# RECONSTRUCTION AND TEXTURING OF 3D SURFACES FROM FUSED LOW-COST AERIAL LIDAR AND OPTICAL IMAGERY

by

Samuel Kiguthi

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

| | |
|---|---|
| Scott E. Budge, Ph.D.<br>Major Professor | Jonathan Phillips, Ph.D.<br>Committee Member |
| Jacob Gunther, Ph.D.<br>Committee Member | David F. Feldon, Ph.D.<br>Vice Provost of Graduate Studies |

UTAH STATE UNIVERSITY
Logan, Utah

2025

# ABSTRACT

Reconstruction and Texturing of 3D Surfaces from Fused Low-Cost Aerial LiDAR and Optical Imagery

by

Samuel Kiguthi, Master of Science

Utah State University, 2025

Major Professor: Scott E. Budge, Ph.D.
Department: Electrical and Computer Engineering

Using aerial LiDAR point clouds and optical imagery, it is possible to create a Fused LiDAR image (Texel image). Multiple Texel images can be combined into a larger 3D terrain representation called a Textured Digital Surface Model (TDSM). TDSMs have broad applications in surveying, agriculture, and infrastructure development. Despite their utility, traditional surface reconstruction methods frequently misrepresent concave features, such as caves, overhangs, doorways, or outcroppings, particularly when data is collected aerially from drones. Drone-based LiDAR scans densely sample top surfaces but produce sparse points on vertical or inclined surfaces, severely affecting the quality and completeness of the resulting TDSM.

This work presents a surface-reconstruction pipeline for aerial LiDAR data that leverages the Ball-Pivoting Algorithm (BPA) and camera poses recorded during data acquisition. The presented method specifically addresses previous assumptions that are invalid for drone-collected aerial data, such as uniform point density and complete coverage. The improvements include clustering LiDAR point clouds, BPA radius estimation to handle local variations in point density, and camera-view stitching. Real-world experiments using synchronized LiDAR, optical imagery, and inertial navigation demonstrate the proposed

surface reconstruction method more effectively preserves concave features, mitigates data sparsity issues from aerial collection, and substantially improves the accuracy and fidelity of the final TDSMs.

(82 pages)

# PUBLIC ABSTRACT

Reconstruction and Texturing of 3D Surfaces from Fused Low-Cost Aerial LiDAR and Optical Imagery

Samuel Kiguthi

Drones equipped with laser scanners (LiDAR) and cameras capture detailed 3D scenes. Combining the laser points with photos builds realistic, photo-textured 3D maps used in surveying, agriculture, and infrastructure planning. Conventional methods to create these maps often misrepresent inward shapes such as doorways, overhangs, and outcrops. These shapes are misrepresented because drones mainly view top surfaces and do not collect significant data on vertical or hidden areas.

This study introduces a surface-reconstruction method that groups LiDAR points into clusters, reconstructs smooth surfaces for each cluster, and uses information from recorded camera poses to stitch clusters together. Tests on real drone flights with synchronized LiDAR, imagery, and navigation data show that the method better preserves concave features, mitigates errors from sparse sampling, and improves the completeness and accuracy of the resulting 3D maps.

# ACKNOWLEDGMENTS

I thank Dr. Scott Budge for steady guidance and many thoughtful conversations that shaped this work. Thanks also to Dr. Calvin Coopmans and the Aggie Air team for their help with data collection and flight support.

Samuel Kiguthi

# CONTENTS

# LIST OF FIGURES

# ACRONYMS

| | |
|---|---|
| 2D | 2-dimensional |
| 3D | 3-dimensional |
| BA | Bundle Adjustment |
| BPA | Ball-Pivoting Algorithm |
| CAIL | Center for Advanced Imaging Ladar |
| CDT | Constrained Delaunay Triangulation |
| DCM | Direction Cosine Matrix |
| DBSCAN | Density-Based Spatial Clustering |
| DSM | Digital Surface Map |
| EO | Electro-optical |
| GPS | Global Positioning System |
| IMU | Inertial Measurement Unit |
| INS | Inertial Navigation System |
| LLA | Latitude, Longitude, Altitude |
| LMA | Levenberg-Marquardt Algorithm |
| LiDAR | Light Detection and Ranging |
| NCC | Normalized Cross Correlation |
| NED | North-East-Down |
| PSR | Poisson Surface Reconstruction |
| RGB-D | Red-Green-Blue-Depth |
| SBA | Streaming Bundle Adjustment |
| sUAV | Small Unmanned Aerial Vehicle |
| TDSM | Textured Digital Surface Map |

# CHAPTER 1

# INTRODUCTION

The modeling and reconstruction of objects or scenes is a core challenge in computer vision and graphics research. Classic applications of reconstruction include virtual and augmented reality, computer animation, computer-aided design, and robotics [1]. Given data in 3D Euclidean space, researchers typically reconstruct only the visible surfaces of objects or scenes, rather than the full solid objects themselves. This type of reconstruction represents the surface geometry of the objects, which can be modeled as a two-dimensional (2D) manifold embedded within three-dimensional (3D) Euclidean space.

This work focuses on reconstructing outdoor scenes from real-world environments, which are often used in autonomous vehicles, virtual tourism, and environmental surveys. Reconstructing an outdoor scene with both 3D and visual elements opens up new possibilities in various fields. For example, autonomous vehicles can use a combination of visual and depth information to make informed decisions. Similarly, people can explore places they have never visited or gain insight into how environments change over time by reconstructing the same scene at multiple points in time.

Data from outdoor scenes can be acquired using various types of scanners. These include Light Detection and Ranging (LiDAR) systems, which capture 3D geometry; multi-view electro-optical (EO) cameras [2,3], which capture visual imagery and infer 3D geometry; and Red-Green-Blue-Depth (RGB-D) systems [4], which record both color information and depth. Most modern scanners generate point clouds [5], which are a discrete approximation of a 2D manifold. A point cloud on its own is valuable; however, a continuous surface, often referred to as a Digital Surface Model (DSM), is frequently more desirable for applications in agriculture [6], ecology [7], geography [8], hydrology [9], terrain analysis [10] , and virtual tourism [11]. This work focuses on creating a DSM by reconstructing a continuous surface from a point cloud generated through the fusion of LiDAR and camera

data. After generating the surface, imagery is overlaid on top of it to create a Textured Digital Surface Model (TDSM).

Although a TDSM is useful, outdoor data is traditionally acquired using full-scale aircraft equipped with a Global Positioning System (GPS) and high-cost Inertial Measurement Units (IMUs). These aircraft often carry multiple high-resolution EO cameras and, in some cases, high-end LiDAR systems. This combination of equipment produces fairly uniform surface coverage of the subject scene and a *high-quality* resulting TDSM; however, it can be an expensive undertaking.

A Small Unmanned Aerial Vehicle (sUAV) is a natural low-cost alternative to full-scale aircraft, due to its wide range of applications [12]. Additionally, low-cost IMUs, cameras, and LiDAR systems are now widely accessible. However, these come with trade-offs: position and attitude estimates are more prone to error, LiDAR data is more susceptible to dropouts, and the overall sensing range is reduced.

This study investigates a 3D surface reconstruction method meant for low-quality data acquired from low-cost aerial platforms. In particular, this work addresses the challenges associated with reconstructing surfaces from aerial LiDAR data. Aerial LiDAR data is often characterized by large vertical discontinuities and non-uniform point density. Existing reconstruction methods don't accurately represent certain 3D features in this type of data, especially concave structures such as overhangs. This research proposes a reconstruction pipeline that leverages the Ball-Pivoting Algorithm (BPA) [13], as well as known camera positions from the data acquisition process.

## 1.1 Motivation and Context

To motivate the research presented in this thesis, this section reviews previous work on reconstructing outdoor scenes into TDSMs.

### 1.1.1 Data Acquisition

There are many methods of gathering data from outdoor scenes. One widely used approach is photogrammetry, which reconstructs 3D information using images captured from

multiple cameras or viewpoints [14]. A single image represents a projection of the 3D world onto a 2D plane, viewed from the camera's perspective. By capturing images from different angles, it becomes possible to infer depth and reconstruct a 3D scene. Google Earth [2, 15] and Apple Flyover [3] are notable examples of photogrammetry used to reconstruct outdoor scenes for virtual tourism. The high-quality results of these efforts are undeniable; however, the data acquired already provides an accurate model of the scene even before reconstruction. Whether captured from full-scale aircraft or ground vehicles, Google Earth and Apple Flyover gather complete imagery from all angles—a luxury not available when using an sUAV with a traditional survey flight pattern.

Photogrammetry remains widely used, even in autonomous driving systems. For example, Wang et al. [16] showed that accurate object detection, distance estimation, and motion prediction can be achieved using only multi-camera input, without the need for active depth sensors. Wang's work assumes that objects can be reliably detected; however, the scenes addressed in the present work are not restricted to those where objects can be reliably detected.

Although photogrammetry is a well-established technique, an alternative approach involves fusing LiDAR data with optical imagery. LiDAR provides precise 3D point measurements within the accuracy limits of the instrument, offering advantages in conditions where cameras may struggle—such as in fog, darkness, or glare. As a result, combining LiDAR with cameras is often seen as a more robust solution. A notable example is in autonomous vehicle applications, where Kocić et al. [17] demonstrated that leveraging both image and 3D point cloud data led to reliable moving object detection and environmental mapping in dynamic scenes even in poor visibility conditions.

The use of LiDAR–camera fusion is increasingly common in applications such as robotics [18], autonomous vehicles [19], and mapping [20]. However, these applications primarily fuse LiDAR point clouds with imagery to improve depth estimation or object detection. This work will use LiDAR-camera fusion to create a TDSM.

### 1.1.2 Surface Reconstruction

Whether data is gathered using photogrammetry, LiDAR, or LiDAR–camera fusion, the resulting 3D representation is typically a point cloud. However, such data is often affected by a variety of issues, including point-wise noise, non-uniform point distribution, outliers, misalignment, and missing points..

*Point-wise noise* can arise from sensor limitations, ambient interference, surface reflectivity, or other environmental factors. Inevitably, measured points will deviate from the true surface.

*Non-uniform point distributions* may occur when certain regions are scanned multiple times, when the scanner's velocity varies during acquisition, or when differences in surface reflectance lead to areas of higher or lower sampling density.

*Misalignment* often results from combining scans taken from different viewpoints. If the extrinsic parameters of the camera or LiDAR (position and attitude) are inaccurately estimated, registration errors can occur when merging point sets.

