaken to justify the use of the novel segmentation algorithm, TPA, over the existing state-of-the-art algorithms. The following sections outline the metrics that were used in the evaluation.

4.3.1 Surface reconstruction and texturing

The experiments that were carried out concerning surface reconstruction and texturing were evaluated qualitatively. For surface reconstruction, the process involved qualitatively visualising the difference between the input point cloud and the output mesh. For texturing, the process involved qualitatively visualising an untextured environment in VR against a textured environment.

4.3.2 Quantitative evaluation of 3D mesh segmentation

The Princeton dataset was used to quantitatively evaluate the proposed algorithm against the human-generated ground-truth segments and also against the state-of-the-art algorithms. The benchmark [65] defines four metrics that were used to evaluate the algorithms.

4.3.2.1 Cut Discrepancy

The Cut Discrepancy (CD) measures the distances between boundary cuts of the algorithm's segments and the boundary cuts of the ground-truth human-generated segments. The task is achieved by computing the distance between points in the generated segments' cuts with the points in the closest cuts of the ground-truth segments. Suppose that for segmentations S_1 and S_2 with cuts C_1 and C_2 respectively, the geodesic distance between a point p_1 in C_1 and the cut C_2 is given by (4.1);

$$d_G(p_1, C_2) = min\Big\{ d_G(p_1, p_2) \Big\}, \tag{4.1}$$

where p_2 is a point in C_2.

The Directional Cut Discrepancy (DCD) of S_1 relative to S_2 is given in (4.2),

$$DCD(S_1, S_2) = mean\Big\{ d_G(p, C_2) \Big\}, \tag{4.2}$$

where $d_G(p, C_2)$ is the distribution of the function evaluated in (4.1) for all points p in C_1.

The Cut Discrepancy (CD) is finally defined in (4.3),

$$CD(S_1, S_2) = \frac{DCD(S_1, S_2) + DCD(S_2, S_1)}{AveR}, \tag{4.3}$$

where *AveR* is the distance from a point on the mesh's surface to the centre of the mesh.

The CD metric provides an intuitive way to evaluate how well the boundaries of two segmented objects align. The metric is, however, sensitive to the number of segments identified. For instance, it is undefined when either of the segmented objects has no cuts and it decreases towards zero as more segments are added to the ground-truth.

4.3.2.2 Hamming Distance

The Hamming Distance (HD) measures the differences between the regions of two segmentations. For two sets of segmentations $S_1 = \{S_1^1, S_1^2 ... S_1^m\}$ and $S_2 = \{S_2^1, S_2^2 ... S_2^n\}$, the Directional Hamming Distance of S_1 relative to S_2 , $D_H(S_1, S_2)$ is given in (4.4),

$$D_H(S_1, S_2) = \sum_i \Big| S_2^i \backslash S_1^{it} \Big|, \tag{4.4}$$

where $\backslash$ is an operator that measures the difference between two sets.

The missing rate R_m and the false rate R_f are given by (4.5) and (4.6), respectively;

$$R_m(S_1,S_2) = \frac{D_H(S_1,S_2)}{|S|}, \tag{4.5}$$

$$R_f(S_1,S_2) = \frac{D_H(S_2,S_1)}{|S|}, \tag{4.6}$$

where $|S|$ is the total surface area of the mesh.

The Hamming Distance, HD, is finally given by (4.7),

$$HD(S_1,S_2) = \frac{1}{2}\Big(R_m(S_1,S_2) + R_f(S_1,S_2)\Big), \tag{4.7}$$

4.3.2.3 Rand Index

The Rand Index (RI) measures the likelihood of two faces being in the same segment in two segmentations or being in different segments in those two segments. The metric measures the overlapping area of the segments without the need to find segment correspondences. For two segmentations S_1 and S_2, s_i^1 is the segment ID of face i in S_1 and correspondingly s_i^2 for S_2, RI is defined in (4.8),

$$RI(S_1,S_2) = \begin{bmatrix} 2 & N \end{bmatrix} \sum_{i,j,i<j} \left(S_2^i \Big[C_{ij}P_{ij} + (1-C_{ij})(1-P_{ij}) \Big] \right), \tag{4.8}$$

where N is the total number of faces in the mesh, $C_{ij} = 1$ if $s_i^1 = s_j^1$ and $P_{ij} = 1$ if $s_i^2 = s_j^2$.

When the faces i and j have the same segment ID in both S_1 and S_2, then $C_{ij}P_{ij} = 1$, and when the faces i and j have different segment IDs in both S_1 and S_2, then $(1-C_{ij})(1-P_{ij}) = 1$. RI, therefore, gives a measure of how faces agree or disagree on their given IDs in S_1 and S_2.

4.3.2.4 Consistency Error

The Consistency Error (CE) uses the theory that humans perceptually identify a hierarchical tree structure on objects to account for hierarchical similarities and dissimilarities in the algorithm's segmentation results and those of the ground-truth. For two segmentations S_1 and S_2, the local refinement error, E, is defined in (4.9),

$$E(S_1,S_2,f_i) = \frac{\Big|R(S_1,f_i) \backslash R(S_1,f_i)\Big|}{\Big|R(S_1,f_i)\Big|}, \tag{4.9}$$

where $\setminus$ is an operator that measures the difference between two sets, f_i is a mesh face and $R(S_1, f_i)$ is a segment that contains the face f_i.

Two metrics, the Global Consistency Error, GCE, and the Local Consistency Error, LCE, are defined in (4.10) and (4.11) respectively,

$$GCE(S_1, S_2) = \frac{1}{n} min\left\{ \sum_i E(S_2, S_1, f_i), \sum_i E(S_1, S_2, f_i) \right\}, \tag{4.10}$$

$$LCE(S_1, S_2) = \frac{1}{n} \sum_i min\left\{ E(S_2, S_1, f_i), E(S_1, S_2, f_i) \right\}, \tag{4.11}$$

where n is the total number of faces in the mesh.

4.3.2.5 Summary

The benchmark provides scripts to evaluate the TPA segmentation algorithm against the ground-truth, along with the evaluation results of the state-of-the-art algorithms; making it possible to evaluate the performance of the TPA algorithm, whilst also comparing the results with those of the state-of-the-art algorithms. One disadvantage of the benchmark, relative to the TPA algorithm, is that the CAD models do not have a colour property. However, since the proposed algorithm can use any homogenous attribute of the regions of an object to calculate a similarity metric, the colour property was replaced by the SDF attribute [69] during the benchmark evaluation experiments.

4.3.3 Qualitative evaluation of colour-based segmentation

Apart from quantitative analysis, another technique that was used to evaluate segmentation algorithms involved manually inspecting the results against the input [41], [48] and [52]. The segments produced by an algorithm are manually inspected to check if they are accurate. Advantages of such an approach are:

- The approach is flexible enough to evaluate the results based on the definition of regions according to a particular application;
- The approach does not depend on the availability of ground-truth data.

However, the approach's main disadvantages are:

- The approach is subjective and different evaluators may have different opinions on the results;
- For large meshes, qualitatively comparing the segments against the input, may become infeasible.

4.4 SUMMARY

This chapter detailed how the evaluation of the various algorithms was conducted and the datasets that were used for the evaluation. The output being qualitative, surface reconstruction and texturing algorithms were evaluated using a qualitative approach. The segmentation algorithm was evaluated using both a qualitative and quantitative approach. The quantitative approach was adopted from the Princeton segmentation benchmark.

CHAPTER 5 EXPERIMENTS AND RESULTS

5.1 CHAPTER OVERVIEW

This chapter provides the experiments that were carried out to evaluate the algorithms that were implemented in the research. Section 5.2 presents the experiments that were conducted to evaluate the surface reconstruction algorithm. The experiments included those that were meant to evaluate point cloud to mesh reconstruction and those that were meant to evaluate the creation of meshes with different LOD. The datasets of the real-world environments were used for the experiments.

Section 5.3 provides the experiments that were carried out to evaluate the texturing algorithm. The experiments were meant to show how surfaces appeared with and without texturing in VR. The datasets of the real-world environments were used for the experiments.

Section 5.4 provides the experiments that were carried out to evaluate the TPA segmentation algorithm. The experiments involved comparing the novel algorithm with state-of-the-art algorithms using both an existing benchmark and the datasets of real-world environments.

5.2 POINT CLOUD SURFACE RECONSTRUCTION BY THE POISSON SURFACE RECONSTRUCTION ALGORITHM

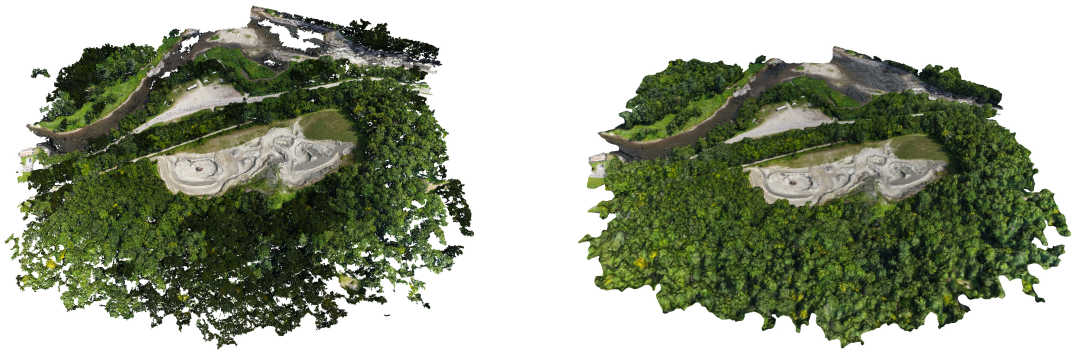
5.2.1 Generating meshes from point clouds

As discussed in Chapter 3, the Poisson surface reconstruction algorithm was selected as the surface reconstruction algorithm for the VR application's preprocessing pipeline. The algorithm was tested on various point clouds of real-world landscapes. Figures 5.1, 5.2, 5.3 and 5.4 show the results of

reconstructing point clouds generated from the Swiss quarry [36], Gravel quarry [79], Red Rocks [75] and Reservoir datasets [75]. The point clouds were generated by a photogrammetric process that used multiple images taken by a camera mounted on a drone.

(a) The input point cloud **(b)** The corresponding surface reconstructed mesh

Figure 5.1. Mesh generation by the Poisson surface reconstruction algorithm, on a point cloud generated using images from the Swiss quarry dataset [36].

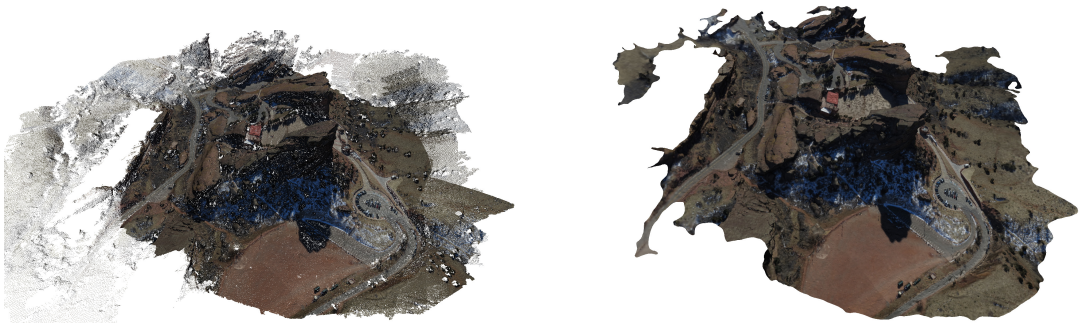


(a) The input point cloud **(b)** The corresponding surface reconstructed mesh

Figure 5.2. Mesh generation by the Poisson surface reconstruction algorithm, on a point cloud generated using images from the Gravel quarry dataset [79].

The results showed that the Poisson surface reconstruction algorithm:

1. Preserved the colour of the point cloud in the resulting mesh;
2. Filled up holes that were present in the point cloud;
3. Produced a mesh that visually represented the point cloud.



(a) The input point cloud **(b)** The corresponding surface reconstructed mesh

Figure 5.3. Mesh generation by the Poisson surface reconstruction algorithm, on a point cloud generated using images from the Red Rocks dataset [75].

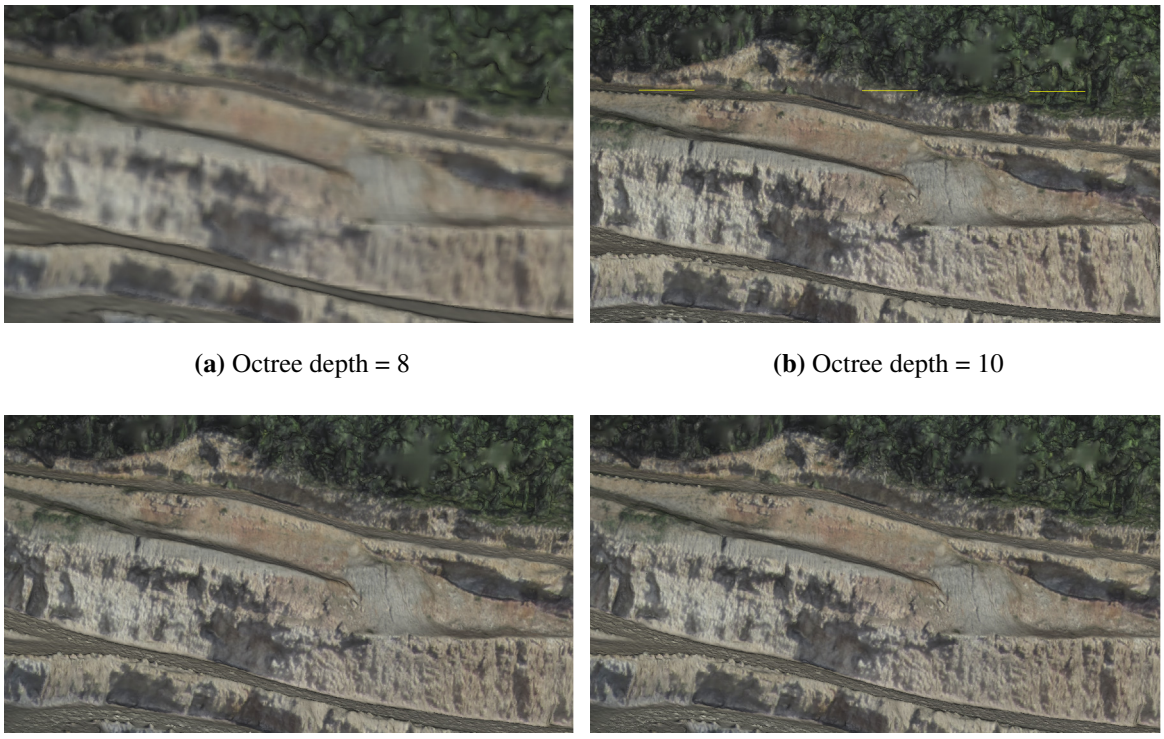
(a) The input point cloud **(b)** The corresponding surface reconstructed mesh

Figure 5.4. Mesh generation by the Poisson surface reconstruction algorithm, on a point cloud generated using images from the Reservoir dataset [75].

5.2.2 Creating low and high-resolution meshes, using the Poisson surface reconstruction algorithm

Due to the requirements of the VR application, the surface reconstruction algorithm needs to be able to produce low and high-resolution mesh versions of the same point cloud. Generating meshes with different LOD can be achieved by the Poisson surface reconstruction algorithm. The following section shows how the algorithm achieves this.

The Poisson surface reconstruction algorithm uses an octree data structure to store data during run-time [30]. Three point cloud scenes, *scene 1*, *scene 2* and *scene 3* were reconstructed into meshes using octree depths between 8 and 15. Figures 5.5, 5.6 and 5.7 show the meshes that were reconstructed at octree depths of 8, 10, 11 and 13.



(a) Octree depth = 8

(b) Octree depth = 10

(c) Octree depth = 11

(d) Octree depth = 13

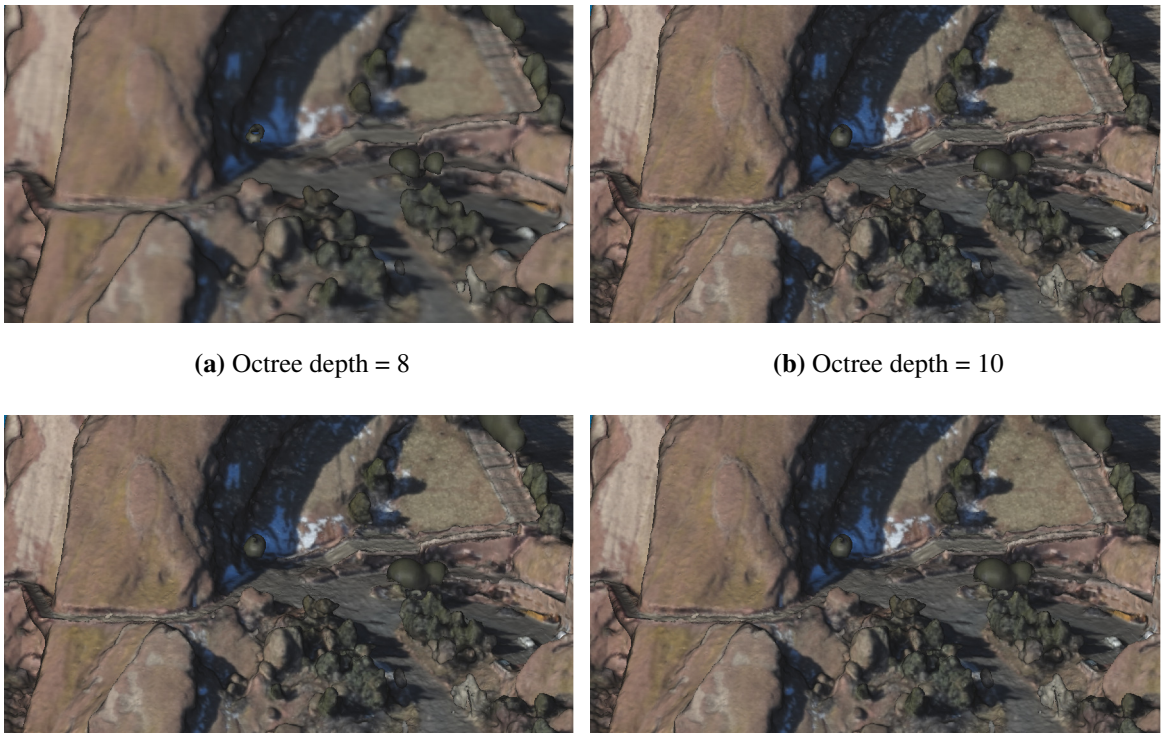
Figure 5.5. Poisson surface reconstruction results using octree depths of $8, 10, 11$ and 13 for *scene 1*. The input point cloud was generated using images from the Swiss quarry dataset [36].

Figures 5.5, 5.6 and 5.7 show that qualitatively:

- as the octree depth increased, the LOD increased;
- the difference between the LOD of the meshes became less-pronounced as the octree depth increased.

Figure 5.8 shows the variation of the number of faces of the mesh with octree depth. The larger the number of valid polygonal faces that a mesh contains, the higher the LOD of that mesh [82]. Therefore, the LOD between two meshes generated from the same point cloud can be compared by their number of faces. Figure 5.8 shows that:

- As the octree depth increased, the number of faces in the reconstructed meshes also increased;
- The difference between the number of polygonal faces of the meshes at consecutive octree depths decreased as the octree depth increased; at higher octree depths, the variation flattens out.



(a) Octree depth = 8

(b) Octree depth = 10

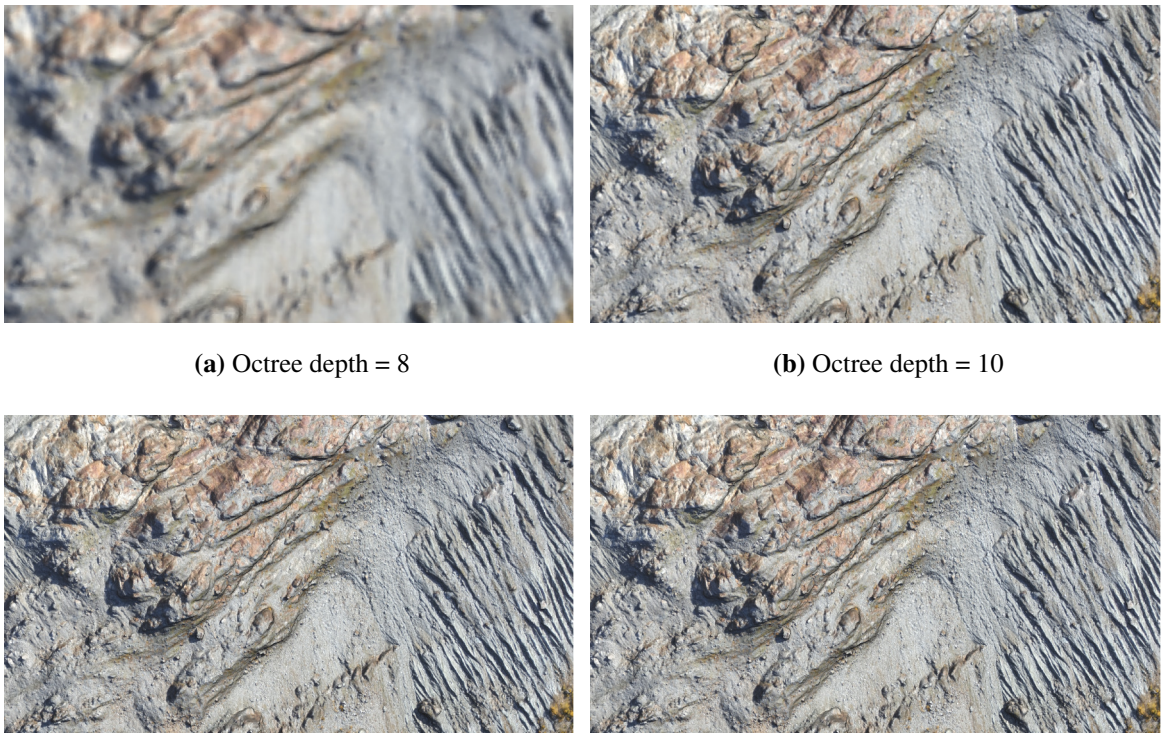
(c) Octree depth = 11

(d) Octree depth = 13

Figure 5.6. Poisson surface reconstruction results using octree depths of $8, 10, 11$ and 13 for *scene 2*. The input point cloud was generated using images from the Red Rocks dataset [75].

Two conclusions can be drawn from the observations, the first one being that, by reconstructing a point cloud using the Poisson surface reconstruction algorithm, different-sized meshes can be generated by varying the octree depth parameter of the algorithm. Therefore, the lower resolution mesh can be obtained by reconstructing the point cloud at a lower octree depth and the higher resolution mesh can be obtained by reconstructing the same point cloud at a higher octree depth. The main advantage of the approach is that only one point cloud is required to achieve both outputs.

The second conclusion that can be drawn from the observations is that, to obtain a mesh of reasonable higher resolution, it is not necessary to keep increasing the octree depths since the difference in the LOD becomes less significant as the octree depth gets higher. Figure 5.8 shows that, beyond an octree depth of 13, there was little difference in the LOD of the reconstructed meshes. Hence, in the preprocessing pipeline of the VR application, the higher resolution meshes were reconstructed using an octree depth of 13, whilst the lower resolution meshes were reconstructed using an octree depth of 9.



(a) Octree depth = 8

(b) Octree depth = 10

(c) Octree depth = 11

(d) Octree depth = 13

Figure 5.7. Poisson surface reconstruction results using octree depths of $8, 10, 11$ and 13 for *scene 3*. The input point cloud was generated using images from the Gravel quarry dataset [79].

5.2.3 Summary on surface reconstruction

The Poisson surface reconstruction algorithm preserves the colour in a reconstructed mesh; an advantage since the colour property is required by the TPA segmentation algorithm. The Poisson surface reconstruction algorithm also manages to produce watertight surfaces free from holes; an advantage since the surfaces in the VR environment are free from holes and gaps. The reconstructed meshes resemble the input point cloud to a large extent, hence, the output meshes can be used to model the environments in the virtual environments.

The Poisson surface reconstruction algorithm also has the advantage of being able to be to generate lower and higher resolution meshes from the same point cloud. The need to generate the two meshes from two separate point clouds is eliminated.

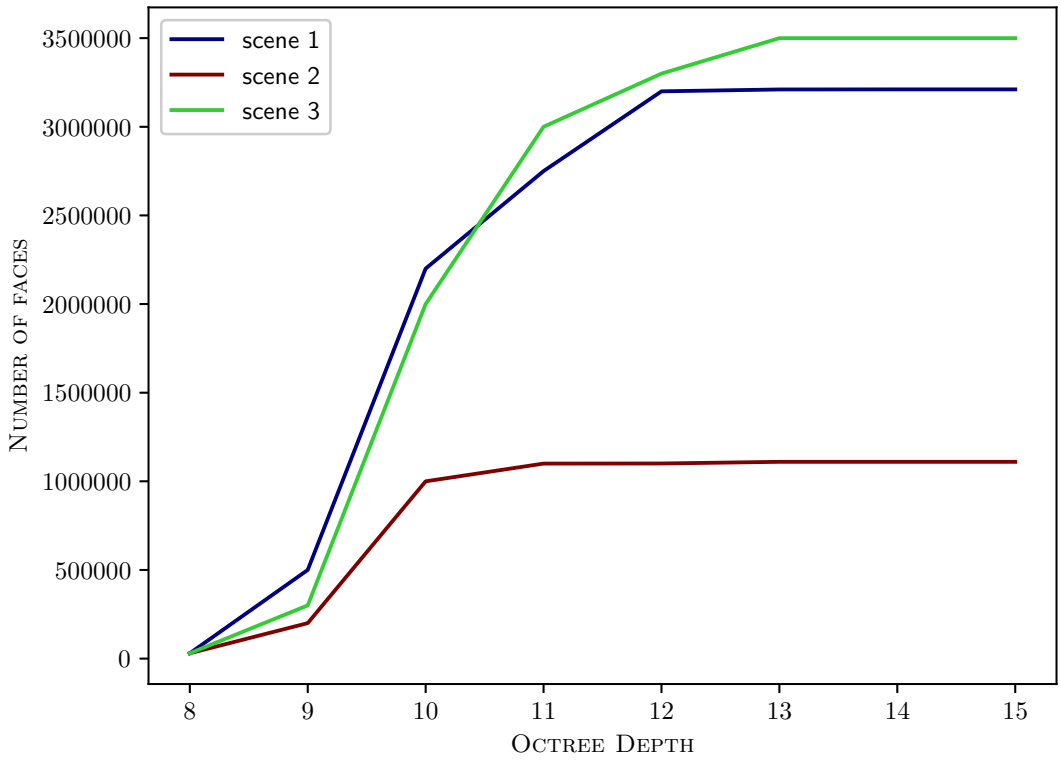
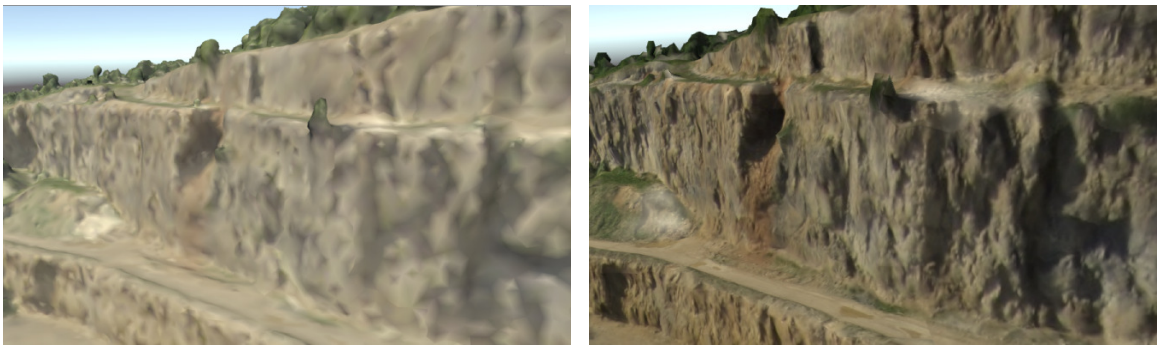


Figure 5.8. Variation of the number of faces with octree depth of the reconstructed meshes of *scene 1*, *scene 2* and *scene 3*

5.3 TEXTURING

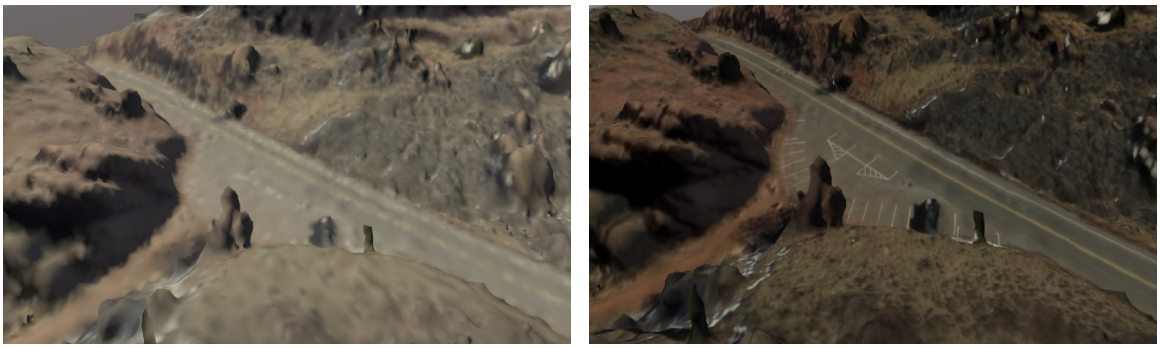
5.3.1 Texturing in VR

The following section provides an analysis of how the textured landscapes appeared in VR and aims to justify why texturing was necessary for the virtual environments in the VR application. The following experiment gives a qualitative analysis showing how the real-world environments appeared in the VR application when texture maps were both present and absent. The colour that was present in the untextured meshes was generated using the vertex-colours of the mesh. The texture maps of the textured meshes were generated using Pix4Dmapper [35] during the photogrammetric point cloud generation process. Figures 5.9, 5.10 and 5.11 show how the textured landscapes appeared in VR compared to how they appeared when coloured using their vertex-colours.