These properties of a measured point cloud have led to significant research into *Surface Reconstruction* - the process of creating a continuous surface from the discrete point cloud. Huang et al. [1] introduce several robust methods to mitigate noise, non-uniformity, outliers, and misalignment; however, the challenge of handling missing points—gaps in the data where no measurements are available—remains far less effectively addressed. Because this work uses data collected from an aerial perspective in a regular survey pattern, the resulting point clouds often contain significant gaps.

## 1.2 Contribution

This thesis presents a surface reconstruction method for point clouds that does not assume uniform sampling and overcomes missing points. The approach treats the input point cloud as ground truth, avoiding approximation-based reconstruction methods. Texture is subsequently overlaid onto the reconstructed surface to produce a TDSM. The pipeline overcomes dropouts and large vertical discontinuities by clustering the original point cloud, triangulating each cluster using BPA, and stitching clusters together based on knowledge of camera positions in the collected data.

The proposed method generates a smoother mesh than in previous work and accurately selects textures corresponding to each triangle, even when certain triangles are occluded from specific viewpoints. This enables the creation of TDSMs using low-cost sUAV systems with traditional flight patterns, while preserving the concave features present in the data.

The remainder of this thesis is organized as follows: Chapter 2 introduces the Texel camera and reviews previous surface reconstruction methods. Chapter 3 details the proposed reconstruction pipeline. Chapter 4 compares the proposed and previous methods on multiple datasets. Chapter 5 presents the conclusions and outlines potential directions for future work. Finally, Appendix 6 contains supplementary material used in this work.

# CHAPTER 2

# BACKGROUND

Creating a Textured Digital Surface Map (TDSM) requires acquiring or simulating both visual imagery and 3D measurements. If multiple frames are captured from different locations, they must be registered into a common reference frame. While this registered dataset is already useful, producing a TDSM involves additional processing, including surface reconstruction and texture mapping.

A critical step in creating a TDSM is reconstructing the 3D scene from the registered point cloud. For aerial LiDAR data, point density is typically high on horizontal surfaces—such as rooftops, tree canopies, and the ground—but much sparser on vertical surfaces like building walls or tree trunks. Most existing surface reconstruction methods are not designed to address this uneven sampling, often resulting in incomplete or inaccurate reconstructions.

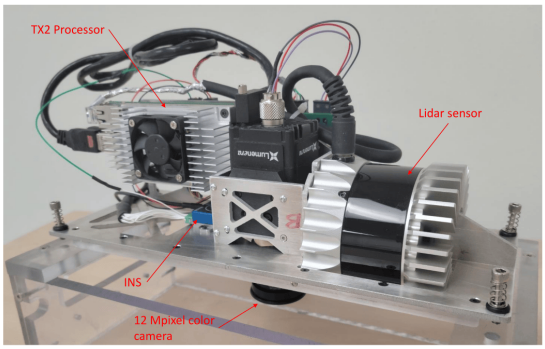
The research presented in this thesis builds on prior work conducted at the Center for Advanced Imaging Ladar (CAIL), which developed methods for LiDAR–camera fusion and the creation of TDSMs [21–24]. This chapter will first provide background on previous work at CAIL, then provide background on different methods of Surface Reconstruction.

## 2.1 Texel Camera

Prior work introduced the *Texel Camera*, where the term *texel* refers to the combination of imagery and 3D information, analogous to the use of *voxel* in computer graphics [25, 26]. The Texel Camera is a payload designed for use on a small Unmanned Aerial Vehicle (sUAV). The Texel Camera's sensors are comprised of an Electro-Optical (EO) camera, a rotational LiDAR unit, and an Inertial Navigation System (INS). In this context, an INS refers to an Inertial Measurement Unit (IMU) supplemented with a Global Positioning System (GPS) for positioning, and other sensors such as a barometer to improve position and attitude

estimates.

Figure 2.1a shows the complete Texel camera system, while Figure 2.1b shows the Texel camera mounted on an example sUAV.



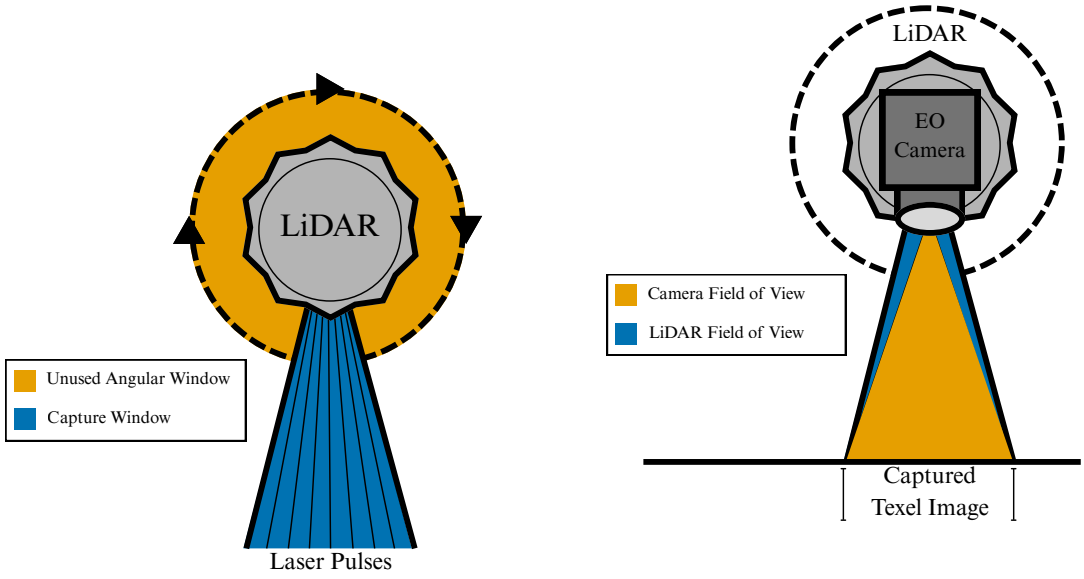

(a) Texel Camera

(b) Texel Camera Mounted on Drone

Fig. 2.1: The Texel camera system (left) and the camera mounted on sUAV (right).

The LiDAR system in the Texel Camera is capable of rotating 360 degrees. From an aerial perspective, full 360-degree coverage is not useful, so data is instead collected over a narrow angular window that aligns with the camera's field of view, as shown in Figure 2.2. In this figure, the LiDAR is viewed perpendicular to the axis of rotation. When the LiDAR's rotation reaches the center of this window, the camera receives a hardware trigger to capture a frame, and the INS records position and attitude measurements. This provides the synchronized data necessary to create a single Texel Image.

As seen in Figure 2.1a, the Texel Camera is constructed such that the EO camera and the LiDAR unit are bi-static, meaning the centers of projection of each sensor are at different locations, which introduces parallax. The sensors are aligned such that the principal rays are parallel; therefore, transforming the location of the LiDAR and its measured 3D points into the camera's frame of reference is simply a shift in the in-track direction. The calibration




(a) Rotational LiDAR

(b) Rotational LiDAR and EO Camera Acquisition Window

Fig. 2.2: LiDAR-Camera Diagram

process described in [22] aligns each LiDAR shot with a pixel in the camera.

The INS on the Texel Camera is not physically aligned with the EO camera. Jensen [27] implements a motion-capture–based calibration procedure to align the INS with the EO camera. After this calibration, the LiDAR, EO camera, and INS are fully aligned and synchronized, enabling the capture of unified Texel images.

## 2.2 Data Acquisition

During data acquisition, the sUAV carrying the Texel Camera payload typically ascends to a designated altitude and then flies a survey pattern—often in the form of S-turns. An example flight path over Utah State University's campus is shown in Figure 2.3. The Texel Camera captures 10 frames per second, generally resulting in substantial overlap between consecutive frames. However, because the sUAV is at a static elevation and the camera/LiDAR system generally points downwards, most data collected is on the top of objects in the scene.

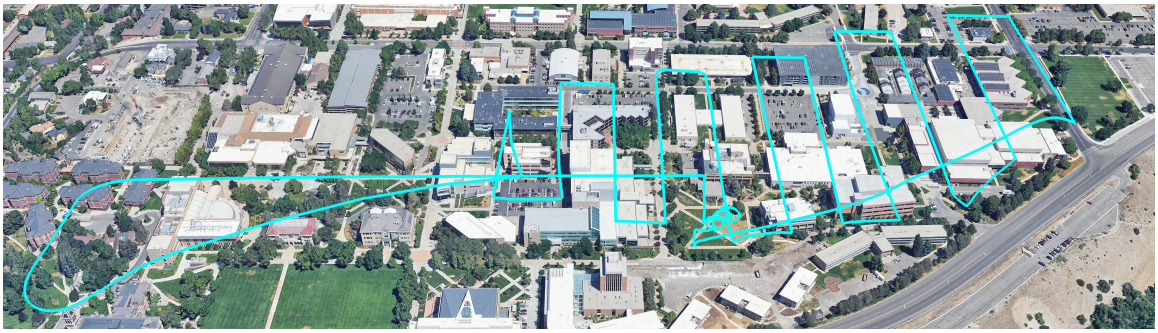


Fig. 2.3: Example Flight Pattern

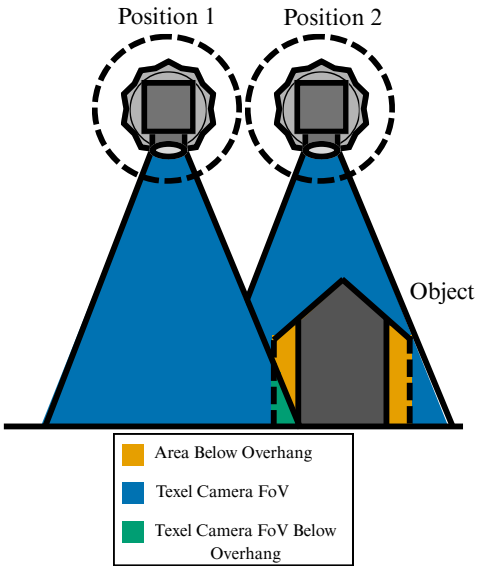
Although most of the data collected is from the tops of objects, it is possible to capture information from their sides, even in the case of concave shapes. Side information is obtained primarily due to the frustrum geometry of the Texel Camera and the slight banking of the sUAV during flight.

Figure 2.4a shows an example of a concave object, and two positions of the Texel Camera that are both looking straight down. Position 1 is able to gather some information underneath the overhang, while position 2 gathers information on top of the overhang. Figure 2.4b shows the same concave object, and a position of the Texel Camera with a slight bank, allowing it to capture significant data underneath the overhang.