(a) A vertex-coloured mesh **(b)** A textured mesh

Figure 5.9. A comparison of a vertex-coloured mesh with a mesh that was textured using Pix4Dmapper [35]. The meshes were generated using images from the Swiss quarry dataset [36].



(a) A vertex-coloured mesh **(b)** A textured mesh

Figure 5.10. A second comparison of a vertex-coloured mesh with a mesh that was textured using Pix4Dmapper [35]. The meshes were generated using images from the Red Rocks dataset [75].

Addendum A provides additional comparisons of textured meshes and vertex-coloured meshes.

Observations showed that:

- The textured environments were more detailed than the vertex-coloured environments;
- The colours of the textured surfaces were sharp and not as blurry as observed in the vertex-coloured environments;
- The textured environments had a more realistic appearance than the vertex-coloured environments.

(a) A vertex-coloured mesh **(b)** A textured mesh

Figure 5.11. A third comparison of a vertex-coloured mesh with a mesh that was textured using Pix4Dmapper [35]. The meshes were generated using images from the Reservoir dataset [75].

5.3.2 Summary on texturing

The analysis that was provided in Section 5.3 justifies the need to use textured surfaces, instead of vertex-coloured surfaces, to model the surfaces of the meshes. Textured surfaces give the impression of a more realistic surface as per requirements of the VR application.

As confirmed in Chapter 3, using the original images of the surfaces to generate the texture maps, is the preferred method since it is independent of the point cloud's information that may be lost during the point cloud generation or the surface reconstruction phases.

5.4 SEGMENTATION

5.4.1 Overview

The following section provides an experimental analysis of the performance of the TPA segmentation algorithm on different kinds of meshes. In the first experiment, the Princeton dataset [69] was used to test the performance of the algorithm using an existing benchmark. The results were compared with human-generated ground-truths as well as the results from the state-of-the-art algorithms, that were detailed in Chapter 2.

The experiments that were performed in the following section intend to:

1. Show how the TPA algorithm compares with state-of-the-art algorithms, using an existing benchmark for evaluation;
2. Show how well the algorithms can segment meshes of landscapes into meaningfully-grouped regions;
3. Justify the use of the TPA algorithm in the pipeline;
4. Demonstrate how the high-resolution mesh is broken down into multiple segments of its regions using the TPA algorithm.

5.4.2 Evaluation using the Princeton dataset

The performance of the TPA algorithm was compared to seven state-of-the-art mesh segmentation algorithms. The experiment aims to show that the TPA algorithm produces results comparable to those of the state-of-the-art algorithms. The four metrics, RI, CD, HD and CE, were used for evaluation.

As previously mentioned in Chapter 4, since the CAD models in the Princeton dataset do not have a colour attribute, the colour attribute of each triangular face was replaced by the SDF attribute. If $\mathbf{z}_i$ is the SDF value for the i^{th} face of the CAD model, substituting $\mathbf{l}_i$ with $\mathbf{z}_i$ in (3.3), s_i is given by (5.1),

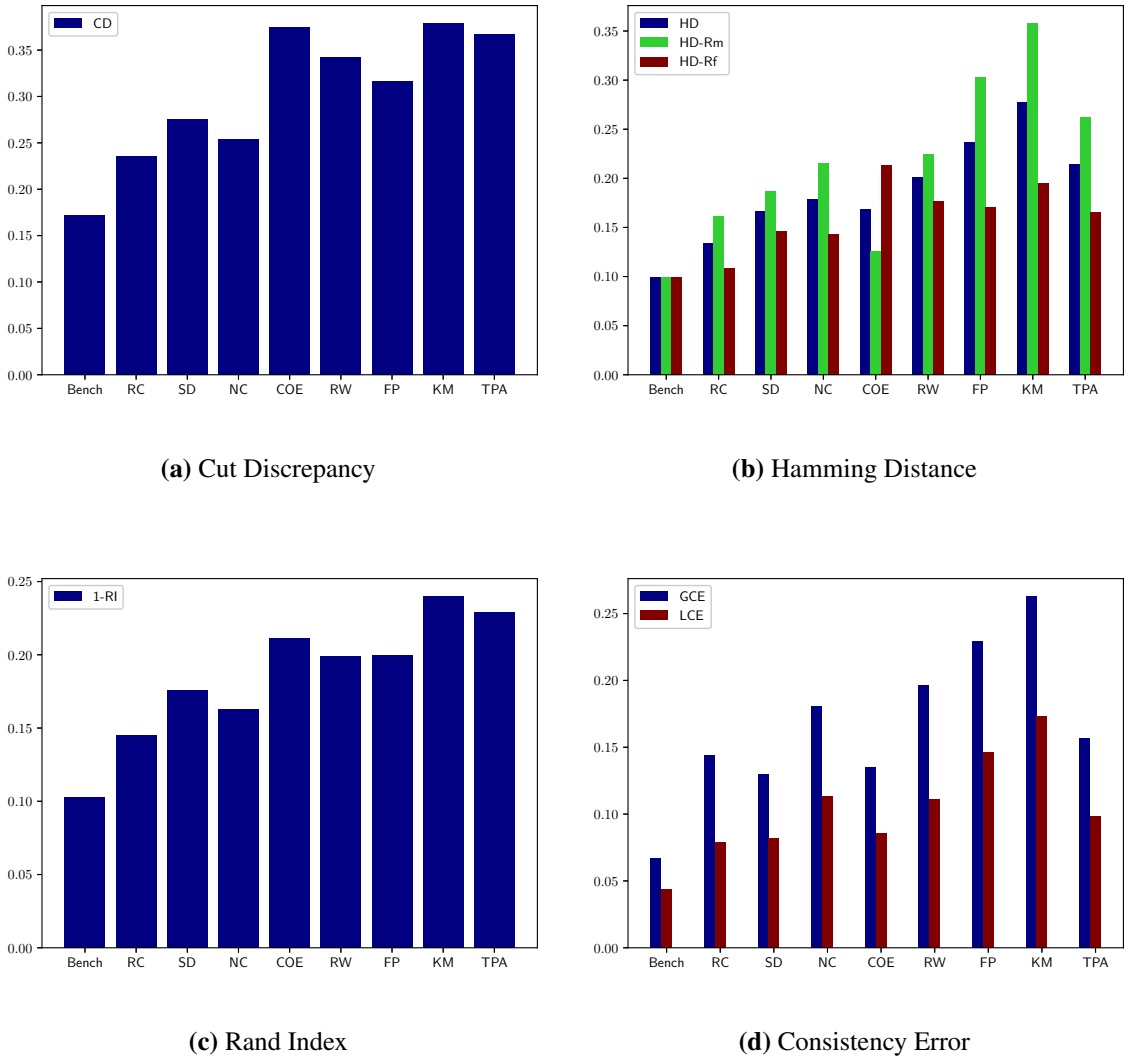
$$s_i = \mu \mathbf{z}_i + \tau \mathbf{d}_i, \tag{5.1}$$

where $\mu = 1$ and $\tau = 0.1$.

Experimental analysis showed that a Th value of 0.7 and an sTh value of 0.015 produce the best results. Values of Th higher than 0.7 will under-segment the mesh and lower values will over-segment the mesh. Values of sTh higher than 0.015 will lead to the incorrect merging of large segments and lower values will lead to small segments not being merged with large segments.

The names of the algorithms are keyed in as follows; Benchmark (Bench), Random Cuts (RC), Shape Diameter (SD), Normalised Cuts (NC), Core Extraction (COE), Random Walks (RW), Fitting Primitives (FP), K-Means (KM) and Triangle Pool Algorithm (TPA). Figure 5.12 shows the summarised

experimental results of segmenting the CAD models in the Princeton dataset, using the CD, HD, RI and CE metrics. Tables B.1, B.2, B.3, B.4, B.5, B.6 and B.7 in Addendum B, give the detailed evaluation results of the algorithms.



(a) Cut Discrepancy

(b) Hamming Distance

(c) Rand Index

(d) Consistency Error

Figure 5.12. Evaluation of 3D segmentation algorithms, using the CD, HD, RI and CE metrics. The algorithms were evaluated using the Princeton dataset [65].

5.4.2.1 CD evaluation

Sub-figure 5.12(a) shows how the TPA algorithm compared with the state-of-the-art algorithms, using the CD metric. Table B.1 gives the average CD values of each algorithm per category of the Princeton dataset. The lower the CD value, the better the algorithm's performance.

In the Octopus and Vase categories, the TPA was outperformed by only one state-of-the-art algorithm and was outperformed by only two algorithms in the Ant and Fish categories. The TPA algorithm, however, had the least accurate performance in the Plier and Bird categories. The TPA algorithm managed to outperform the k-means and the Core Extraction algorithms overall on the whole dataset.

5.4.2.2 HD evaluation

Sub-figure 5.12(b) shows how the TPA algorithm compared with the state-of-the-art algorithms, using the HD metric. Tables B.2, B.3 and B.4 show the HD, HD-Rm and HD-Rf values of all the algorithms. The lower the values, the better the algorithm's performance.

The TPA algorithm managed to outperform the k-means and Fitting Primitives algorithms relative to overall HD values. The TPA algorithm managed to outperform the same two algorithms relative to the HD-Rm metric; however, the TPA algorithm also outperformed the Core Extraction and Random Walks relative to the HD-Rf metric.

In segmenting the Fish category, The TPA algorithm managed to get the same HD score as the Shape Diameter algorithm but managed to outperform all the other state-of-the-art algorithms. In the Four-leg and Vase categories, the TPA algorithm was outperformed by only one other algorithm and was outperformed by only two other algorithms in the Aeroplane, Ant, Octopus, Armadillo and Bust categories. The TPA algorithm, however, had the least accurate performance in the Cup, Glasses and Table categories.

5.4.2.3 RI evaluation

Sub-figure 5.12(c) shows how the TPA algorithm compared with the state-of-the-art algorithms, using the RI metric. Table B.5 gives the average $1-\text{RI}$ values of each algorithm per category. The lower the $1-\text{RI}$ value, the better the algorithm's performance.

In segmenting the Octopus class, the TPA algorithm produced the best score alongside the Shape Diameter algorithm, to outperform all the other state-of-the-art algorithms. In the Fish and Vase categories, it was outperformed by only one other algorithm and in the Teddy and Ant categories, it

was outperformed by only two other algorithms. The TPA algorithm, however, had the least accurate performance in the Cup, Table, Plier, Bust, Mech and Chair categories. The TPA algorithm managed to perform better than the *k*-means algorithm overall on the whole dataset.

5.4.2.4 CE evaluation

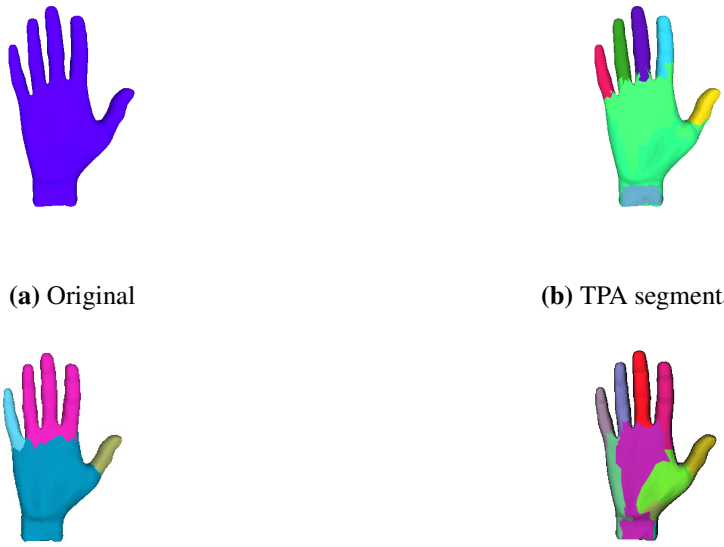
Sub-figure 5.12(d) shows how the TPA algorithm compared with the state-of-the-art algorithms, using the CE metric. Table B.6 gives the GCE scores of the segmentation results and Table B.7 gives the LCE scores. The lower the CE value, the better the algorithm's performance.

Relative to the GCE metric, the TPA algorithm outperformed the Normalised Cuts, Random Walks, Fitting Primitives and *k*-means algorithms on the overall dataset. The TPA algorithm outperformed all the state-of-the-art algorithms in segmenting the Cup category. The TPA algorithm was outperformed by only one algorithm in the Aeroplane, Octopus, Armadillo, Mech and Vase categories. In the Human, Ant, Table, Teddy and Four-leg categories, the TPA algorithm was outperformed by only two other algorithms. The TPA algorithm did not have the least accurate performance in any category.

The TPA algorithm performed better than Normalised Cuts, Random Walks, Fitting Primitives and *k*-means algorithms when evaluated using the LCE metric on the whole dataset. The TPA algorithm outperformed all the other algorithms relative to the LCE metric in the Bearing and Four-leg categories. Only one algorithm outperformed the TPA algorithm in the Octopus and Vase classes, whilst in the Cup, Aeroplane, Ant and Bust categories, only two state-of-the-art algorithms outperformed the TPA algorithm. The TPA algorithm, however, had the least accurate performance in the Hand category.

5.4.2.5 Coloured images of segmentation areas

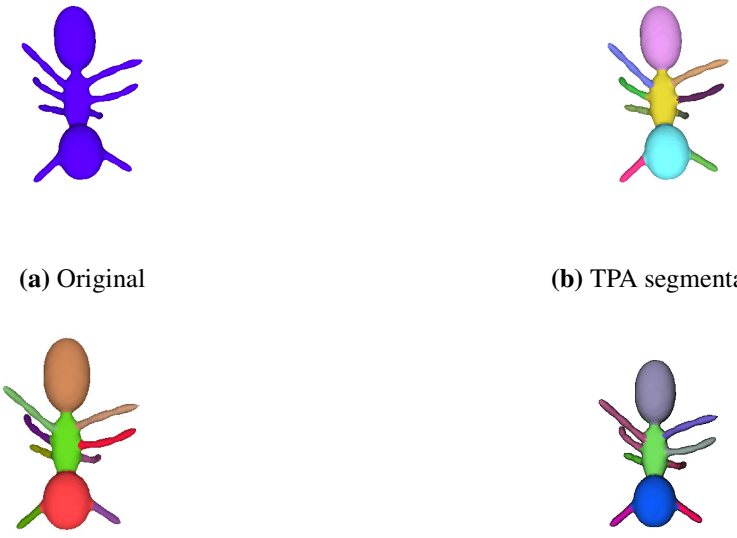
Figures 5.13 and 5.14 show images of the qualitative segmentation results that were produced by the TPA algorithm on the Hand and Ant CAD models, respectively. Each unique segment is represented by a unique colour. Additional coloured segmentation results are given in Addendum C.