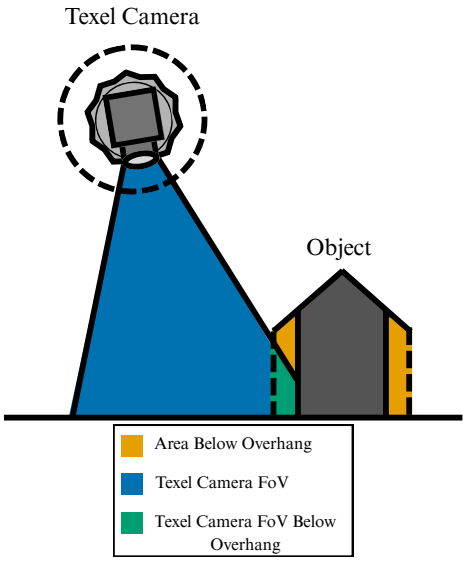
## 2.3 Transformation Between Coordinate Frames

After data acquisition, the result is a set of individual Texel images captured from different camera positions. For each image, the INS records latitude, longitude, and altitude (LLA) to describe position. However, the LLA is measured at the GPS antenna, which is not necessarily centered on the INS during flight. The acquisition software then transforms the LLA coordinates into a common (world) frame of reference defined by the first capture, assigning it a zero translation. Each subsequent frame is expressed relative to this first frame, where $x$ is the cross-track direction, $y$ is the in-track direction, and $z$ is the up direction from the first frame.

The INS also records a quaternion representing body rotation with respect to the North–East–Down (NED) frame to describe attitude. While quaternions avoid the problem




(a) Multiple Positions of Texel Camera Scanning Concave Object




(b) Banking Texel Camera Scanning Concave Object

of gimbal lock [28], in this work, rotations between frames of reference use a Direction Cosine Matrix (DCM). For the $j$-th camera, the conversion from a quaternion $q_j = (q_{j0}, q_{j1}, q_{j2}, q_{j3})$ to the DCM $R_j$ is given by:

$$[R_j] = \begin{bmatrix} 1 - \frac{2(q_{j2}^2 + q_{j3}^2)}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & \frac{2(q_{j1}q_{j2} - q_{j0}q_{j3})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & \frac{2(q_{j1}q_{j3} + q_{j0}q_{j2})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} \\ \frac{2(q_{j1}q_{j2} + q_{j0}q_{j3})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & 1 - \frac{2(q_{j1}^2 + q_{j3}^2)}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & \frac{2(q_{j2}q_{j3} - q_{j0}q_{j1})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} \\ \frac{2(q_{j1}q_{j3} - q_{j0}q_{j2})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & \frac{2(q_{j2}q_{j3} + q_{j0}q_{j1})}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} & 1 - \frac{2(q_{j2}^2 + q_{j1}^2)}{q_{j0}^2 + q_{j1}^2 + q_{j2}^2 + q_{j3}^2} \end{bmatrix}. \text{[29]}$$

Let a point $\mathbf{p} \in \mathbb{R}^3$, a DCM $R \in \mathbb{R}^{3\times3}$, and a translation vector $\mathbf{t} \in \mathbb{R}^3$ be defined as:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix}.$$

DCMs are orthogonal, meaning

$$RR^T = I \quad \Rightarrow \quad R^{-1} = R^T.$$

The pose of the $j$th camera is represented as:

$$[\, R_j \mid \mathbf{t}_j \,].$$

where $\mathbf{t}_j$ is the translation vector from the first frame to the $j$th frame in the world frame of reference and $R_j$ is the DCM for the $j$th camera.

Let the camera offset be denoted by $\mathbf{c}_{\text{off}} \in \mathbb{R}^3$ and the antenna offset by $\mathbf{a}_{\text{off}} \in \mathbb{R}^3$. These vectors represent the lever arms from the camera's center to the INS center and from the GPS antenna center to the INS center, respectively, expressed in the world frame (i.e., $\mathbf{c}_{\text{off}} = \mathbf{p}_{\text{INS}} - \mathbf{p}_{\text{cam}}$, $\mathbf{a}_{\text{off}} = \mathbf{p}_{\text{INS}} - \mathbf{p}_{\text{ant}}$).

$$\mathbf{c}_{\text{off}} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}, \quad \mathbf{a}_{\text{off}} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}.$$

Transforming a point $\mathbf{p}_j$ from the $j$th camera frame to the world frame is given by:

$$\mathbf{p}_{\text{world}} = R_j \left[\mathbf{p}_j - \mathbf{a}_{\text{off}} + \mathbf{c}_{\text{off}}\right] + \mathbf{t}_j. \tag{2.1}$$

Conversely, transforming a point $\mathbf{p}_{world}$ from the world frame to the $j$th camera frame can be expressed as:

$$\mathbf{p}_j = R_j^T \left(\mathbf{p}_{\text{world}} - \mathbf{t}_j\right) + \mathbf{a}_{\text{off}} - \mathbf{c}_{\text{off}}. \tag{2.2}$$

It is often useful to project a 3D point into the camera's normalized image plane, yielding $p_j^{2D} \in \mathbb{R}^2$, which represents what the camera actually observes. For ease of notation, let

$$\mathbf{p}_j = (p_{j,x},\, p_{j,y},\, p_{j,z})^\top \in \mathbb{R}^3,$$

Then its projection onto the normalized image plane is

$$\mathbf{p}_j^{2D} = \left( -\frac{p_{j,x}}{p_{j,z}},\ -\frac{p_{j,y}}{p_{j,z}} \right) \in \mathbb{R}^2. \tag{2.3}$$

## 2.4 Texel Image Registration / Optimization

Given a sequence of Texel images captured consecutively, CAIL has developed a process to register these images into a common world coordinate system and to optimize the points in each LiDAR point cloud using signal-processing techniques. This optimization helps correct position and attitude errors from the INS, as well as range measurement errors from the LiDAR [21–24, 30, 31].

### 2.4.1 Projection Matrix

The first step in the registration process is to form a matrix containing all 3D points and their projections into each visible image, called the Projection Matrix. During data acquisition, each 3D point is mapped to a 2D image point using its $(u, v)$ coordinates from calibration. The $(u, v)$ coordinates represent the projection of the 3D LiDAR point onto the 2D image, creating a correspondence between $(x, y, z)$ and $(u, v)$. Due to frame overlap, a 3D point acquired from position $i$ may also be visible from position $j$ and other positions.

The Projection Matrix is a table that, for each camera position, lists the 2D image projections (2.3) of all visible 3D points into that camera image. The *owner position* is the position that originally captured the 3D point, and in the Projection Matrix, any 2D point acquired from the owner position is entered directly with its measured range. For any other position $j$ that contains that 3D point, the point's 2D projection is computed through Normalized Cross Correlation (NCC), which does not provide a range measurement; in such cases, the point is entered with a range set to `NaN`.

Given a point $p_i$ observed in image $i$ that also appears in image $j$, the NCC method implemented by Khatiwada [22] determines its corresponding location in image $j$. NCC identifies corresponding points between two EO images by finding the locations with the highest normalized cross-correlation, where the correlation coefficient ranges from $-1$ to 1

and a value of 1 indicates a perfect match. NCC is commonly used in image processing; however, it can be computationally expensive when applied to many images and a large number of points per image.

To reduce computational cost, each 3D point from image $i$ is first rotated into each $j$th image space using a homography. This rotation provides an initial estimate of the 3D point's location in the $j$th image, allowing NCC to be constrained to a small search window around that estimate. The rotation could place the estimated location outside the bounds of image $j$, indicating the point is not present in that image. The homography also maps both images onto the same image plane, eliminating issues related to rotation, scale, and other geometric effects that could otherwise reduce the correlation.

The homography satisfies $p_j = H_{ij}\, p_i$, where $p_i$ is a point in image $i$, $p_j$ is the corresponding point in image $j$, and $H_{ij}$ is the transformation from image $i$ to image $j$. To estimate $H_{ij}$, matching keypoint correspondences $(p_i, p_j)$ are first detected in both images using Speeded Up Robust Features (SURF) [32].

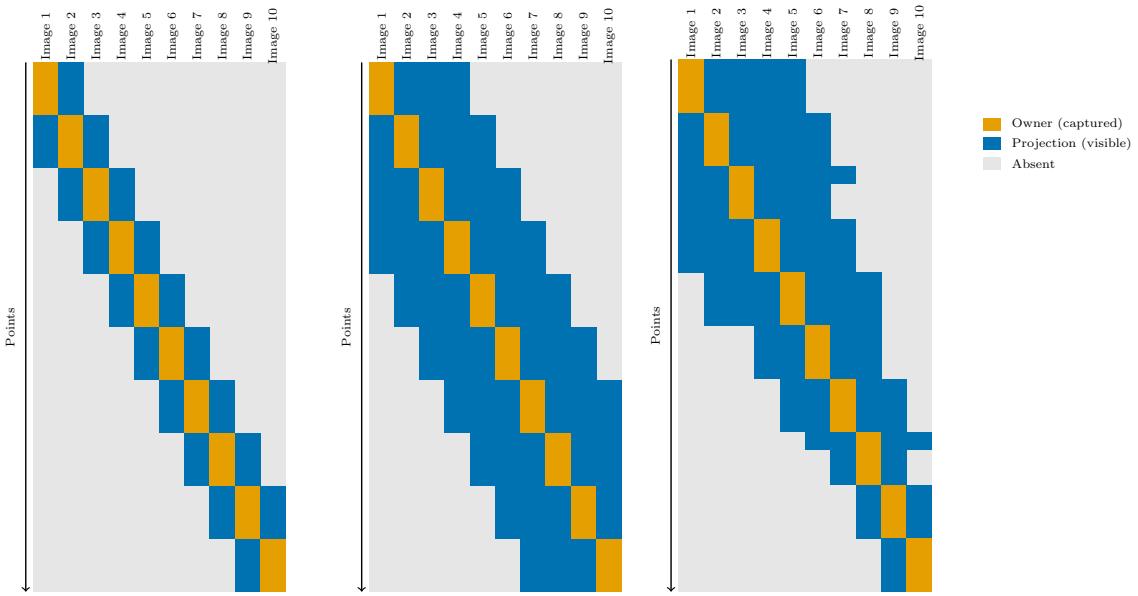
SURF consists of two stages: interest point detection and descriptor computation for the region surrounding each interest point. It is both rotation-invariant and scale-invariant, allowing it to identify corresponding features across images that differ only by rotation and scale. Since Texel images are captured in quick succession, any additional appearance changes from skew, anisotropic scaling, and perspective effects are assumed to be negligible.

After identifying correspondences using SURF, the homography $H_{ij}$ is estimated using Random Sample Consensus (RANSAC) [33] rather than a full least-squares fit. RANSAC operates by repeatedly selecting minimal subsets of four correspondences to compute candidate homographies. Each candidate homography is evaluated based on the number of inliers. An inlier is a correspondence with reprojection error that lies within a specified threshold. The candidate with the highest inlier count is selected as the homography $H_{ij}$.

Homographies are computed pairwise starting from the first frame (e.g., $H_{12}, H_{23}, \ldots$ $H_{(N-1)N}$), and are cascaded to project points into non-adjacent frames. Using these homo-

graphies, all 3D points are transformed into every image in which they are visible, completing the Projection Matrix.

The Projection Matrix is *banded*, meaning that point $p_i$ is typically projected into images with indices $j$ that are *close* to $i$. When the sUAV moves quickly during acquisition, the number of images into which point $p_i$ can be projected is reduced, as shown in Fig. 2.5a. When the sUAV moves slowly, point $p_i$ can be projected into more images, as shown in Fig. 2.5b. If the sUAV changes speed, the width of the band of projected entries for point $p_i$ varies accordingly, as shown in Fig. 2.5c.




(a) Const. Speed (Fast) (b) Const. Speed (Slow) (c) Varying Speed (Slow then Fast)

Fig. 2.5: Projection Matrix Structure

### 2.4.2 Streaming Bundle Adjustment

The original data has errors in the camera poses and the 3D points, so an optimization step is required to generate the final TDSM. A non-linear optimization method was developed [21]. This optimization problem is often described as a Bundle Adjustment (BA), which minimizes error through a cost function. Conventionally, BA is performed as a final

step to jointly optimize all camera poses and their associated points. Bybee's work applied the Levenberg–Marquardt Algorithm (LMA) to perform this optimization [31].

In Streaming Bundle Adjustment (SBA) by Khatiwada [22], rather than applying a global optimization, a local optimization is applied over a sliding window. To determine the size of this window, a parameter called the *look length*, denoted by $\mathcal{L}$, is introduced. It is assumed that this parameter corresponds to the band of projected points shown in Figure 2.5. In other words, if we are considering Texel image $i$, then all points visible in $i$ are assumed to also be visible within $\mathcal{L}$ neighboring images on either side.

This means that changing the pose or points of image $I_i$ does not affect images after $I_{i+\mathcal{L}}$ or before $I_{i-\mathcal{L}}$. The SBA window is set to $3\mathcal{L}$, where the first $\mathcal{L}$ images are the *past* images, the next $\mathcal{L}$ images are the *present* images, and the final $\mathcal{L}$ images are the *future* images, as shown in Figure 2.6. This figure shows a total of $N$ texel images; the first $M+\mathcal{L}$ are the previously optimized texel images.

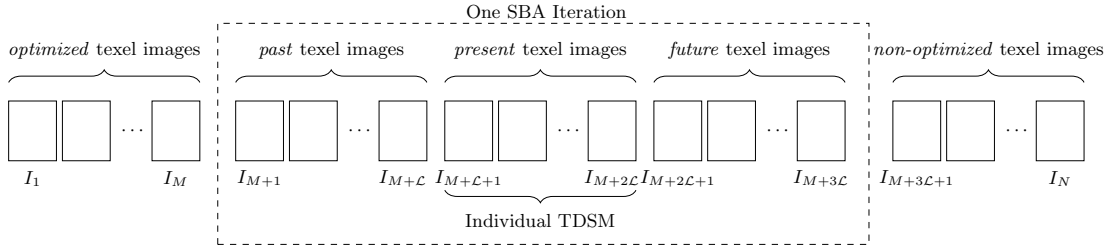



Fig. 2.6: SBA Window

In each iteration of SBA, the following measurements are optimized: (1) the image projections of all 3D points captured by the present and future texel images onto all texel images, (2) the measured ranges (where available) for the present and future texel images, and (3) the image projections of 3D points captured by past texel images onto the present texel images. At the end of one SBA window, the present texel images are fully optimized, while the future texel images are only partially optimized using the present images. This means that in the next window, the new present images (formerly future images) will

converge more quickly.

Once the present texel images are optimized, they are passed through the surface reconstruction process. Specifically, the optimized point cloud is triangulated in the world frame of reference, as detailed in Section 2.5.1. After triangulation, the surface is textured according to a cost function that selects the optimal view for each triangle. The individual surfaces generated from each window are then combined to form the final TDSM.

## 2.5 Surface Reconstruction

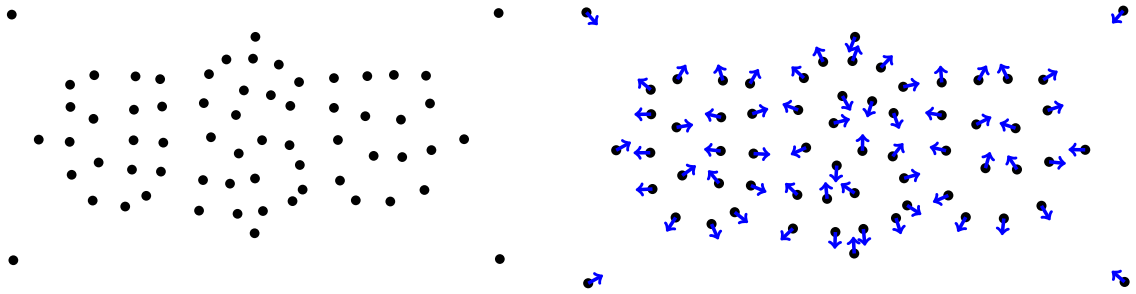
For the purposes of this work, a point cloud $\mathcal{P}$ is defined as a set of unique points $p \in \mathbb{R}^3$. Surface reconstruction is the process of generating a continuous 2D manifold from $\mathcal{P}$, which is designated as $\mathcal{S}$. This problem is ill-posed, since there are infinite possible solutions for $\mathcal{S}$. Additionally, *Point-wise noise*, *Non-uniform point distributions*, *Misalignment*, and *Missing points* complicate the surface reconstruction process.

A wide variety of methods have been proposed to address the surface reconstruction problem, each making different assumptions about the acquired data and requiring varying amounts of prior information. There are comprehensive surveys of these techniques [1, 34–37]. However, this section will focus on triangulation-based techniques and approximation-based techniques. Recently, there has been significant research into learning-based techniques that leverage deep neural networks. However, as highlighted in the surveys, triangulation-based and approximation-based methods continue to achieve the best results for point clouds with missing data, such as those collected from aerial LiDAR.

In addition to discussing triangulation-based, and approximation-based methods, this section will provide examples of these methods. These examples will use the example 2D point set in Fig. 2.7. This example includes oriented point normals, which are used in some methods.

### 2.5.1 Triangulation-Based Techniques

When the target surface $\mathcal{S}$ to be reconstructed from a point set $\mathcal{P}$ is assumed to contain all points in $\mathcal{P}$ and to be locally differentiable up to the first order, it can be



(a) 2D Point Set (b) 2D Point Set with Oriented Point Normals

Fig. 2.7: Example Point Set

modeled as a piecewise linear surface. In this formulation, $\mathcal{S}$ is represented by a triangular mesh that interpolates the points in $\mathcal{P}$. The resulting surface is a continuous 2D manifold that interpolates $\mathcal{P}$, but it is only piecewise smooth: while locally differentiable within each triangle, smoothness is not preserved along the edges where triangles meet.

Most triangulation-based techniques can be viewed as extensions of Voronoi diagrams. The Voronoi diagram of $\mathcal{P}$ partitions space into $n$ cells, one for each point $p_i \in \mathcal{P}$. The cell corresponding to $p_i$ is the set of all points $q \in \mathbb{R}^n$ that are closer to $p_i$ than to any other vertex in $\mathcal{P}$:

$$\text{Cell}(p_i) = \{\, q \in \mathbb{R}^n \mid \text{dist}(q, p_i) < \text{dist}(q, p_j),\ \forall p_j \in \mathcal{P},\ j \neq i \,\}.$$

The geometric dual of the Voronoi diagram is the Delaunay triangulation. In a Delaunay triangulation, two points $p_i, p_j \in \mathcal{P}$ are connected by an edge if their corresponding Voronoi cells $q_i$ and $q_j$ share a boundary.

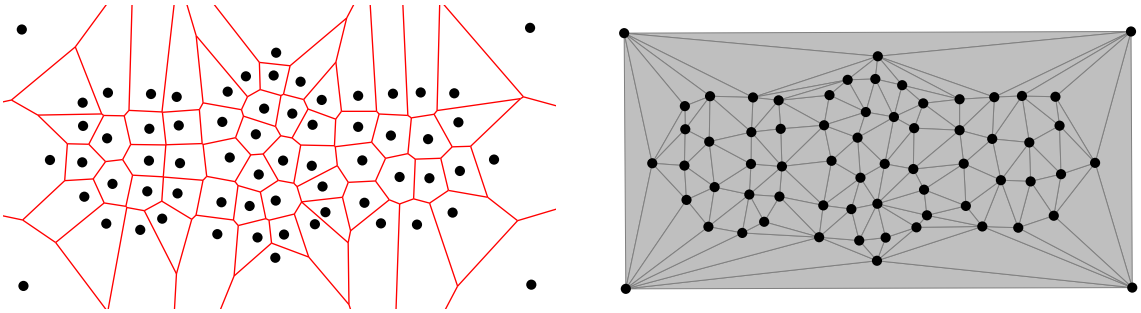
$\alpha$-shapes generalize convex hulls and are closely related to Delaunay triangulation for surface reconstruction [38]. The Ball-Pivoting Algorithm (BPA) can be viewed as an extension of $\alpha$-shapes [13]. Several related techniques exist, such as greedy surface-growing methods based on the Delaunay criterion [39]; however, BPA is presented in this work because it always creates a 2D manifold.