(a) Original **(b)** TPA segmentation

(c) Shape Diameter segmentation **(d)** Fitting Primitives segmentation

Figure 5.13. Qualitative segmentation results of a CAD model from the Hand category in the Princeton dataset [65].



(a) Original **(b)** TPA segmentation

(c) Shape Diameter segmentation **(d)** Fitting Primitives segmentation

Figure 5.14. Qualitative segmentation results of a CAD model from the Ant category in the Princeton dataset [65].

The qualitative analysis shows that the TPA algorithm produced comparable results to those of the state-of-the-art algorithms. Figure 5.13 shows how the TPA algorithm managed to uniquely separate the individual fingers of the Hand from one another. The results also show how the main part of the hand was correctly classified as a single distinct segment. Figure 5.14 shows how the main body of the Ant was correctly partitioned into 3 segments, whilst the legs and antlers were correctly grouped into 8 distinct segments.

5.4.2.6 Summary

The results show that the TPA algorithm scores within the range of scores of the state-of-the-art algorithms for all the metrics. For each of the metrics, the TPA algorithm manages to outperform at least one of the state-of-the-art algorithms. Its best ranking can be observed when evaluated using the CE metric, the TPA algorithm manages to outperform four out of the seven state-of-the-art algorithms.

In segmenting the individual categories, in most cases, the TPA algorithm manages to outperform at least one of the state-of-the-art algorithms and in other cases, it manages to outperform all of the state-of-the-art algorithms. These results confirm that the TPA algorithm produces segmentation results that are comparable with those of the state-of-the-art algorithms.

5.4.3 Evaluation using meshes of real-world landscapes

5.4.3.1 Overview

The analysis that was provided in Section 5.4.2 shows that the TPA algorithm performs well on the Princeton benchmark dataset. The SDF similarity attribute of the faces of the meshes was used to perform the segmentation since the meshes in the dataset do not have a colour attribute. Since, in the preprocessing pipeline of the VR application, the algorithm uses the colour attribute of the mesh to perform segmentation, the next experiments intend to evaluate how the TPA algorithm performs on coloured meshes of landforms, compared with the performance of two state-of-the-art-algorithms, namely the Shape Diameter segmentation algorithm (SD) and the Fitting Primitives (FP) algorithm. The two algorithms were selected since their implementations that were used in the Princeton Benchmark

analysis were readily available. The SD and FP algorithms did not use the colour attribute of the meshes but used both orientation and geometric attributes of the mesh, as described in Chapter 2.

Due to the unavailability of a segmentation benchmark dataset of coloured meshes of landforms, the three algorithms were evaluated using sections of meshes that were generated from the Swiss quarry dataset. In the experiment, the ground-truth segmentations were manually created for each of the meshes. The Rand Index (RI) metric was used to evaluate the algorithms; the metric was selected since it does not rely on finding the boundaries and regions of the segments, tasks that may be computationally too expensive due to the complex nature of the meshes.

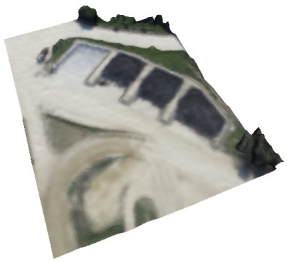
5.4.3.2 Experimental results

In the following experiments, a coloured input mesh was presented to each of the three algorithms for segmentation. For the TPA algorithm, s_i was evaluated using (3.3). Th was kept at a constant 0.7 and sTh was kept at a very low constant value of 0.015. When $Th = 0.7$, shades of the same coloured faces are grouped correctly into the same *level 2 pools*. Higher values will under-segment the mesh and lower values will over-segment the mesh. A low value of $sTh = 0.015$ ensures that large *level 3 pools* are not merged.

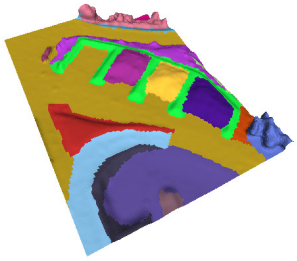
The SD algorithm was executed using the available implementation in the Computational Geometry Algorithms Library (CGAL), called Surface Mesh Segmentation [83], and the FP algorithm [62] was executed using its freely available executable. Figures 5.15, 5.16 and 5.17 show the qualitative results of segmenting 3 scenes, *scene 1*, *scene 2* and *scene 3* of meshes generated from the Swiss quarry dataset [36]. Tables 5.1, 5.2 and 5.3 show the corresponding tables of 1−RI scores.

Table 5.1. 1−RI scores of the algorithms on *scene 1*

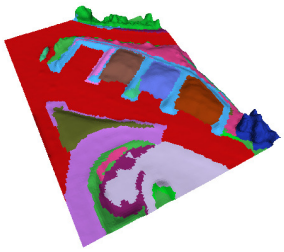
Algorithm	1−**RI**
TPA	0.017
SD	0.312
FP	0.121



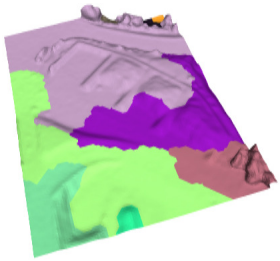
(a) *scene 1*, a section of a mesh generated from the Swiss quarry dataset [36]



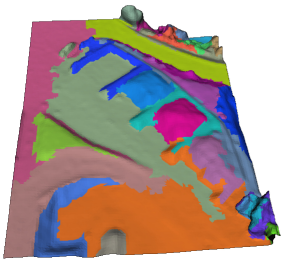
(b) The ground-truth segmentation



(c) Segmentation results of the TPA algorithm



(d) Segmentation results of the SD algorithm

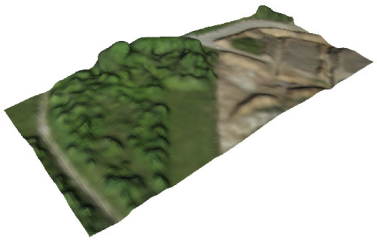


(e) Segmentation results of the FP algorithm

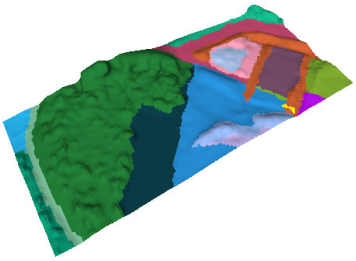
Figure 5.15. Segmentation results of *scene 1*, a section of a mesh generated from the Swiss quarry dataset [36].

Table 5.2. 1−RI scores of the algorithms on *scene 2*

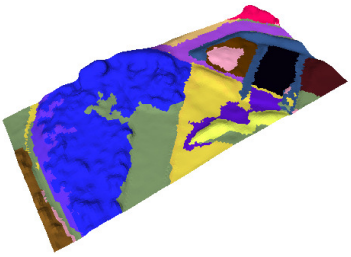
Algorithm	1−**RI**
TPA	0.052
SD	0.372
FP	0.221



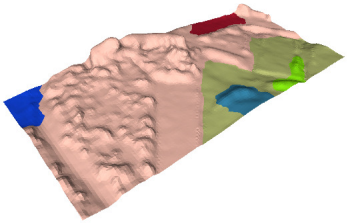
(a) *scene 2*, a section of a mesh generated from the Swiss quarry dataset [36]



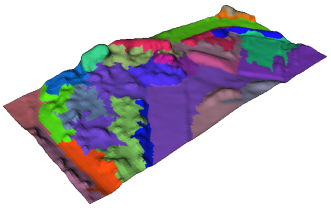
(b) The ground-truth segmentation



(c) Segmentation results of the TPA algorithm



(d) Segmentation results of the SD algorithm



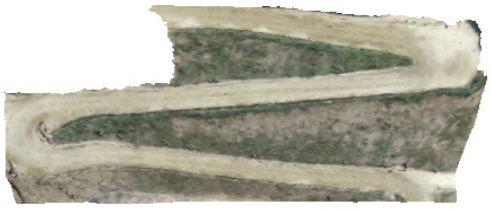
(e) Segmentation results of the FP algorithm

Figure 5.16. Segmentation results of *scene 2*, a section of a mesh generated from the Swiss quarry dataset [36].

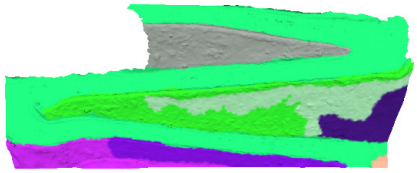
Table 5.3. $1-$RI scores of the algorithms on *scene 3*

Algorithm	$1-$**RI**
TPA	0.195
SD	0.330
FP	0.121

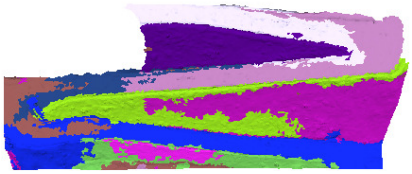
Inspection of the results showed that the TPA algorithm managed to correctly separate the regions that made up the meshes. The major segments of the meshes were separated from one other; for instance, paths or roads in the landscapes were correctly identified by the algorithm. Regions of vegetation and dirt were also correctly segmented. The TPA algorithm; however, failed to handle shadows that were



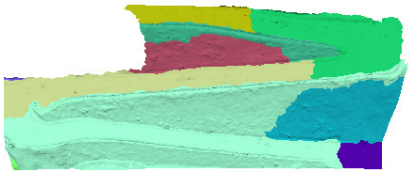
(a) *scene 3*, a section of a mesh generated from the Swiss quarry dataset [36]



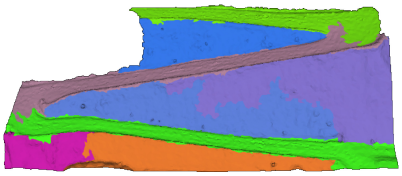
(b) The ground-truth segmentation



(c) Segmentation results of the TPA algorithm



(d) Segmentation results of the SD algorithm



(e) Segmentation results of the FP algorithm

Figure 5.17. Segmentation results of *scene 3*, a section of a mesh generated from the Swiss quarry dataset [36].

present in parts of the meshes. The segmentation due to the SD algorithm was under-segmented. The results show that the FP algorithm outperformed the SD algorithm and its performance was comparable to that of the TPA algorithm. The results shown in Figure 5.17 illustrate a case where the FP algorithm outperformed the TPA algorithm.

5.4.3.3 Summary

The TPA algorithm manages to identify the distinct regions of the meshes due to the regions' differences in colour. The TPA algorithm is designed to separate regions that can be represented by an attribute that is homogeneous to one region but different in another.

In Section 5.4.2, it was seen that the SD algorithm produces the best results when segmenting the CAD models in the Princeton dataset compared to all the other state-of-the-art algorithms. However, the

algorithm produces poor results when segmenting meshes of real-world landscapes. Apart from the SDF attribute, the SD algorithm also uses other surface attributes, such as the dihedral angle between neighbouring faces and the faces' curvature. The poor segmentation results can, therefore, be attributed to the complex nature of the surfaces, such that using surface attributes for segmentation leads to poor results.

The FP algorithm attempts to segment the meshes by fitting geometrical primitives onto the meshes' surfaces. In the cases where the regions of the meshes show a high level of shape regularity, the FP algorithm performs well. However, when the meshes exhibit a low level of shape regularity, the algorithm performs poorly. The major disadvantage of the FP algorithm relative to the intended application is the need to specify the expected number of segments beforehand.

5.4.4 Run-time performance of the TPA algorithm

The following section provides an investigation of the run-time of the TPA algorithm. Two aspects of particular interest are how the run-time is affected by the size of the mesh and how it is affected by multiprocessing. Multiprocessing is introduced during the creation of *level 3 pools*. Table 5.4 shows the run-time of the algorithm on differently-sized meshes.

Table 5.4. Run-times of the TPA algorithm on differently-sized meshes

Number of faces	**Run-time without multiprocessing (hr)**	**Run-time with multiprocessing (10 cores) (hr)**
49000	0.05	0.008
93435	0.12	0.05
382455	27	1
1000000	72	3
5000000	120	10
12000000	192	22

It can be seen that as the size of the mesh grows, the run-time increases. Since the search space increases with an increase in the size of the mesh, the execution time of the greedy algorithm also increases,

hence the overall increase in run-time. Instead of sequentially searching the *level 2 pools* using the greedy algorithm, multiprocessing allows multiple *level 2 pools* to be searched simultaneously, thereby reducing the run-time significantly.

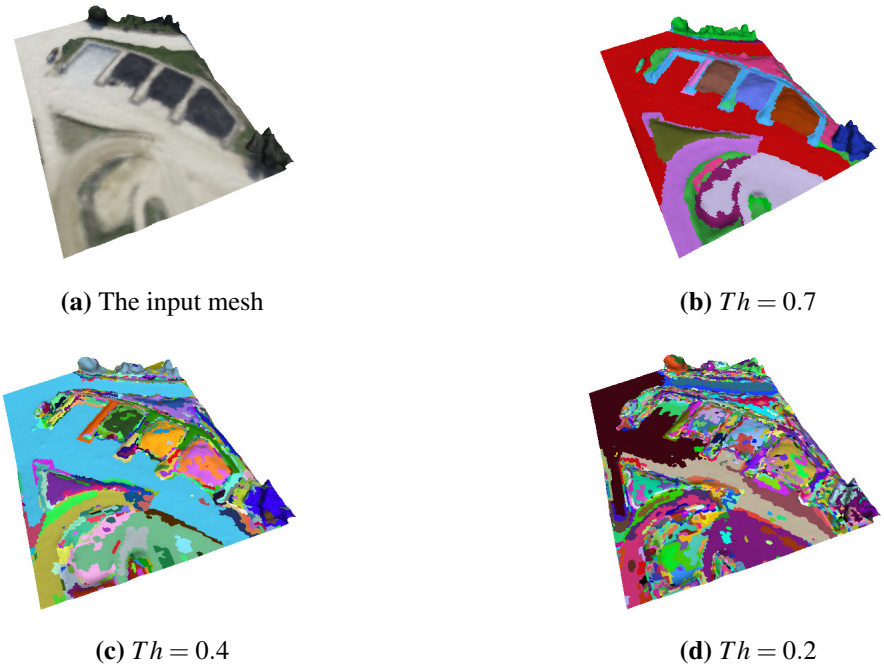
5.4.5 Using the TPA algorithm to extract multiple segments in a real-world landscape

The experimental analysis that has been presented has shown that the TPA algorithm produces comparable results to those of the state-of-the-art algorithms on the Princeton benchmark and better results on meshes of real-world landforms. Unlike most of the state-of-the-art algorithms, the TPA algorithm is also fully automated and does not require external interaction with a user.

For the particular case of the VR application, the segments of the high-resolution mesh need to be small enough to avoid high memory and loading costs. To make sure that the size of the meshes is kept low, the segments that are produced by the TPA algorithm need to be broken down further into even smaller segments.

In principle, using lower values of Th and setting sTh to 0 should increase the number of segments produced; a decrease in Th means a lower similarity threshold, hence the formation of more *level 2 pools*. If sTh is set to 0, the step of merging *level 3 pools* is skipped. The net effect is that more and smaller segments are produced. To illustrate the concept, Figures 5.18, 5.19 and 5.20 show the varying sizes of the segments that were produced as Th was decreased whilst sTh was set to 0.