**Delaunay Triangulation**

Delaunay triangulation was originally developed to triangulate a set of two-dimensional points, but has been extended to higher dimensions (d-dimensional Delaunay). A Delaunay triangulation of a 2D point set is defined as a division of the set's convex hull into triangles such that the circumcircle of each triangle contains only the triangle's three vertices and no other points from the set. More formally, the Delaunay Criterion is:

$$\mathcal{S} = \{\tau_i\}_{i=1}^{N} \quad \text{s.t.} \quad \begin{array}{ll} v_{1,i} \in \mathcal{P},\ v_{2,i} \in \mathcal{P},\ v_{3,i} \in \mathcal{P}, & \forall\, \tau_i \in \mathcal{S}, \\ p \notin CC(\tau_i) & \forall\, p \in \mathcal{P}/\{v_{1,i}, v_{2,i}, v_{3,i}\},\ \ \forall \tau_i \in \mathcal{S}. \end{array} \tag{2.4}$$

Where $\mathcal{S}$ denotes the 2D manifold (surface), $\tau_i$ is the $i$-th triangle in the surface, $N$ is the total number of triangles, $v_{1,i}$, $v_{2,i}$, $v_{3,i}$ represent the 3 vertices of $\tau_i$, and $CC(\tau_i)$ denotes the circumcircle of triangle $\tau_i$. The example point cloud and its corresponding 2-dimensional Delaunay triangulation is shown in Figure 2.8b.



(a) Voronoi Diagram (b) Delaunay Triangulation

Fig. 2.8: (a) Voronoi Diagram (b) the corresponding Delaunay triangulation.

Several algorithms have been developed to construct a Delaunay triangulation from a set of 2D points: the *Flip* algorithm, *Incremental Insertion*, *Divide and Conquer*, and the *Sweepline* algorithm are the most common [40–43]. The Flip and Incremental algorithms have worst-case complexities of $\mathcal{O}(n^2)$, while Divide and Conquer and Sweepline achieve $\mathcal{O}(n \log n)$ performance.

There have also been proposed improvements to Delaunay triangulation aimed at enhancing mesh quality. Rather than solely enforcing the Delaunay criterion, these methods

maximize the minimum interior angles of the triangles and support a graded mesh structure, allowing for larger triangles in some areas and smaller ones in others [5,44,45]. These refinements to Delaunay triangulation are now standard practice when generating a triangulation in modern research.

A Delaunay triangulation of $\mathcal{P} \in \mathbb{R}^d$ divides the convex hull into $d$-simplexes (e.g., triangles in 2D, tetrahedra in 3D), such that the circumscribing hypersphere of each simplex contains only its vertices and no other points from the set $\mathcal{P}$. Delaunay surface reconstruction in three dimensions does not rely on 3D Delaunay triangulation. A 3D Delaunay triangulation fills the convex hull of a point set with tetrahedra that satisfy the Delaunay criterion. As a result, from the perspective of an external observer in three-dimensional space, only the convex hull is visible—an approximation that is unsuitable surface reconstruction.

Instead, surface reconstruction from a set of 3D points begins by projecting the points onto a 2D plane (traditionally, the XY-plane). Delaunay triangulation is then performed on the projected set, and the resulting edges are mapped back to the original 3D point locations to form the reconstructed surface.

This method of surface reconstruction is most useful when attempting to create what Berg Et al. refer to as a *terrain* [5]. A terrain is a 2D manifold in 3D Euclidean space with the property that, for every vertical line, the surface intersects the line at most once. In other words, it is the graph of a function $f : A \subset \mathbb{R}^2 \to \mathbb{R}^3$ that assigns a height $f(p)$ to each point $p$ in the *domain* $A$ of the terrain. An example of a perspective view of a terrain is found in Figure 2.9.

In previous work by Khatiwada [22, 30], a TDSM was reconstructed by first projecting the registered point cloud onto the XY-plane, performing 2D Delaunay triangulation, and then mapping the resulting mesh back to 3D. The triangulation was computed using the divide-and-conquer algorithm developed by Shewchuk [46, 47], achieving an efficient reconstruction in $\mathcal{O}(n \log n)$ time. However, this method assumes the point cloud models a terrain, an assumption that breaks down in the presence of overhangs or other concave structures captured during data collection.

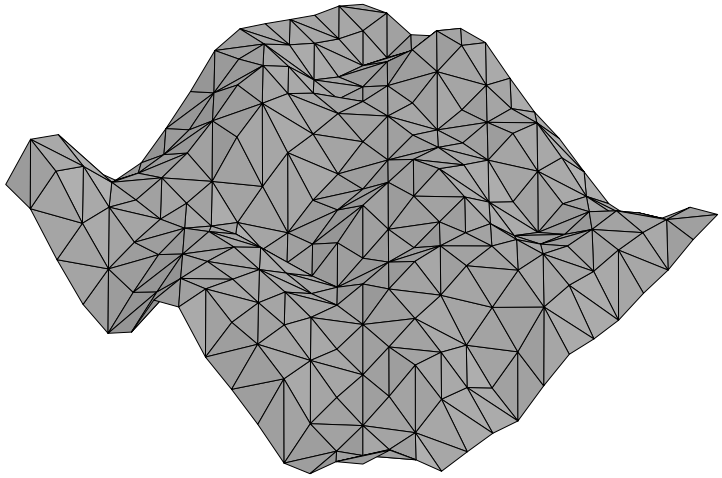
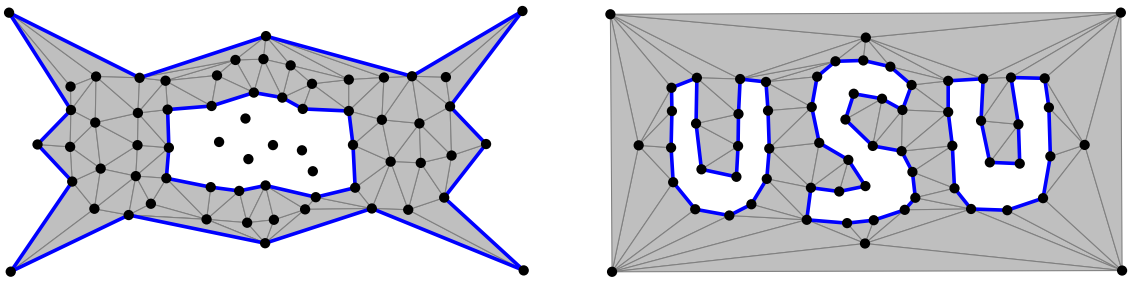


Fig. 2.9: A perspective view of a terrain

**Constrained Delaunay Triangulation**

Constrained Delaunay triangulation (CDT) extends the standard Delaunay method by allowing the insertion of additional constraint edges, even if they violate (2.4). CDT supports concave features in a triangulation, such as interior holes or a concave external boundary. In Figure 2.10a, an example of a CDT is shown where the outer boundary is explicitly constrained, while an interior hole remains unfilled. In Figure 2.10b, the outer boundary is not constrained and therefore coincides with the convex hull, but interior holes are still defined by constraints. In both figures, the constraint edges are blue, while edges created using CDT are black.

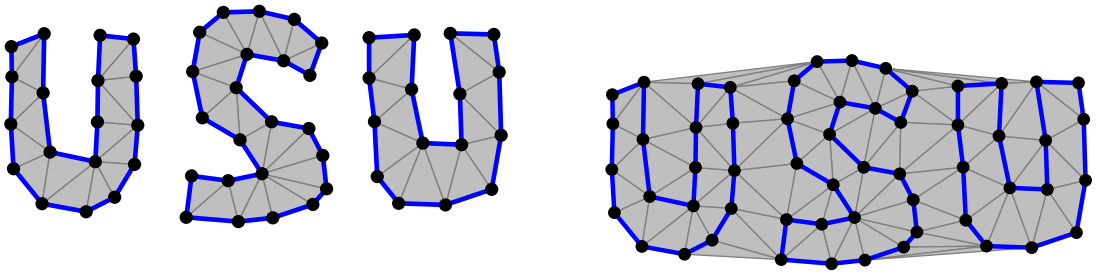


(a) CDT Example 1 (b) CDT Example 2

Fig. 2.10: Constrained Delaunay Triangulations

When using CDT for surface reconstruction, a common application is stitching together two existing triangulations. Loriot, Tournois, and Yaz [48] showed that by defining existing boundary edges as constraints, the algorithm generates a consistent triangulation between separate triangulations. A simple example of stitching existing triangulations together is shown in Figure 2.11. Similarly, if an existing triangulation contains an unwanted hole, a constrained Delaunay triangulation can be used to fill it by specifying the hole's boundary edges as constraint edges.

In work by Soucy and Laurendeau [49], multiple registered point clouds from different views were integrated into a single surface representation by selectively discarding regions of overlap between scans. Their approach begins with the construction of reliable triangulated



(a) Individual triangulations with constraint edges

(b) Stitched Triangulation

Fig. 2.11: (a) Independent triangulations (b) and corresponding stitched triangulation

subsets from each view, followed by a stitching process that merges these subsets while filling gaps with constrained Delaunay triangulation. However, this approach discards overlapping regions between subsets, effectively trading some 3D information for the benefit of a gap-free surface model.

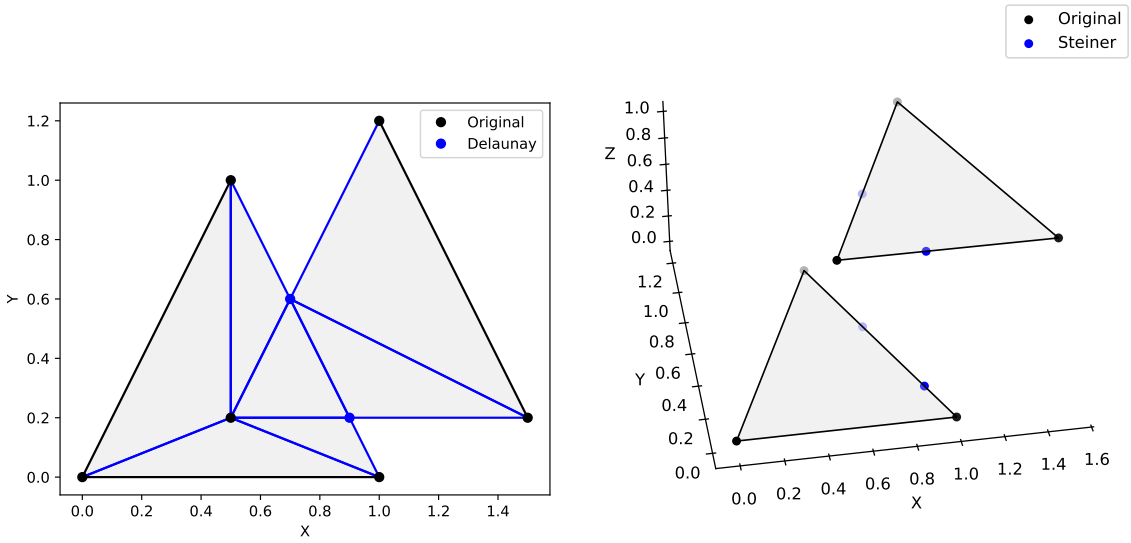
Constrained Delaunay triangulation shares the same fundamental limitation as standard Delaunay triangulation: it operates strictly in two dimensions. CDT is useful for tasks such as stitching together existing triangulations or filling holes, but these operations are not guaranteed to succeed in 3D. For instance, if two triangulated surfaces intersect when projected onto the XY plane, their constraint edges may also intersect in 2D.

To handle such cases, Hert and Seel [50] extend CDT by introducing Steiner points at intersections of overlapping constraint edges. When two edges intersect, they are subdivided into sub-edges with a Steiner point at the intersection. While this approach resolves overlaps in 2D, it violates the assumption that each vertical line intersects the surface at most once, making the mapping from 2D back to 3D ambiguous and potentially invalid.

This ambiguity is illustrated in Figure 2.12. Sub-figure 2.12b shows the original triangles along with the Steiner points in 3D. Sub-figure 2.12a depicts the CDT result, where sub-segments and Steiner points correspond to multiple possible locations in 3D space.

### Alpha Shapes

Alpha shapes were first introduced by Edelsbrunner *et al.* [38] as a generalization of




(a) 2D Projection of triangulation

(b) Original triangles in 3D

Fig. 2.12: Intersecting constraint edges

the convex hull of $\mathcal{P}$. For positive $\alpha$, the $\alpha$-shape is defined as the intersection of all closed discs (or balls in 3D) of radius $1/\alpha$ that contain $\mathcal{P}$; for negative $\alpha$, it is the intersection of the complements of such discs; and for $\alpha = 0$, it reduces to the convex hull. As $\alpha \to 0^+$, the $\alpha$-hull approaches the convex hull, while larger positive values yield hulls determined only by the most extreme points. The family of $\alpha$-hulls therefore ranges from the entire plane to the smallest enclosing circle, the convex hull, and ultimately $\mathcal{P}$ itself. This is illustrated in Figure 2.13, which shows the alpha shapes of the example 2D point cloud as the parameter $\alpha$ increases from small to large values. For small $\alpha$, the result is the original point cloud, while for large $\alpha$ the shape converges to the Delaunay triangulation of the convex hull, as discussed in Section 2.5.1. Edelsbrunner proves that the $\alpha$-shape of $\mathcal{P}$ is a subgraph of the Delaunay Triangulation for $\mathcal{P}$.

Edelsbrunner describes alpha shapes using the analogy of a scoop of ice cream filled with hard chocolate chunks. The point cloud is imagined as chocolate pieces embedded in ice cream. A spherical scoop of a specified radius alpha removes as much of the surrounding ice cream as possible without touching any of the chocolate. The remaining shape is then straightened into triangles to form the surface.

When applied to 3D surface reconstruction, the value of $\alpha$ must typically be estimated or determined empirically. The resulting hull is not necessarily convex; rather, it forms a closed surface enclosing the object, as opposed to a continuous surface with an open outer boundary. However, this approach is sensitive to variations in sampling density and does not perform well on point sets $\mathcal{P}$ with non-uniform density.

**Ball-Pivoting Algorithm**

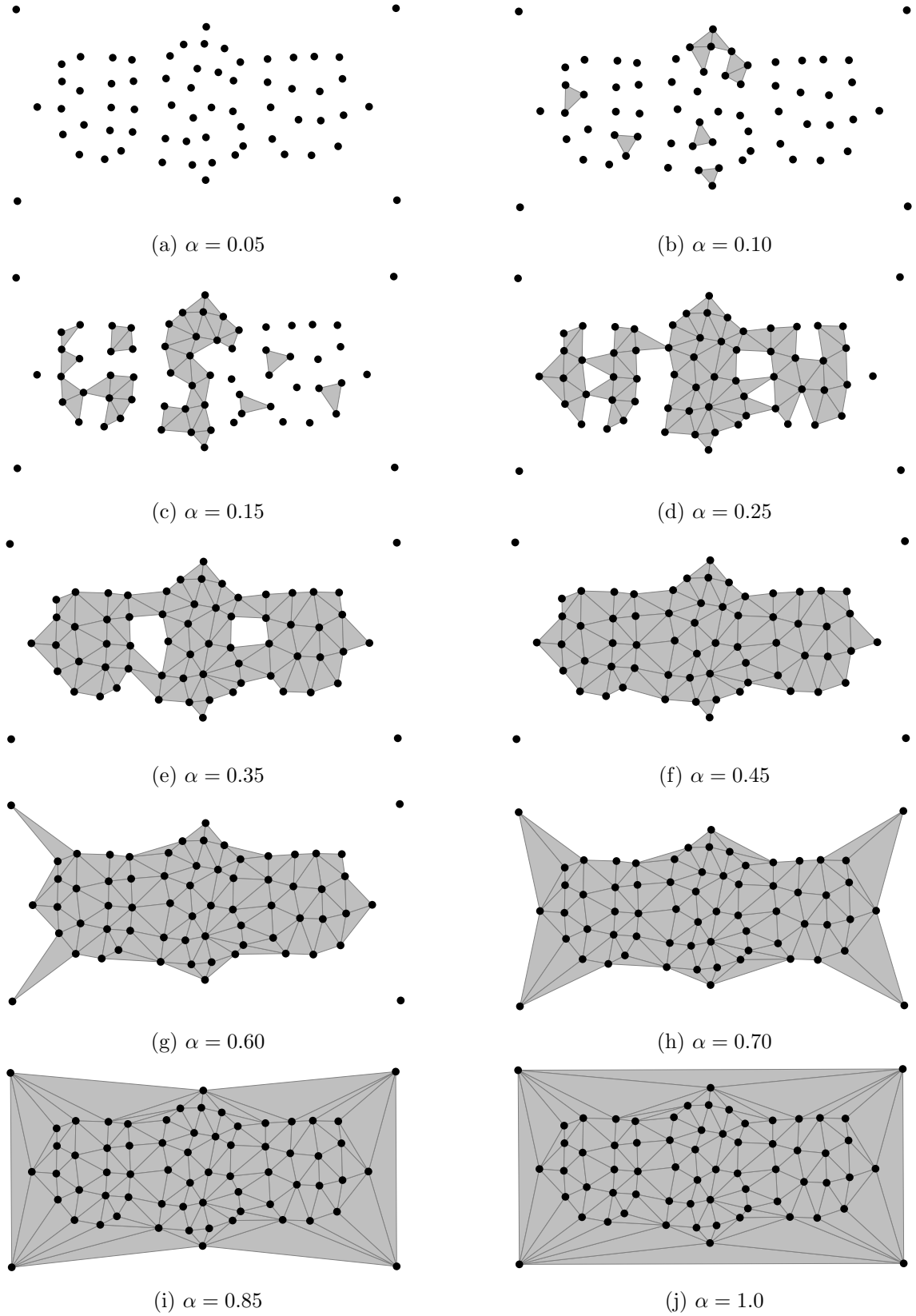
The Ball-Pivoting Algorithm (BPA) [13] can be viewed as an extension of $\alpha$-shapes, designed to handle point sets $\mathcal{P}$ with non-uniform density and to produce smoother surface reconstructions $\mathcal{S}$. Each triangle generated by BPA corresponds to a face of an $\alpha$-shape and, more generally, to a face of the Delaunay triangulation. In practice, $\alpha$-shapes computed directly from noisy point clouds often exhibit non-manifold connections, and their dependence on the full 3D Delaunay triangulation makes them impractical for large datasets. BPA addresses these issues.

BPA requires oriented point normals, which can be estimated efficiently using kd-tree-based neighborhood searches [51]. These normals are then used to compute triangle orientations and to ensure consistent surface growth by determining whether points belong to the same underlying surface.

Any triangle $\tau = \{v_1, v_2, v_3\}$ generated by BPA must satisfy the following condition: a ball of radius $r$ can be placed so that it passes through the three vertices and contains no other point of $\mathcal{P}$ in its interior. Unlike the 2D Delaunay criterion (2.4), the ball is not restricted to be centered at the circumcircle of the triangle in the plane; instead, its center lies on the perpendicular axis to the triangle's plane, offset from the circumcircle center in the direction consistent with the outward normal. Formally,

$$B(c, r) \cap \mathcal{P} = \{v_1, v_2, v_3\}, \quad \text{with } \|c - v_i\| = r \ \ \forall i \in \{1, 2, 3\}, \tag{2.5}$$

where $B(c, r)$ denotes the closed ball of radius $r$ centered at $c$, and $c$ is one of the two possible centers lying along the line through the circumcircle center perpendicular to the



(a) $\alpha = 0.05$ (b) $\alpha = 0.10$

(c) $\alpha = 0.15$ (d) $\alpha = 0.25$

(e) $\alpha = 0.35$ (f) $\alpha = 0.45$

(g) $\alpha = 0.60$ (h) $\alpha = 0.70$

(i) $\alpha = 0.85$ (j) $\alpha = 1.0$

Fig. 2.13: Alpha shapes of the example point cloud for increasing $\alpha$ values normalized so that subfigure (j) corresponds to $\alpha = 1.0$.

triangle's plane, selected according to the outward normal. This geometric condition can be intuitively understood as placing the ball on top of the circumcircle of the triangle: the ball touches the three vertices at the boundary of the circumcircle, while its center is shifted along the normal direction above the plane of the triangle.

BPA generates a triangular mesh by simulating the rolling of the ball over $\mathcal{P}$. Starting from an initial *seed triangle*, the ball pivots around boundary edges to iteratively form new triangles and grow the surface. To address non-uniform sampling density, the algorithm can be executed in multiple passes with different radii. The procedure is summarized in Algorithm 1.

**Algorithm 1:** Ball-Pivoting Algorithm

**input :** Point cloud $\mathcal{P}$; ball radius $r$ **or** a set of radii $\{r_1, \ldots, r_k\}$
**output:** Triangular mesh $\mathcal{S}$
Let $\mathcal{R} \leftarrow \{r\}$ if a single radius is provided, otherwise $\mathcal{R} \leftarrow \{r_1, \ldots, r_k\}$;
**foreach** $\rho \in \mathcal{R}$ **do**
    `/* Seed triangle */`
    Find three points forming a valid triangle such that all lie on the surface of a ball of radius $\rho$ and no other point of $\mathcal{P}$ lies inside the ball;
    **if** *no seed is found* **then**
        **continue**;
    **end**
    Initialize the *front* with the three edges of the seed triangle;
    **while** *the front is nonempty* **do**
        **foreach** *front edge* $e = (u, v)$ **do**
            Pivot a ball of radius $\rho$ around $e$ to find a third vertex $w$ that yields a valid, consistently oriented triangle;
            **if** *such a vertex* $w$ *exists* **then**
                Add triangle $(u, v, w)$ to $\mathcal{S}$;
                Mark $e$ as an inner edge;
                Add edges $(u, w)$ and $(v, w)$ to the front unless already inner or duplicated;
            **end**
        **end**
        Remove closed/duplicate edges from the front;
    **end**
**end**

The examples shown use 2D BPA, which creates segments rather than triangles. Figure 2.14 reconstructs the example point cloud using increasing ball radii. With a very small radius (Fig. 2.14a), the ball cannot reach a second sample to form an edge. As the radius grows, more candidates are rejected by the empty–ball condition because the ball contains other samples. The final panel (Fig. 2.14h) applies a multi–radius pass using all the radii from the previous examples. Figure 2.14h highlights two BPA limitations: (i) when the gap between distinct objects is smaller than the local sample spacing, BPA can spuriously bridge them; and (ii) under non-uniform sampling, selecting radii that yield a watertight surface is difficult.