A lower value of Th results in a larger value of k for the k-means clustering step during the formation of *level 2 pools*. As a result, more segments are created, as opposed to when Th is high. A Th value of 0.7 produces segments that more accurately represent the ground-truth. However, such kinds of segments are too large for the smooth operation of the VR application. When Th is set to 0.2, the segments become small enough to maintain a low overhead on the system's performance. Setting $sTh = 0$ ensures that the small segments are not merged to form larger segments. Therefore, for the preprocessing pipeline, the TPA algorithm was set to use a Th value of 0.2 and an sTh value of 0.



(a) The input mesh

(b) $Th = 0.7$

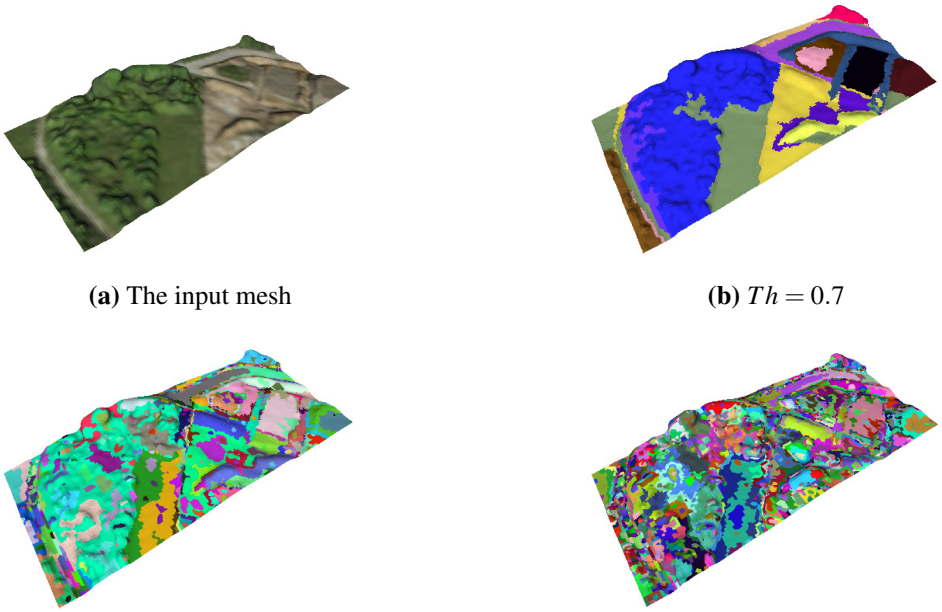
(c) $Th = 0.4$

(d) $Th = 0.2$

Figure 5.18. Segmentation results of *scene 1*, shown in Figure 5.15, at different Th values of the TPA algorithm.

5.5 SUMMARY

This chapter provided the experimental work that was conducted for the research. The experiments were intended to justify the use of the selected algorithms in the preprocessing pipeline of the VR application. The experiments show that the algorithms produced acceptable results that justified their use. Chapter 5 provides a discussion of the results obtained in this chapter. The architecture illustrating how the individual preprocessing steps are combined for the overall preprocessing pipeline will also be shown in Chapter 5.



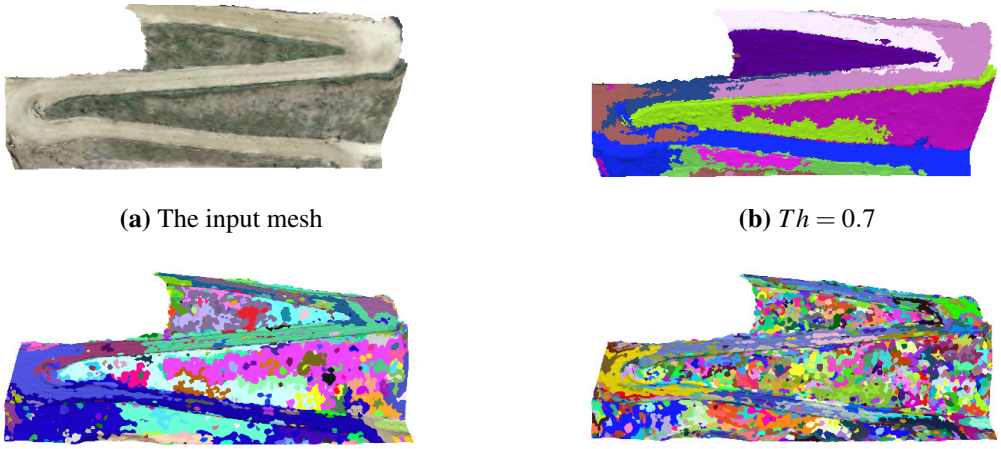
(a) The input mesh

(b) $Th = 0.7$

(c) $Th = 0.4$

(d) $Th = 0.2$

Figure 5.19. Segmentation results of *scene 2*, shown in Figure 5.16, at different Th values of the TPA algorithm.



(a) The input mesh

(b) $Th = 0.7$

(c) $Th = 0.4$

(d) $Th = 0.2$

Figure 5.20. Segmentation results of *scene 3*, shown in Figure 5.17, at different Th values of the TPA algorithm.

CHAPTER 6 DISCUSSION

6.1 CHAPTER OVERVIEW

The focus of the research was to design a preprocessing pipeline that allows the efficient rendering of large landscapes in a VR environment. The pipeline has to ensure that in the final VR application, the size of the meshes rendered at one particular time does not overflow the internal memory, hence, reducing the computational power required to execute the application. The method also aimed to avoid virtual reality induced sickness.

The proposed solution was to design a pipeline that can convert a large point cloud into a format that can be loaded as a landscape in a VR application. The following steps were identified as requirements to achieve such a task:

1. Convert the point cloud into a mesh;
2. Texture the mesh;
3. Segment the mesh.

The following sections discuss how these three areas were handled in the research and if the observed results were satisfactory for the intended application. Sections 6.2, 6.3 and 6.4 provide an analysis of the surface reconstruction, texturing and segmentation's experimental results, respectively. Section 6.5 provides a concise discussion of the architecture of the final pipeline.

6.2 SURFACE RECONSTRUCTION

6.2.1 Overview

A raw point cloud is a collection of points in 3D space that represents a real-world object. A point cloud, however, does not occupy the whole volume of space as is the case with the actual real-world object. Point clouds, therefore, represent a sample of the actual objects that they represent.

Surface reconstruction methods attempt to infer the complete volume of space that is represented by a point cloud. The output of surface reconstruction algorithms is a data structure referred to as a mesh, that, in addition to the information of the points, also contains the information on how to triangulate these points into solid polygons. Each polygon, also referred to as a face, occupies a volume of space that otherwise is empty in a raw point cloud.

In the research, the reconstructed mesh had to be visually accurate (similar to the actual real-world object) and its properties had to satisfy the requirements of the VR application. The properties included the need to preserve the colour attribute of the mesh and the need for the surface to be watertight. The experimental results showed that the Poisson surface reconstruction algorithm satisfied all of the three requirements, unlike the other state-of-the-art algorithms that were considered. The algorithm was, therefore, selected to carry out the surface reconstruction task for the VR application's preprocessing pipeline.

The results showed that the Marching cubes, Voronoi filtering and Fast surface reconstruction algorithms all failed to preserve the vertex-colours of the resulting mesh, hence they were all removed from consideration. The colour attribute is a key input in the TPA segmentation relative to the VR application.

A direct application of Delaunay triangulation on the points resulted in meshes that were watertight and preserved vertex-colours but were not visually accurate. The landscape environments in the VR application need to be as similar as possible to the real-world landscapes that they are derived from. Failing the test, automatically eliminated Delaunay triangulation from consideration.

Even though the Ball Pivoting algorithm managed to pass the accuracy and vertex-colour preservation

tests, the meshes that were generated would at times not be watertight; that is, holes were present in parts of the mesh. The problem can be fixed by adjusting the ball radius parameter of the algorithm; however, the radius is dependent on the particular point cloud being processed. Such an approach would entail the undesirable approach of a user having to adjust the ball radius iteratively until the required result is achieved for each point cloud that is presented to the system. The presence of holes, or empty unoccupied spaces, is unwanted for a mesh that is supposed to represent the real-world. Such a scenario removes the realistic feel of the VR environment.

In addition to the Poisson surface reconstruction algorithm being able to produce meshes that satisfy the requirements of the VR application, the results also show that the algorithm can facilitate the creation of lower and higher resolution meshes using the same point cloud. This is achieved by using two different parameter values of the octree depth attribute of the algorithm: a larger value for the higher resolution mesh and a lower value for the lower resolution mesh. If the Poisson surface reconstruction algorithm had not been able to do this, the higher resolution mesh would need to be generated from a larger-sized point cloud and the lower resolution mesh would need to be generated from a separate smaller-sized, sampled version. Disadvantages that can arise from such a scenario:

- There would be a need to identify or design an additional sampling algorithm that produces an accurate lower resolution version of the original point cloud.
- Since the surface reconstruction would be performed on two separate point clouds, their alignment would not be guaranteed. There would, therefore, be a need to design an accurate alignment algorithm to align the lower and higher resolution meshes.

In conclusion, the Poisson surface reconstruction algorithm produced satisfactory results relative to the intended VR application, hence justifying its selection for the surface reconstruction step.

6.2.2 Advantages of the Poisson surface reconstruction algorithm relative to the VR application

The advantages of using the Poisson surface reconstruction algorithm in the pipeline are as follows:

- It preserves the colour information.
- It produces a watertight surface.
- It produces accurate results.
- It can generate meshes with different LOD using the same point cloud, by just adjusting its octree depth parameter value.

6.2.3 Disadvantages of the Poisson surface reconstruction algorithm relative to the VR application

The disadvantages of using the Poisson surface reconstruction algorithm in the pipeline are as follows:

- If the input point cloud has missing data, the algorithm may produce distorted meshes.
- If the normals of the point clouds' vertices are not accurate, the algorithm may produce distorted meshes.
- The outer edges of the output are generally distorted.

6.3 TEXTURING

6.3.1 Overview

When a mesh is rendered in a 3D rendering engine, it can be colourised by either using its vertex-colours or its texture map. The experimental results showed that a textured mesh provided a more realistic feel to the mesh compared to the vertex-coloured version. The VR application needs to provide an environment that is realistic to the user, hence, an additional texturing step in the system's pipeline is necessary.

Texturing in the rendering engine involves mapping each vertex of the mesh to a coordinate on a texture map. Such a mapping is performed by a process known as UV mapping. UV mapping algorithms follow two configurations, a face to texture mapping or a vertex to texture mapping. Unity and Unreal engines support only the vertex to texture mapping configuration, however, the algorithm from Pix4Dmapper [35], that was used to generate the texture map, supported the face to texture

mapping configuration. A correction step was, therefore, introduced to convert the per-face mapping to a per-vertex mapping.

Generation of the texture map can be carried out using two approaches:

1. Using the original images;
2. Using the vertex-colours of the mesh.

Observations show that the texture map that is generated using the mesh's vertex-colours is not ideal due to data loss during the photogrammetry process. The vertex-colours alone are, therefore, not ideal to produce the required texture map. Generating from the original images, however, produces texture maps that match the requirements. In the pipeline, the texture maps are, therefore, generated using the original images.

6.3.2 Advantages of texturing, using the original images

The advantages of creating a texture map for the mesh using the original images are as follows:

- The resulting texture maps are accurate despite the data loss during point cloud generation and surface reconstruction.
- The textures provide a realistic feel to the mesh compared to those that are produced by using vertex-colours.

6.3.3 Disadvantages of texturing, using the original images

The disadvantages of creating a texture map for the mesh using the original images are as follows:

- If the mesh has a very low polygon-count, parts of the textured areas may appear blurry in the rendering engines.
- If the UV mapping is not in a format compatible with the rendering engine, the textured surfaces will be distorted.

6.4 SEGMENTATION

6.4.1 Overview

Since all the data that is contained in a large detailed mesh cannot be feasibly loaded into the VR application due to memory and performance constraints, a solution was proposed to load the data in parts, loading only the relevant parts at a time. To achieve the task, a proposal was made to segment the higher resolution mesh into meaningful parts that can be loaded individually at relevant times and discarded when no longer needed. The relevant time to load a set of segments is when a user in the VR application is located within a prescribed radius of the segments' location; the segments are discarded when the user moves away from within their radius.

The existing state-of-the-art algorithms do not produce accurate segments that are compatible with the requirements of the VR application. As seen in the experiments and results section, other state-of-the-art algorithms depend on geometric features that are too complex to efficiently segment meshes of landscapes. As seen from the results and the literature study, other state-of-the-art algorithms depend on an external user to specify the expected number of segments. However, it is infeasible for an external user to accurately determine the number of segments in the large and complex meshes.

To overcome the shortcomings of the existing algorithms, a new segmentation algorithm, referred to as the Triangle Pool Algorithm (TPA), was designed. The algorithm depends on two parameters, the threshold values Th and sTh. When sTh is kept at a value of 0 and Th at a value of 0.2, the algorithm produces segments that are compatible with the requirements of the VR application. The algorithm can, therefore, be automated without the need for external interaction to change the threshold values.

The TPA algorithm was designed to segment meshes of landscapes for the VR application. However, the algorithm can also be used as a segmentation algorithm for other kinds of meshes. As seen in the experiments, the SDF attribute was used to successfully segment the CAD objects in the Princeton benchmark dataset, producing results that were comparable with those of the state-of-the-art algorithms.

The running time of the TPA algorithm is affected by the size of the mesh. As the size of the mesh increases, the time it takes for the algorithm's completion becomes extremely high. To significantly

reduce the run-time, the algorithm is executed in a multiprocessing configuration. Even with multiprocessing, the run-time for segmenting large meshes is still relatively high, however, since this is a preprocessing step done before executing the VR application, the time taken to perform the actual segmentation does not affect the run-time of the VR application.

6.4.2 Advantages of the TPA algorithm

The advantages of the TPA algorithm are as follows:

- The algorithm requires minimal user interaction.
- By changing the value of Th, the level of segmentation that is required can be controlled.
- The algorithm can use a similarity metric based on any homogenous property of a mesh.

6.4.3 Disadvantages of the TPA algorithm

The disadvantages of the TPA algorithm are as follows:

- Erroneous results can be obtained if the surface reconstruction is poorly done since the contiguity of the faces will be incorrect.
- The run-time is relatively high.

6.5 EFFICIENTLY RENDERING LARGE LANDSCAPES IN VR

After successfully setting up the previously discussed pipeline, large landscapes can be rendered in the VR visualising application. The rendering process can be summarised as follows:

- A low-resolution mesh, generated from a point cloud of the real-world by the Poisson surface reconstruction algorithm, is loaded as the base environment, The need to render meshes based on the user's field-of-view is eliminated since regions that are in any possible field-of-view are already rendered; the risk of virtual reality induced sickness is also, therefore, reduced. The low-resolution mesh is generated such that it is small enough to avoid memory overflow during rendering.

- Texturing adds a realistic feel to the environment by adding details that are present in the original images of the scene.
- Segmentation partitions the high-resolution mesh of the real-world scene into multiple meaningful parts. Only a certain fraction of the meshes are loaded from external storage at a given time depending on the location of the VR user. The number of meshes that are loaded at a given time is controlled so as not to overflow the memory.
- The result is that the regions that are near the user have a high LOD, whilst those further away have a low LOD. The total size of meshes that are loaded at a given time does not overflow the memory and the performance is optimised.

6.6 SUMMARY

A combination of existing and new methods were used to design the algorithms that are used in the preprocessing pipeline of the VR application. The experimental results showed that the methods were good enough to meet the requirements of the research. Existing surface reconstruction and texturing methods were selected since they produced adequate results.

Since the state-of-the-art algorithms do not produce suitable results for the preprocessing pipeline, the TPA algorithm was designed to solve the mesh segmentation problem. The TPA algorithm not only produces suitable results for the VR application but also produces results that are comparable to those of the state-of-the-art algorithms.

CHAPTER 7 CONCLUSION

7.1 CHAPTER OVERVIEW

The research aimed to design a pipeline that addressed the preprocessing requirements of a VR landmark viewing application. The VR application was designed to allow the user to move from one location to another in a virtual environment. To design the VR application, the memory and performance aspects of the system had to be considered due to the large memory size taken up by the real-world data.

7.2 RESEARCH FINDINGS

The research aimed to design a method to render large sized environments in the VR application, whilst preserving performance and scalability. The environments were created from 3D point clouds of the real-world. Research questions that were presented in Chapter 1 are as follows:

1. How can the raw point cloud be converted into a 3D mesh, suitable for VR rendering?
2. How can the mesh be texture mapped to make it visually realistic in VR?
3. How can the mesh be optimally rendered in a VR application?

The following sections show how the research questions were answered.

7.2.1 Point Cloud to Mesh conversion

The process of surface reconstruction served the purpose of answering the first research question. In the literature study that was presented in Chapter 2, it was shown that there are various methods to reconstruct a point cloud into a mesh. The justification for the use of the Poisson surface reconstruction method [30] in the research was provided in Chapter 3. In Chapter 4, it was shown how the Poisson surface reconstruction algorithm effectively transforms point clouds of real-world environments into triangular-faced meshes.

7.2.2 Texture mapping

Chapter 2 introduced the concept of mapping texture onto a 3D mesh. In Chapter 3, it was shown that a suitable method of generating a texture map involved retrieving the texture coordinates from the images that were used to create the initial point cloud of the real-world environment. In Chapter 4, it was shown how texture mapping significantly improved the quality of the rendered mesh, hence, making it visually realistic.

7.2.3 Optimally rendering the mesh

As discussed in Chapter 2, it was impossible to render meshes with sizes larger than the internal memory. To solve the problem, a rendering architecture involving two versions of the same mesh was proposed; a mesh with a smaller memory size and the original mesh with a larger size. The lower resolution mesh is loaded at all times during the operation of the VR application, whilst segments of the higher resolution mesh are only loaded near the VR user. The architecture ensures that the size of the meshes that are loaded at a particular time maintain the operation of the VR application within optimal operating conditions.

The architecture led to the need for partitioning the larger mesh into multiple segments. A novel segmentation algorithm was introduced in Chapter 3 to solve the segmentation problem. The justification of using the segmentation algorithm instead of existing state-of-the-art algorithms was given in Chapter 4.

7.2.4 Operation of the VR application

The operation of the VR application is detailed in the following steps:

1. On startup, the mesh with a lower LOD is loaded into the application.
2. Segments of the higher LOD mesh are loaded within a radius of a location where the user will be initially placed.
3. The VR user is initially placed in the prescribed location of the environment.
4. When the user teleports to another location of interest, the user's new position is calculated and the segments of the higher LOD mesh within a specific radius of the new position are loaded into the application.
5. The segments that had been loaded before the teleportation are removed.

The research hypothesis can, therefore, be accepted: A large point cloud of a real-world landscape can be transformed into a mesh that can be used to simulate the real-world landscape in a VR application by loading only segments of that mesh that are within proximity of the VR user.

7.3 FUTURE WORK

Even though the objectives of the research were met, certain aspects may provide possible research gaps in the future:

- The run-time of the TPA segmentation algorithm was high for large meshes. Future work may explore ways to reduce the run-time, by exploring more efficient designs.
- The TPA segmentation algorithm provides an algorithm to automatically set the number of expected *level 2 pools*. Future work may improve on the current algorithm by exploring the use of machine learning clustering techniques that do not need a user-specified number of clusters.
- Even though the Poisson surface reconstruction algorithm provides relatively good results, future work may be carried out to find ways to improve on how the algorithm handles edges of the point cloud.
- Future work may focus on recognition of objects during run-time of the VR application. Such a system could be used in applications where the VR user needs to be given information related to

particular regions that they would be viewing. Much research has been done concerning 3D object recognition; however, the majority of research focuses on identifying known, geometrically planar objects. The distinct regions of real-world meshes are generally not geometrically well-defined and are usually complex structures. Such an aspect opens a window for such a research gap.

- The effectiveness of the VR application in reducing the risks of VR induced sickness can be further investigated in a study involving human subjects.
- The overall real-time performance of the VR application can be further investigated.
- How the preprocessing pipeline scales to meshes of various complexities can be explored in a future study.
- An investigation into how the proposed technique can be used with mobile VR applications can be done.
- Future work can explore the possibility of introducing additional meshes of intermediate LOD instead of the VR application just using the high and low LOD meshes.

7.4 SUMMARY

The research objectives were met, research questions were answered and the research hypothesis was accepted. A pipeline to convert a point cloud into an environment suitable for a VR application was successfully designed. Two VR applications were designed, using the Unity and Unreal engines. Both applications were compatible with the architecture that was presented in the research.

Successfully loading large environments into the VR application was the first step in creating a full-scale, interactive, real-time VR application. Since the application allows users to roam around real-world virtual environments, further interactive simulations can be designed.

REFERENCES

[1] L. Zhang and L. Zhang, "Deep learning-based classification and reconstruction of residential scenes from large-scale point clouds," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 4, pp. 1887–1897, April 2018.

[2] S. Mandal, "Creating textured 3D models from image collections using open source software," *International Journal of Scientific and Engineering Research*, vol. 4, pp. 304–309, April 2013.

[3] G. Mingming and Z. Keping, "Virtual reality simulation system for underground mining process," in *ICCSE 2009, Proceedings of the International Conference on Computer Science and Education*, August 2009, pp. 1013 – 1017.

[4] M. Kizil, "Virtual reality applications in the Australian minerals industry," in *APCOM 2003, Application of Computers and Operations Research in the Minerals Industries*, May 2003, pp. 569–574.

[5] "Unity user manual (2019.2)," Available: https://docs.unity3d.com/Manual/index.html, Accessed on: Mar. 12, 2019. [Online].

[6] "Unreal Engine," Available: https://www.unrealengine.com, Accessed on: Mar. 15, 2019. [Online].

[7] N. Singh and S. Singh, "Virtual reality: A brief survey," in *ICICES 2017, International Conference on Information Communication and Embedded Systems*, Feb 2017, pp. 1–6.

[8] W. Cheng, W. Lin, X. Zhang, M. Goesele, and M. Sun, "A data-driven point cloud simplification framework for city-scale image-based localization," *IEEE Transactions on Image Processing*, vol. 26, no. 1, pp. 262–275, Jan 2017.

[9] J. S. Vitter, "External memory algorithms and data structures: Dealing with massive data," *ACM Computing Survey*, vol. 33, no. 2, pp. 209–271, Jun. 2001.

[10] L. Rebenitsch and C. Owen, "Review on cybersickness in applications and visual displays," *Virtual Reality*, vol. 20, no. 2, pp. 101–125, Jun 2016.

[11] V. Vishal and E. Walia, "3D rendering - techniques and challenges," *International Journal of Engineering and Technology*, vol. 2, pp. 29–33, April 2010.

[12] O. Brooks, R. Goodenough, C. Crisler, D. Klein, L. Alley, L. Koon, C. Logan, J. Ogle, A. Tyrrell, and F. Wills, "Simulator sickness during driving simulation studies," *Accident Analysis and Prevention*, vol. 42, pp. 788–96, May 2010.

[13] A. G. Vladimir, "Point clouds registration and generation from stereo images," *International Journal Information Content and Processing*, vol. 3, no. 2, pp. 193–199, 2016.

[14] A. Maiti and D. Chakravarty, "Performance analysis of different surface reconstruction algorithms for 3D reconstruction of outdoor objects from their digital images," *SpringerPlus*, vol. 5, p. 932, June 2016.

[15] F. Rottensteiner, "Automatic generation of high-quality building models from LIDAR data," *IEEE Computer Graphics and Applications*, vol. 23, no. 6, pp. 42–50, Nov 2003.

[16] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment - a modern synthesis," in *ICCV 1999, Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, 2000, pp. 298–372.

[17] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE Transactions on MultiMedia*, vol. 19, no. 2, pp. 4–10, Feb 2012.

[18] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, pp. 1–104, May 2015.

[19] F. Zeng and R. Zhong, "The algorithm to generate color point-cloud with the registration between panoramic image and laser point-cloud," in *IOP Conference Series: Earth and Environmental Science*, vol. 17, March 2014, pp. 12–19.

[20] A. Alouache, X. Yao, and Q. Wu, "Creating textured 3D models from image collections using open source software," *International Journal of Computer Applications*, vol. 163, pp. 14–19, April 2017.

[21] H. Badino, D. Huber, Y. Park, and T. Kanade, "Fast and accurate computation of surface normals from range images," in *ICRA 2011, IEEE International Conference on Robotics and Automation*, May 2011, pp. 3084–3091.

[22] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A survey of surface reconstruction from point clouds," *Computer Graphics Forum*, vol. 36, no. 1, pp. 301–329, January 2017.

[23] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, June 1980.

[24] P. Su and R. Drysdale, "A comparison of sequential Delaunay algorithms," *Computational Geometry*, no. 7, pp. 361–385, 1997.

[25] F. Hurtado, M. Noy, and J. Urrutia, "Flipping edges in triangulations," *Discrete and Computational Geometry*, vol. 22, no. 3, pp. 333–346, October 1999.

[26] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, July 1987.

[27] N. Amenta and M. Bern, "Surface reconstruction by Voronoi filtering," *Discrete and Computational Geometry*, vol. 22, no. 4, pp. 481–504, December 1999.

[28] F. Aurenhammer, "Voronoi diagrams - A survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, December 1991.

[29] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," in *TVCG 1999, IEEE Transactions on Visualisation and Computer Graphics*, October 1999, pp. 349–359.

[30] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the EG/SIGGRAPH Symposium on Geometry Processing*, June 2006, pp. 61–70.

[31] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy datasets," in *ICRA 2009, IEEE International Conference on Robotics and Automation*, May 2009, p. 2829–2834.

[32] M. Gopi and S. Krishnan, "A fast and efficient projection-based approach for surface reconstruction," *High Performance Computer Graphics, Multimedia and Visualisation*, vol. 1, no. 1, pp. 1–12, 2000.

[33] P. S. Heckbert, "Survey of texture mapping," *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 56–67, Nov. 1986.

[34] C. Yuksel, "Mesh color textures," in *HPG 2017, High-Performance Graphics*, July 2017, pp. 1–11.

[35] "Pix4Dmapper," Available: www.pix4d.com/product/pix4dmapper-photogrammetry-software, Accessed on: Apr. 19, 2019. [Online].

[36] "senseFly Datasets: Swiss Quarry," Available: https://www.sensefly.com/education/datasets/?dataset=1418, Accessed on: Apr. 19, 2019. [Online].

[37] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, "TextureMontage: Seamless texturing of arbitrary surfaces from multiple images," in *Association for Computing Machinery, Inc.*, August 2005, pp. 99–105.

[38] W. Niem and H. Broszio, "Mapping texture from multiple camera views onto 3D-object models for computer animation," in *The International Workshop on Stereoscopic and Three Dimensional Imaging*, 1995, pp. 99–105.

[39] A. Baumberg, "Blending images for texturing 3D models," in *BMVC 2002, British Machine Vision Conference*, September 2002, pp. 38.1–38.10.

[40] H. M. Nguyen, B. Wünsche, P. Delmas, C. Lutteroth, W. Van der Mark, and E. Zhang, "High-definition texture reconstruction for 3D image-based modeling," in *WSCG 2013, 21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, June 2013, pp. 39–48.

[41] A. Nguyen and B. Le, "3D point cloud segmentation: A survey," in *IEEE 6th Conference on Robotics, Automation and Mechatronics*, Nov 2013, pp. 225–230.

[42] P. Theologou, I. Pratikakis, and T. Theoharis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3D mesh segmentation," *Computer Vision and Image Understanding*, vol. 135, pp. 49–82, June 2015.

[43] A. Agathos, I. Pratikakis, S. Perantonis, and N. Sapidis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3D mesh segmentation," *Computer-Aided Design and Applications*, vol. 4, no. 6, pp. 827–842, Jan 2007.

[44] B. Bhanu, S. Lee, H. Chih-Cheng, and T. Henderson, "Range data processing: Representation of surfaces by edges," in *ICPR 1986, IEEE International Conference on Pattern Recognition*, January 1986, pp. 236–238.

[45] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H. Seidel, "Intelligent mesh scissoring using 3D snakes," in *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*,

April 2004, pp. 279–287.

[46] P. Besl and R. Jain, "Segmentation through variable order surface fitting," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 2, March 1988, pp. 167–192.

[47] P. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, September 2004.

[48] A. Golovinskiy and T. Funkhouser, "Min-cut based segmentation of point clouds," in *ICCV Workshops 2009, IEEE 12th International Conference on Computer Vision Workshops*, Sept 2009, pp. 39–46.

[49] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *ICML 2001, Proceedings of the 18th International Conference on Machine Learning*, Sept 2001, pp. 282–289.

[50] G. Vosselman, B. Gorte, G. Sithole, and B. Rabbani, "Recognising structure in laser scanner point clouds," in *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, October 2004, pp. 33–38.

[51] T. Rabbani, F. Van den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," in *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 35, no. 6, January 2006, pp. 248–253.

[52] Q. Zhan, Y. Liang, and Y. Xiao, "Color-based segmentation of point clouds," *ISPRS Laser Scanning Workshop*, vol. 38, pp. 155–161, July 2009.

[53] J. H. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *ICIRS 2010, IEEE International Conference on Intelligent Robots and Systems*, 2010, pp. 2131–2136.

[54] P. Nishad and R. Chezian, "Various colour spaces and colour space conversion algorithms," *Journal of Global Research in Computer Science*, vol. 4, pp. 44–48, January 2013.

[55] M. Vieira and K. Shimada, "Surface mesh segmentation and smooth surface extraction through region growing," *Computer Aided Geometric Design*, vol. 22, pp. 771–792, 2005.

[56] N. S. Sapidis and P. J. Besl, "Direct construction of polynomial surfaces from dense range images through region growing," *ACM Transactions on Graphics*, vol. 14, pp. 171–200, 1995.

[57] Y. Zhang, J. Paik, A. Koschan, M. Abidi, and D. J. Gorsich, "Simple and efficient algorithm for part decomposition of 3D triangulated models based on curvature analysis." in *Proceedings of the International Conference on Image Processing*, vol. 3, January 2002, pp. 273–276.