Several modifications to the original algorithm have been proposed to improve radius estimation.

Digne *et al.* [52] introduce a heuristic based on the bounding box size and the number of points in the cloud; however, this method assumes that the point cloud represents a single closed object, which limits its applicability.

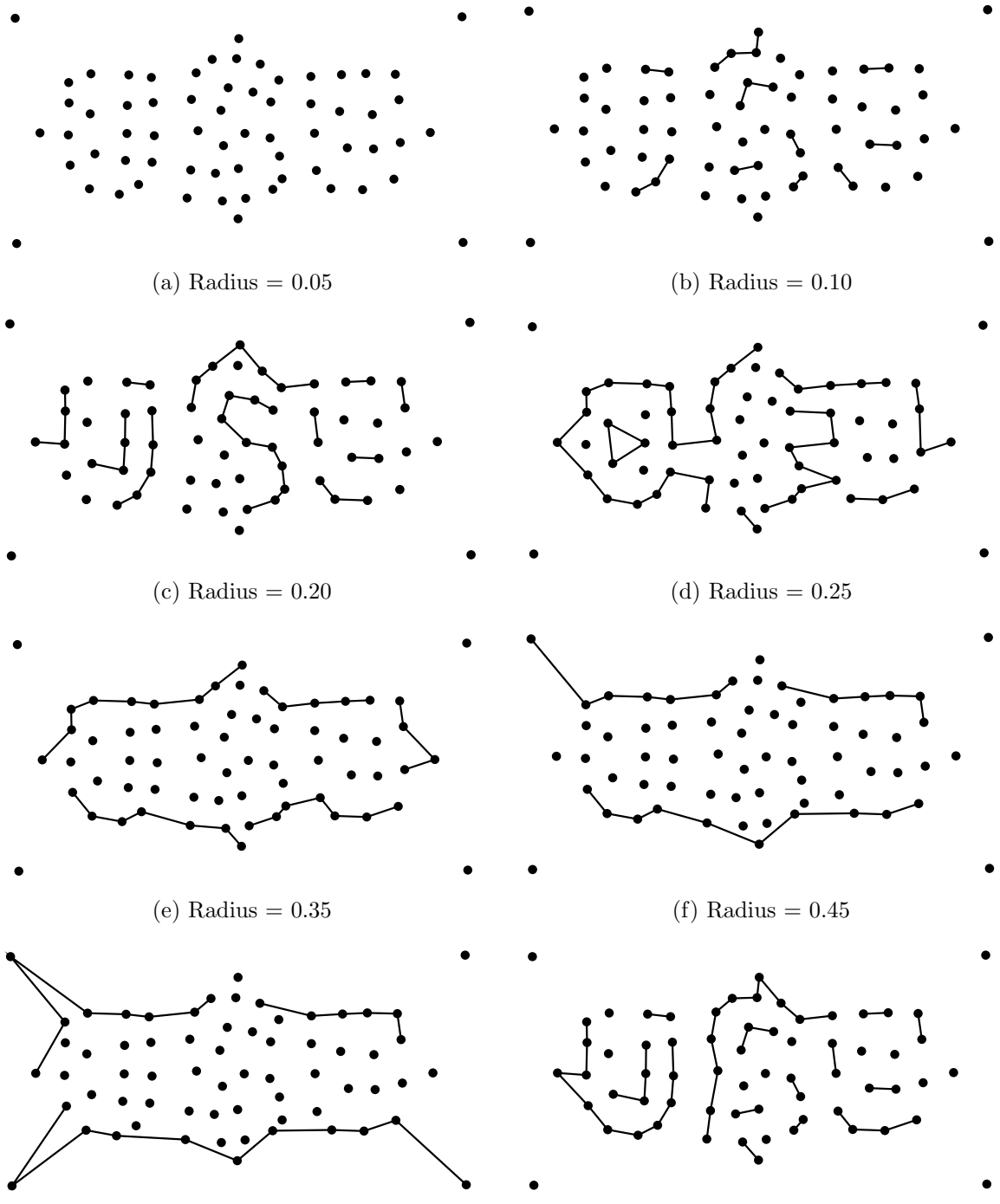
More recently, Saffi *et al.* [53] propose using Fast Point Feature Histograms (FPFH) to automatically estimate suitable radii from local geometric features, but this approach is also restricted to point clouds representing closed surfaces.

In contrast, Prithiviraj *et al.* [54] address non-uniform density by filling holes through constrained Delaunay triangulation.

### 2.5.2 Approximation-Based Techniques

*Approximation-based techniques*—often referred to as *implicit methods*—are used when the target $\mathcal{S}$ is continuously differentiable (smooth).

In these techniques, an *implicit function* is defined over 3D Euclidean space (e.g., $F(x, y, z)$. The surface is not described by $F$ directly, but instead by a subset of $\mathcal{P}$ that satisfies a condition on $F$. The *zero-level set* of $F$ is the set of all points where the function evaluates to zero: $\{(x, y, z) \mid F(x, y, z) = 0\}$. When reconstructing a surface, the zero-level set defines the surface itself, while the function's values indicate whether a point lies inside or outside the object or scene.



(a) Radius = 0.05

(b) Radius = 0.10

(c) Radius = 0.20

(d) Radius = 0.25

(e) Radius = 0.35

(f) Radius = 0.45

(g) Radius = 1.0

(h) Radii =

$\{0.05, 0.10, 0.15, \ldots, 1.00\}$

Fig. 2.14: BPA with varying radii. The radius shown for each subfigure is normalized such that the largest radius used (g) is 1.0.

Several types of implicit functions are commonly used, including signed distance functions, radial basis functions, piecewise polynomial functions, indicator functions, and wavelets [35]. These implicit functions are defined to represent the target surface $S$. For example, Hoppe et al. [55] defined the implicit function as the signed distance to the tangent plane at the nearest point $p \in \mathcal{P}$.

Because approximation techniques reconstruct a surface by fitting an implicit function $F : \mathbb{R}^3 \to \mathbb{R}$ such that its zero-level set $\{x \in \mathbb{R}^3 \mid F(x) = 0\}$ approximates the target surface, they generally alter the original point set $\mathcal{P}$. This alteration may occur by introducing new points sampled from the zero-level set, modifying the positions of the original points, or both.

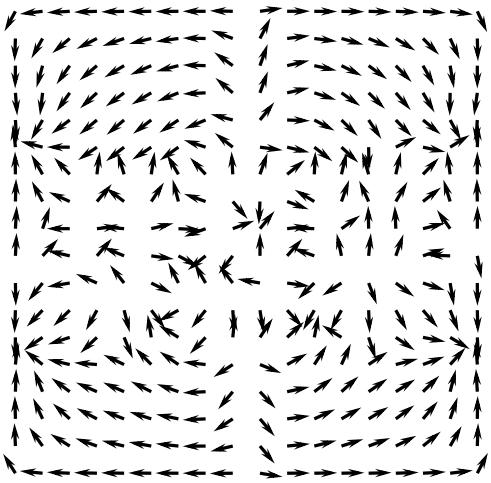
**Poisson Surface Reconstruction**

A canonical example of approximation-based reconstruction is Poisson Surface Reconstruction (PSR). PSR was first introduced by Kazhdan *et al.* [56], and is able to reconstruct a watertight surface given a point cloud $\mathcal{P}$ and the corresponding set of oriented point normals. To do this, PSR computes an *indicator function* $\chi : \mathbb{R}^3 \to \{0, 1\}$ where 1 is defined as inside the object and 0 is outside. The gradient of $\chi$ is equal to the inward-pointing normal on the surface, and is zero otherwise. So, the oriented normals from the point cloud can be viewed as samples of $\nabla\chi$ (the gradient of the indicator function).

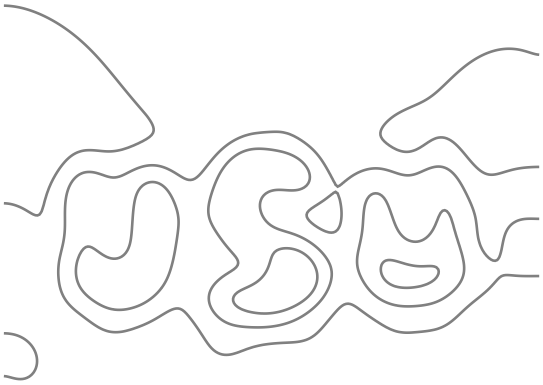
To reconstruct the surface, PSR finds $\chi$ whose gradient best fits the oriented point normals. This becomes a standard Poisson problem:

$$\Delta\chi = \nabla \cdot V, \tag{2.6}$$

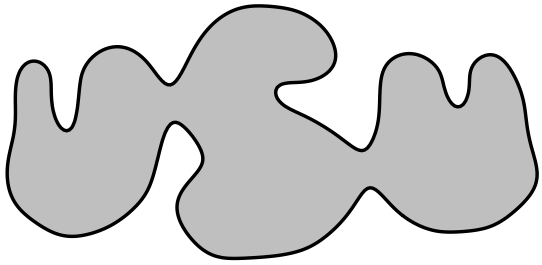
where $V$ is the vector field defined by the oriented point normals. Solving this equation yields a smooth approximation of the indicator function, from which the surface can be extracted. The indicator gradient (Fig. 2.15a), the indicator function (Fig. 2.15b), and the reconstructed surface (Fig. 2.15c) are shown for the example point set.



(a) Indicator Gradient for Example Point Set



(b) Indicator Function for Example Point Set



(c) Surface for Example Point Set

Fig. 2.15: Poisson Surface Reconstruction for Example Point Set

## 2.6 Triangulation vs. Approximation Methods

The target surface $\mathcal{S}$ to be reconstructed from the point cloud $\mathcal{P}$, in the context of the texel camera, is a 2D manifold. Unlike some reconstruction settings, it is not necessarily desired that the surface be continuously differentiable or perfectly smooth.