[58] A. Leonardis, A. Jaklic, and F. Solina, "Superquadrics for segmenting and modeling range data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 11, pp. 1289–1295, Nov 1997.

[59] K. Wu and M. Levine, "3D part segmentation using simulated electrical charge distributions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, pp. 1223 – 1235, December 1997.

[60] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H. Seidel, "Mesh scissoring with minima rule and part salience," *Comput. Aided Geom. Des.*, vol. 22, no. 5, pp. 444–465, July 2005.

[61] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of polyhedral surfaces using decomposition," *Comput. Graph. Forum*, vol. 21, pp. 219–228, September 2002.

[62] M. Attene, B. Falcidieno, and M. Spagnuolo, "Hierarchical mesh segmentation based on fitting primitives," *The Visual Computer*, vol. 22, no. 3, pp. 181–193, March 2006.

[63] K. Inoue, T. Itoh, A. Yamada, T. Furuhata, and K. Shimada, "Face clustering of a large-scale CAD model for surface mesh generation," *Computer-Aided Design*, vol. 33, pp. 251–261, 2001.

[64] R. Liu and H. Zhang, "Segmentation of 3D meshes through spectral clustering," in *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, October 2004, pp. 298–305.

[65] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, no. 3, pp. 1–12, Aug. 2009.

[66] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3D mesh analysis," *ACM Transactions on Graphics*, vol. 27, no. 5, pp. 145–145, Dec. 2008.

[67] Y. Lai, S. Hu, R. Martin, and P. Rosin, "Fast mesh segmentation using random walks," in *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, August 2008, pp. 183–191.

[68] S. Katz, G. Leifman, and A. Tal, "Mesh segmentation using feature point and core extraction," *The Visual Computer*, vol. 21, no. 8, pp. 649–658, Sep 2005.

[69] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, pp. 249–259, Jan 2008.

[70] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool," in *Sixth Eurographics Italian Chapter Conference*, vol. 1, January 2008, pp. 129–136.

[71] "CloudCompare," Available: www.cloudcompare.org/, Accessed on: Jan. 27, 2019. [Online].

[72] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 1–4.

[73] "The Stanford 3D scanning repository," Available: http://graphics.stanford.edu/data/3Dscanrep/, Accessed on: Aug. 06, 2019. [Online].

[74] L. Ma, T. Whelan, E. Bondarev, P. H. N. De With, and J. McDonald, "Planar simplification and texturing of dense point cloud maps," in *ECMR 2013, European Conference on Mobile Robots*, September 2013, pp. 164–171.

[75] "DroneMapper Datasets," Available: https://www.dronemapper.com/sample_data/, Accessed on: Apr. 19, 2019. [Online].

[76] M. Luo, G. Cui, and B. Rigg, "The development of the CIE 2000 colour-difference formula: CIEDE2000," *Color Research and Application*, vol. 26, pp. 340 – 350, October 2001.

[77] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100–108, 1979.

[78] A. Malik, A. Sharma, and V. Saroha, "Greedy algorithm," *International Journal of Scientific and Research Publications*, vol. 3, pp. 55–60, Aug 2013.

[79] "senseFly Datasets: Gravel Quarry," Available: https://www.sensefly.com/education/datasets/?dataset=5577, Accessed on: Apr. 19, 2019. [Online].

[80] "senseFly Terms and Conditions," Available: https://www.sensefly.com/terms-conditions/, Accessed on: Apr. 19, 2019. [Online].

[81] "DroneMapper Terms of Use," Available: https://dronemapper.com/termsofuse/, Accessed on: Apr. 19, 2019. [Online].

[82] H. Chen, C. Fahn, J. Tsai, R. Chen, and M. Lin, "Generating high-quality discrete LOD meshes for 3D computer games in linear time," *Multimedia Syst.*, vol. 11, pp. 480–494, May 2006.

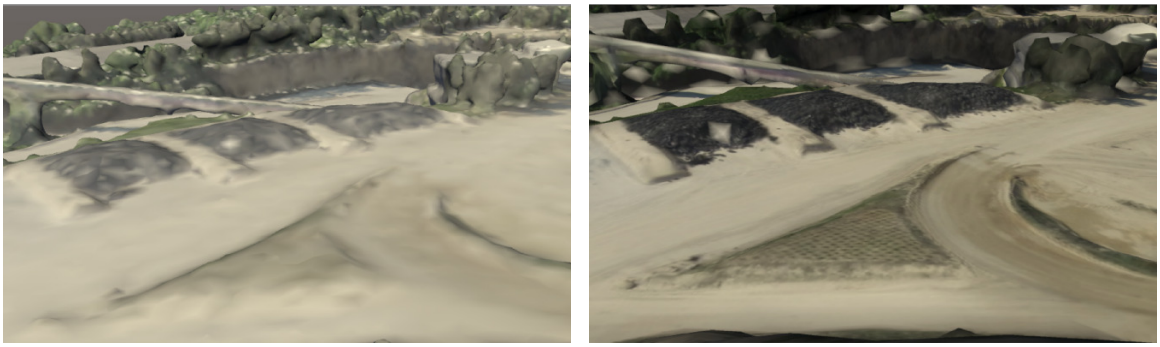
[83] I. O. Yaz and S. Loriot, "Triangulated surface mesh segmentation," in *CGAL User and Reference Manual*, 4.14 ed. CGAL Editorial Board, 2019.

ADDENDUM A COMPARISON OF TEXTURED AND VERTEX-COLOURED MESHES

This section provides a continuation of the comparison of textured and vertex-coloured meshes that was provided in Section 5.3.1.

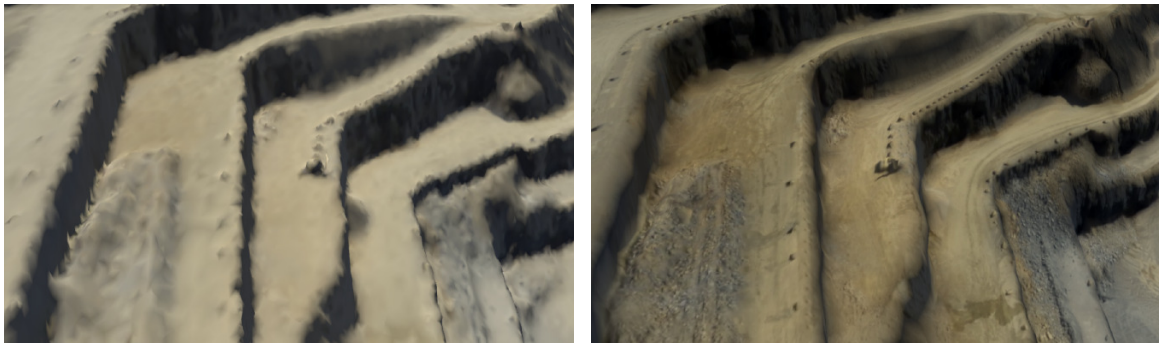
(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.1. The 4^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Swiss Quarry dataset [79].



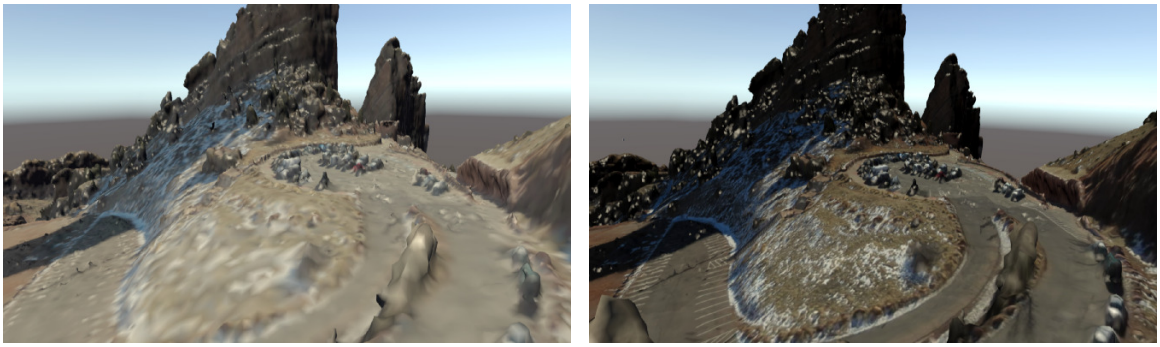
(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.2. The 5^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Swiss Quarry dataset [79].



(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.3. The 6^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Swiss Quarry dataset [79].



(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.4. The 7^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Red Rocks dataset [75].

(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.5. The 8^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Red Rocks dataset [75].

(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.6. The 9^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Reservoir dataset [75].

(a) A vertex-coloured mesh **(b)** A textured mesh section

Figure A.7. The 10^{th} comparison of a vertex-coloured mesh with a mesh that was textured using pix4Dmapper [35]. The meshes were generated using images from the Reservoir dataset [75].

ADDENDUM B QUANTITATIVE SEGMENTATION RESULTS OF THE PRINCETON DATASET

The following sections provide the detailed quantitative results of the segmentation experiments carried out in Section 5.4.2. Tables B.1, B.2, B.3, B.4, B.5, B.6 and B.7 give the results of segmenting each of the CAD model classes contained in the Princeton dataset, using the CD, HD, RI and CE metrics. The names of the algorithms are keyed in as follows; Benchmark (Bench), Random Cuts (RC), Shape Diameter (SD), Normalised Cuts (NC), Core Extraction (COE), Random Walks (RW), Fitting Primitives (FP), K-Means (KM) and Triangle Pool Algorithm (TPA).

Table B.1. Average CD values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.293	0.220	0.298	0.231	0.420	0.299	0.220	0.280	0.334
Cup	0.428	0.879	0.970	0.658	1,017	1,179	0.982	1,164	1,052
Glasses	0.231	0.263	0.408	0.277	0.535	0.608	0.237	0.408	0.404
Airplane	0.097	0.147	0.114	0.256	0.371	0.335	0.311	0.299	0.377
Ant	0.087	0.055	0.075	0.130	0.151	0.186	0.249	0.296	0.080
Chair	0.143	0.252	0.187	0.190	0.342	0.275	0.252	0.314	0.266
Octopus	0.067	0.102	0.119	0.152	0.141	0.175	0.175	0.228	0.113
Table	0.135	0.390	0.209	0.120	0.421	0.155	0.166	0.458	0.395
Teddy	0.057	0.052	0.062	0.131	0.202	0.115	0.123	0.207	0.203
Hand	0.176	0.138	0.312	0.225	0.262	0.278	0.258	0.247	0.285
Plier	0.091	0.176	0.538	0.309	0.162	0.380	0.472	0.449	0.551
Fish	0.120	0.268	0.169	0.244	0.275	0.257	0.270	0.258	0.252
Bird	0.108	0.192	0.187	0.342	0.216	0.446	0.414	0.354	0.474
Armadillo	0.142	0.122	0.132	0.161	0.256	0.170	0.133	0.184	0.177
Bust	0.286	0.201	0.317	0.327	0.485	0.334	0.300	0.373	0.401
Mech	0.186	0.425	0.322	0.218	0.773	0.256	0.442	0.595	0.664
Bearing	0.218	0.212	0.329	0.291	0.743	0.414	0.446	0.411	0.430
Vase	0.219	0.207	0.292	0.311	0.400	0.363	0.354	0.454	0.287
FourLeg	0.190	0.185	0.182	0.246	0.274	0.275	0.191	0.221	0.219
Average	0.172	0.236	0.275	0.254	0.375	0.342	0.316	0.379	0.367

Table B.2. Average HD values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.202	0.197	0.292	0.224	0.267	0.252	0.280	0.301	0.279
Cup	0.073	0.098	0.187	0.132	0.157	0.166	0.210	0.261	0.300
Glasses	0.100	0.101	0.217	0.111	0.221	0.214	0.294	0.165	0.295
Airplane	0.108	0.161	0.127	0.264	0.268	0.298	0.280	0.331	0.201
Ant	0.067	0.051	0.050	0.110	0.111	0.141	0.199	0.264	0.077
Chair	0.074	0.160	0.106	0.088	0.128	0.138	0.274	0.255	0.231
Octopus	0.040	0.058	0.057	0.078	0.066	0.084	0.157	0.147	0.059
Table	0.058	0.218	0.111	0.075	0.122	0.089	0.125	0.253	0.254
Teddy	0.046	0.053	0.057	0.151	0.121	0.149	0.159	0.282	0.148
Hand	0.114	0.109	0.256	0.175	0.172	0.196	0.282	0.228	0.237
Plier	0.083	0.144	0.313	0.229	0.122	0.230	0.241	0.308	0.316
Fish	0.100	0.196	0.163	0.327	0.170	0.309	0.383	0.362	0.163
Bird	0.074	0.117	0.154	0.221	0.141	0.241	0.313	0.280	0.225
Armadillo	0.167	0.171	0.215	0.242	0.235	0.276	0.222	0.315	0.218
Bust	0.153	0.173	0.267	0.300	0.202	0.276	0.296	0.359	0.262
Mech	0.078	0.131	0.144	0.109	0.181	0.117	0.152	0.289	0.309
Bearing	0.072	0.100	0.085	0.125	0.213	0.182	0.132	0.203	0.142
Vase	0.081	0.075	0.155	0.184	0.140	0.174	0.187	0.324	0.124
FourLeg	0.190	0.235	0.209	0.257	0.221	0.288	0.318	0.327	0.219
Average	0.099	0.134	0.166	0.179	0.169	0.201	0.237	0.277	0.214

Table B.3. Average HD-Rm values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.202	0.229	0.284	0.246	0.145	0.246	0.342	0.354	0.220
Cup	0.073	0.121	0.306	0.191	0.214	0.243	0.331	0.411	0.557
Glasses	0.100	0.122	0.217	0.098	0.128	0.137	0.311	0.180	0.410
Airplane	0.108	0.185	0.138	0.305	0.190	0.310	0.323	0.396	0.272
Ant	0.067	0.057	0.034	0.126	0.052	0.179	0.252	0.316	0.059
Chair	0.074	0.197	0.142	0.091	0.076	0.174	0.371	0.357	0.345
Octopus	0.040	0.059	0.058	0.086	0.060	0.095	0.196	0.166	0.063
Table	0.058	0.321	0.162	0.096	0.140	0.127	0.189	0.359	0.456
Teddy	0.046	0.064	0.049	0.185	0.094	0.217	0.236	0.372	0.216
Hand	0.114	0.114	0.277	0.192	0.163	0.207	0.331	0.269	0.239
Plier	0.083	0.163	0.200	0.244	0.133	0.175	0.255	0.332	0.220
Fish	0.100	0.259	0.220	0.504	0.173	0.427	0.562	0.561	0.172
Bird	0.074	0.136	0.197	0.240	0.126	0.216	0.383	0.332	0.319
Armadillo	0.167	0.183	0.217	0.229	0.052	0.274	0.255	0.342	0.132
Bust	0.153	0.222	0.336	0.417	0.133	0.376	0.423	0.508	0.276
Mech	0.078	0.144	0.200	0.139	0.155	0.156	0.212	0.387	0.563
Bearing	0.072	0.126	0.080	0.182	0.113	0.206	0.132	0.294	0.108
Vase	0.081	0.078	0.228	0.239	0.099	0.203	0.256	0.452	0.173
FourLeg	0.190	0.273	0.201	0.275	0.159	0.309	0.403	0.418	0.185
Average	0.099	0.161	0.187	0.215	0.126	0.225	0.303	0.358	0.262

Table B.4. Average HD-Rf values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.202	0.165	0.299	0.203	0.388	0.259	0.217	0.248	0.337
Cup	0.073	0.075	0.067	0.073	0.101	0.089	0.089	0.111	0.043
Glasses	0.100	0.081	0.216	0.125	0.315	0.290	0.277	0.151	0.180
Airplane	0.108	0.138	0.115	0.224	0.346	0.286	0.237	0.266	0.130
Ant	0.067	0.045	0.066	0.094	0.170	0.102	0.146	0.213	0.094
Chair	0.074	0.123	0.069	0.085	0.180	0.102	0.176	0.152	0.117
Octopus	0.040	0.058	0.055	0.071	0.071	0.072	0.118	0.128	0.054
Table	0.058	0.114	0.060	0.055	0.105	0.050	0.061	0.147	0.051
Teddy	0.046	0.042	0.065	0.118	0.148	0.081	0.081	0.191	0.080
Hand	0.114	0.105	0.234	0.157	0.180	0.186	0.233	0.186	0.235
Plier	0.083	0.125	0.426	0.214	0.112	0.286	0.227	0.284	0.412
Fish	0.100	0.134	0.106	0.149	0.166	0.192	0.203	0.162	0.155
Bird	0.074	0.098	0.112	0.202	0.156	0.266	0.243	0.228	0.130
Armadillo	0.167	0.159	0.212	0.255	0.418	0.278	0.189	0.288	0.303
Bust	0.153	0.123	0.197	0.182	0.271	0.177	0.169	0.210	0.247
Mech	0.078	0.117	0.088	0.079	0.207	0.078	0.093	0.192	0.056
Bearing	0.072	0.074	0.091	0.069	0.312	0.159	0.131	0.112	0.175
Vase	0.081	0.073	0.082	0.129	0.180	0.146	0.117	0.197	0.075
FourLeg	0.190	0.197	0.217	0.239	0.283	0.268	0.233	0.236	0.254
Average	0.099	0.108	0.146	0.143	0.213	0.177	0.171	0.195	0.165

Table B.5. Average $1 - RI$ values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.135	0.111	0.179	0.121	0.225	0.154	0.130	0.140	0.185
Cup	0.136	0.198	0.358	0.220	0.307	0.327	0.388	0.442	0.518
Glasses	0.101	0.102	0.204	0.107	0.301	0.263	0.209	0.139	0.216
Airplane	0.092	0.116	0.092	0.180	0.256	0.231	0.163	0.207	0.146
Ant	0.030	0.024	0.022	0.054	0.065	0.071	0.090	0.124	0.035
Chair	0.089	0.181	0.111	0.082	0.187	0.128	0.205	0.203	0.211
Octopus	0.024	0.060	0.045	0.066	0.051	0.069	0.102	0.106	0.045
Table	0.093	0.381	0.184	0.094	0.244	0.130	0.176	0.373	0.384
Teddy	0.049	0.048	0.057	0.122	0.114	0.128	0.133	0.183	0.093
Hand	0.091	0.093	0.202	0.147	0.155	0.179	0.195	0.161	0.194
Plier	0.071	0.112	0.375	0.182	0.093	0.225	0.170	0.247	0.386
Fish	0.155	0.298	0.248	0.391	0.273	0.374	0.421	0.409	0.264
Bird	0.062	0.103	0.115	0.180	0.124	0.238	0.195	0.188	0.161
Armadillo	0.083	0.096	0.090	0.127	0.141	0.116	0.086	0.117	0.122
Bust	0.220	0.202	0.298	0.300	0.315	0.285	0.278	0.319	0.327
Mech	0.131	0.233	0.238	0.144	0.387	0.183	0.266	0.401	0.497
Bearing	0.104	0.109	0.119	0.153	0.398	0.222	0.166	0.244	0.210
Vase	0.144	0.126	0.239	0.235	0.226	0.249	0.250	0.376	0.163
FourLeg	0.149	0.166	0.161	0.190	0.191	0.215	0.180	0.181	0.190
Average	0.103	0.145	0.176	0.163	0.211	0.199	0.200	0.240	0.229

Table B.6. Average GCE values of each algorithm per model.

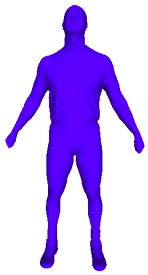
Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.126	0.219	0.258	0.244	0.174	0.274	0.298	0.308	0.235
Cup	0.025	0.083	0.066	0.115	0.109	0.138	0.159	0.199	0.064
Glasses	0.033	0.121	0.120	0.125	0.131	0.180	0.309	0.204	0.189
Airplane	0.090	0.182	0.117	0.283	0.215	0.325	0.303	0.343	0.172
Ant	0.035	0.068	0.036	0.119	0.075	0.136	0.192	0.276	0.070
Chair	0.051	0.181	0.075	0.112	0.101	0.130	0.237	0.219	0.155
Octopus	0.029	0.068	0.046	0.100	0.076	0.104	0.153	0.186	0.058
Table	0.021	0.176	0.049	0.075	0.082	0.065	0.081	0.209	0.068
Teddy	0.048	0.072	0.068	0.186	0.144	0.133	0.140	0.274	0.114
Hand	0.094	0.136	0.229	0.199	0.177	0.211	0.298	0.255	0.263
Plier	0.077	0.174	0.237	0.258	0.136	0.219	0.311	0.321	0.294
Fish	0.079	0.188	0.137	0.229	0.193	0.284	0.306	0.245	0.197
Bird	0.077	0.142	0.135	0.247	0.157	0.237	0.332	0.301	0.188
Armadillo	0.112	0.204	0.201	0.266	0.087	0.313	0.261	0.353	0.157
Bust	0.099	0.169	0.230	0.239	0.168	0.228	0.238	0.280	0.261
Mech	0.021	0.135	0.068	0.105	0.097	0.106	0.135	0.254	0.077
Bearing	0.022	0.075	0.054	0.082	0.096	0.154	0.100	0.160	0.087
Vase	0.056	0.091	0.116	0.181	0.124	0.185	0.178	0.286	0.103
FourLeg	0.177	0.255	0.221	0.279	0.214	0.307	0.323	0.324	0.223
Average	0.067	0.144	0.130	0.181	0.135	0.196	0.229	0.263	0.157

Table B.7. Average LCE values of each algorithm per model.

Model	Bench	RC	SD	NC	COE	RW	FP	KM	TPA
Human	0.085	0.122	0.171	0.151	0.129	0.154	0.183	0.207	0.155
Cup	0.018	0.040	0.045	0.066	0.063	0.068	0.078	0.113	0.052
Glasses	0.023	0.064	0.069	0.062	0.075	0.078	0.248	0.128	0.158
Airplane	0.057	0.098	0.073	0.170	0.132	0.202	0.216	0.242	0.103
Ant	0.025	0.040	0.024	0.068	0.053	0.055	0.088	0.179	0.045
Chair	0.028	0.083	0.043	0.061	0.059	0.058	0.117	0.145	0.093
Octopus	0.020	0.034	0.026	0.056	0.048	0.056	0.075	0.110	0.031
Table	0.014	0.087	0.029	0.057	0.051	0.039	0.046	0.119	0.041
Teddy	0.034	0.046	0.051	0.124	0.072	0.054	0.068	0.182	0.073
Hand	0.060	0.079	0.155	0.128	0.122	0.118	0.153	0.163	0.194
Plier	0.056	0.093	0.122	0.148	0.088	0.119	0.231	0.191	0.135
Fish	0.053	0.102	0.090	0.138	0.108	0.194	0.220	0.149	0.122
Bird	0.051	0.072	0.094	0.149	0.097	0.138	0.254	0.205	0.134
Armadillo	0.075	0.121	0.123	0.176	0.072	0.181	0.170	0.244	0.102
Bust	0.065	0.101	0.167	0.181	0.110	0.158	0.179	0.221	0.154
Mech	0.012	0.074	0.048	0.075	0.054	0.050	0.064	0.164	0.060
Bearing	0.015	0.043	0.032	0.050	0.078	0.085	0.054	0.096	0.027
Vase	0.037	0.051	0.068	0.114	0.084	0.108	0.104	0.192	0.059
FourLeg	0.116	0.146	0.135	0.176	0.141	0.198	0.226	0.235	0.130
Average	0.044	0.079	0.082	0.113	0.086	0.111	0.146	0.173	0.098

ADDENDUM C QUALITATIVE SEGMENTATION RESULTS OF THE PRINCETON DATASET

The following sections provide the extended results of the segmentation experiments carried out in Section 5.4.2.5. Figures C.1, C.2, C.3, C.4, C.5, C.6, C.7, C.8 and C.9 show the qualitative segmentation results obtained from segmenting CAD models contained in the Princeton dataset. A region with the same colour represents a segment. The analysis was carried out for the TPA, Shape Diameter and Fitting Primitives algorithms.



(a) Original

(b) TPA segmentation

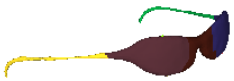
(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

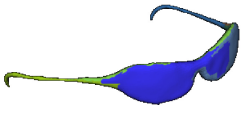
Figure C.1. Qualitative segmentation results of a CAD model from the Human category in the Princeton dataset [65].

(a) Original

(b) TPA segmentation

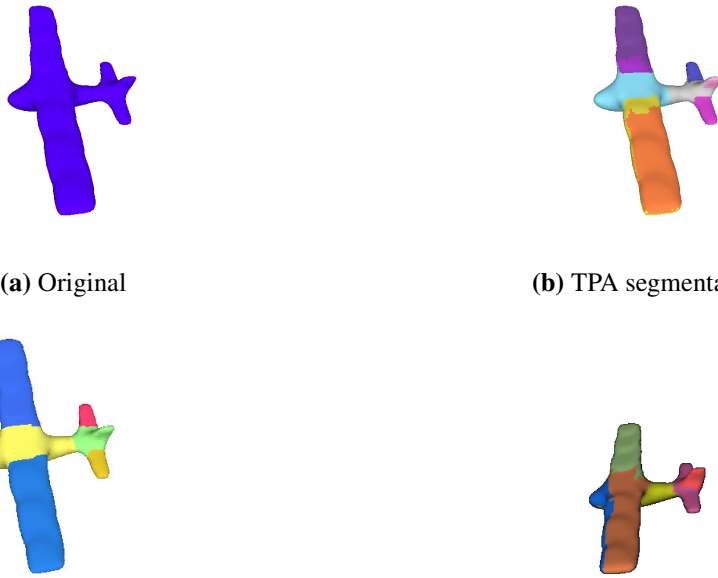


(c) Shape Diameter segmentation



(d) Fitting Primitives segmentation

Figure C.2. Qualitative segmentation results of a CAD model from the Glasses category in the Princeton dataset [65].



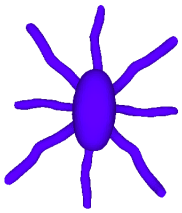
(a) Original

(b) TPA segmentation

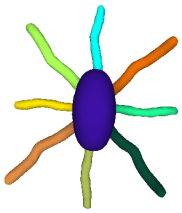
(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

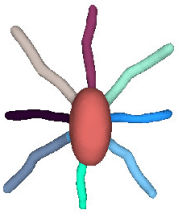
Figure C.3. Qualitative segmentation results of a CAD model from the Aeroplane category in the Princeton dataset [65].



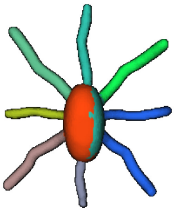
(a) Original



(b) TPA segmentation



(c) Shape Diameter segmentation



(d) Fitting Primitives segmentation

Figure C.4. Qualitative segmentation results of a CAD model from the Octopus category in the Princeton dataset [65].

(a) Original

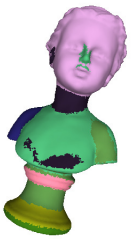
(b) TPA segmentation

(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

Figure C.5. Qualitative segmentation results of a CAD model from the Armadillo category in the Princeton dataset [65].

(a) Original



(b) TPA segmentation

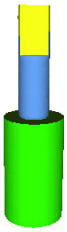
(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

Figure C.6. Qualitative segmentation results of a CAD model from the Bust category in the Princeton dataset [65].

(a) Original

(b) TPA segmentation

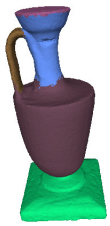


(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

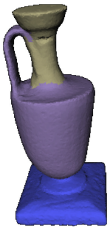
Figure C.7. Qualitative segmentation results of a CAD model from the Bearing category in the Princeton dataset [65].

(a) Original



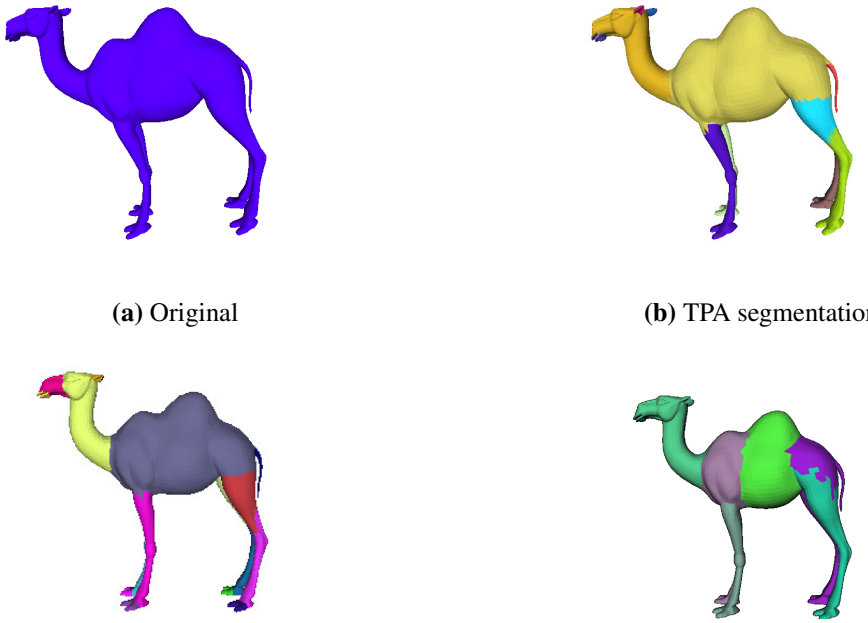
(b) TPA segmentation

(c) Shape Diameter segmentation



(d) Fitting Primitives segmentation

Figure C.8. Qualitative segmentation results of a CAD model from the Vase category in the Princeton dataset [65].



(a) Original

(b) TPA segmentation

(c) Shape Diameter segmentation

(d) Fitting Primitives segmentation

Figure C.9. Qualitative segmentation results of a CAD model from the Four-leg category in the Princeton dataset [65].