The input point cloud $\mathcal{P}$ is registered and optimized using SBA (see Section 2.4.2), and the points are derived directly from 3D LiDAR measurements. These measurements are assumed to lie close to the ground truth. Because texturing uses EO camera images collected from the same acquisition positions as the LiDAR points, interpolating new points introduces unnecessary artifacts where texturing may not be appropriate. The reconstruction, therefore, should rely on the measured data as faithfully as possible.

Approximation-based reconstruction techniques are often well-suited for handling noisy point clouds, since their implicit formulations naturally enforce smoothness. However, this comes at the cost of losing geometric detail in under-sampled regions and interpolating new points. In contrast, triangulation-based methods directly construct a mesh from the input point cloud, and approaches like BPA can handle non-uniform point clouds while preserving their geometric features.

# CHAPTER 3

# TEXTURED SURFACE CONSTRUCTION

This chapter presents the main contribution of this work: a new surface reconstruction pipeline. This chapter also introduces modifications to the Streaming Bundle Adjustment (SBA) regarding the selection of points included in Textured Digital Surfae Map (TDSM) creation, and changes to the texturing process.

## 3.1 Surface Reconstruction Pipeline

The inputs to the pipeline are the optimized point cloud in the world frame of reference

$$\mathcal{P}_{\text{world}} = \{p_1, p_2, \ldots, p_N\}, \quad p_i \in \mathbb{R}^3,$$

and the set of camera poses in the world frame of reference

$$\mathcal{C}_{\text{world}} = \{[R_1 \mid t_1], [R_2 \mid t_2], \ldots, [R_K \mid t_K]\},$$

where each pose $[R_k \mid t_k]$ represents the rotation and translation of the $k$-th camera view (see Section 2.4.2). The output of the pipeline is a triangulated surface and a list of obscured camera views for each triangle in the surface. The triangulated surface is given as:

$$\mathcal{S}_{\text{world}} = \big(\mathcal{P}_{\text{world}}, \mathcal{T}(\mathcal{P}_{\text{world}})\big),$$

Here $\mathcal{T}(\mathcal{P}_{\text{world}}) = \{\tau_1, \tau_2, \ldots, \tau_N\}$ denotes a set of $N$ triangles. Each triangle $\tau \in \mathcal{T}(\mathcal{P}_{\text{world}})$ is defined by three point indices in $\mathcal{P}_{\text{world}}$

$$\tau = (i, j, k), \quad i, j, k \in \{1, \ldots, |\mathcal{P}_{\text{world}}|\},$$

where $|\mathcal{P}_{\text{world}}|$ is the number of points in $\mathcal{P}_{\text{world}}$ The list of obscured views for each triangle is given as

$$\mathcal{V}_{\text{obscured}} = \left\{ (\tau, C_\tau) \;\middle|\; \tau \in \mathcal{T}(\mathcal{P}_{\text{world}}),\ C_\tau \subseteq \mathcal{C}_{\text{world}} \right\},$$

where $C_\tau$ is the subset of camera poses in $\mathcal{C}_{\text{world}}$ for which the triangle $\tau$ is not visible.

The overall process is as follows: first, cluster the input point cloud $\mathcal{P}$ into disjoint subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N$. Run the Ball Pivoting Algorithm (BPA) on each cluster to reconstruct surfaces $S_1, S_2, \ldots, S_N$. Merge the per-cluster surfaces, and then perform camera-view stitching to connect the clusters and fill inter-cluster gaps. The pipeline is shown in Fig. 3.1.

### 3.1.1 Clustering

As discussed throughout this thesis, data gathered from low-cost aerial LiDAR often has non-uniform density and missing information, since the scanner does not capture all sides of an object (Section 2.1). Furthermore, as noted in Section 2.5, missing data in point clouds remains a challenge that is not thoroughly addressed in current research. To address this issue, this work proposes grouping points with approximately uniform density into separate *clusters*, a process commonly referred to as clustering, as a first step in the surface reconstruction pipeline. In this work, the clusters are disjoint, and each contains at least four points to ensure the consistent formation of 3D geometries.

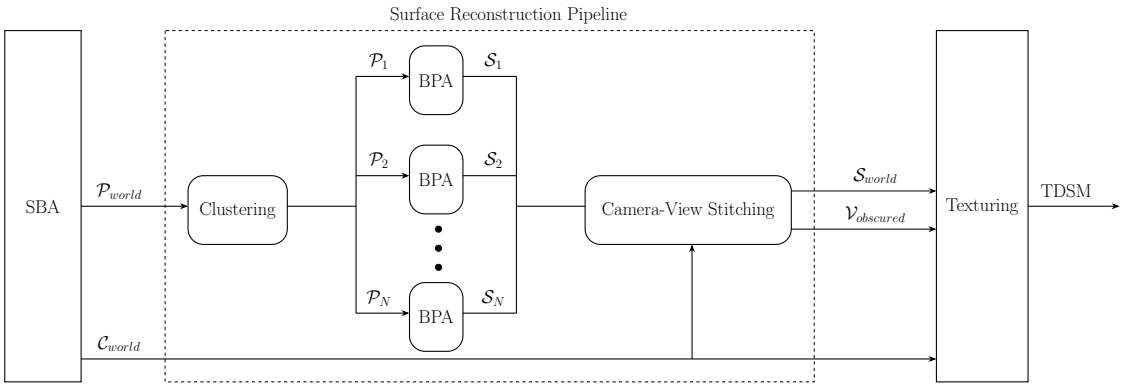



Fig. 3.1: Proposed Surface Reconstruction Pipeline

A commonly used clustering method is Density-Based Spatial Clustering (DBSCAN) [57], which groups data points based on a density threshold defined by a neighborhood radius $\varepsilon$ and a minimum number of points. Specifically, points are classified as core points if they have at least the required number of neighbors within $\varepsilon$, and clusters are formed by connecting core points and their reachable neighbors.

Clusters in data gathered from aerial LiDAR can be grouped according to density. However, since the data is typically much denser on horizontal surfaces, this work extends DBSCAN to use two density parameters: $\varepsilon_{xy}$ for the horizontal $(x/y)$ directions and $\varepsilon_z$ for the vertical $(z)$ direction. This results in ellipsoidal neighborhoods rather than spherical ones. The algorithm for the modified DBSCAN is found in Alg. 2.

To estimate $\varepsilon_z$ for a given point cloud $\mathcal{P}$, each point $p_i \in \mathcal{P}$ is considered. The $k$ nearest neighbors of $p_i$ are identified using a k-d tree, where $p_{ij}$ denotes the $j$-th nearest neighbor of $p_i$ in 3D Euclidian Space. The average deviation of the $z$-coordinates across all points is then computed as

$$\bar{z}_\sigma = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{k-1}\sum_{j=1}^{k-1}(z_{ij}-z_i)^2}, \tag{3.1}$$

where $N$ is the total number of points in the point cloud, $z_i$ is the $z$-coordinate of $p_i$, and $z_{ij}$ is the $z$-coordinate of $p_{ij}$. The final value of $\varepsilon_z$ is obtained by scaling $\bar{z}_\sigma$ with an input factor $\alpha_z$:

$$\varepsilon_z = \alpha_z \cdot \bar{z}_\sigma. \tag{3.2}$$

Here, $\alpha_z$ is a scaling parameter that controls the vertical tolerance. Similarly, the horizontal threshold $\varepsilon_{xy}$ is calculated as:

$$\varepsilon_{xy} = \alpha_{xy} \cdot \bar{xy}_\sigma. \tag{3.3}$$

The standard deviation $\bar{xy}_\sigma$ is given by

$$\bar{xy}_\sigma = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{k-1}\sum_{j=1}^{k-1} d(p_{ij},p_i)^2}, \tag{3.4}$$

**Algorithm 2:** Modified DBSCAN

**input :** point cloud $\mathcal{P} = \{p = (x_p, y_p, z_p)\}$,;
horizontal radius $\varepsilon_{xy}$,;
vertical radius $\varepsilon_z$,;
minimum points $k$
**output:** labels in $\mathcal{P}$ for disjoint clusters $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N$
**Function** $\texttt{Neighbors}_{xy,z}(p, \varepsilon_{xy}, \varepsilon_z)$**:**
  **return** $\{\, q \in \mathcal{P} \mid \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} \leq \varepsilon_{xy} \ \wedge \ |z_q - z_p| \leq \varepsilon_z \,\}$;
**end**
**begin**
  **foreach** $p \in \mathcal{P}$ **do**
    label($p$) $\leftarrow$ UNVISITED
  **end**
  $c \leftarrow 0$;
  **foreach** $p \in \mathcal{P}$ **do**
    **if** *label*($p$) $\neq$ *UNVISITED* **then**
      **continue**
    **end**
    $N \leftarrow \texttt{Neighbors}_{xy,z}(p, \varepsilon_{xy}, \varepsilon_z)$;
    **if** $|N| + 1 < k$ **then**
      label($p$) $\leftarrow$ NOISE;
      **continue**
    **end**
    $c \leftarrow c + 1$;;
    label($p$) $\leftarrow c$;
    initialize queue $Q \leftarrow N$;
    **while** $Q$ *not empty* **do**
      pop $q$ from $Q$;
      **if** *label*($q$) = *UNVISITED* $\vee$ *label*($q$) = *NOISE* **then**
        label($q$) $\leftarrow c$
      **end**
      $N_q \leftarrow \texttt{Neighbors}_{xy,z}(q, \varepsilon_{xy}, \varepsilon_z)$;
      **if** $|N_q| + 1 \geq k$ **then**
        **foreach** $r \in N_q$ ***with*** *label*($r$) = *UNVISITED* **do**
          push $r$ into $Q$;
          label($r$) $\leftarrow c$;
        **end**
      **end**
    **end**
  **end**
  **return** clusters $\{\mathcal{P}_i = \{p \in \mathcal{P} \mid \text{label}(p) = i\}\}_{i=1}^{c}$;
**end**

where $d(\cdot,\cdot)$ denotes the distance in the $x$–$y$ plane. In this case, $\alpha_{xy}$ controls the horizontal tolerance.

Using the estimates $\varepsilon_z$ and $\varepsilon_{xy}$, disjoint clusters of at least four points are constructed according to the modified DBSCAN. Each cluster contains points such that, for every point, the nearest neighbor lies within a $z$-distance of $\varepsilon_z$ and an $x/y$ distance of $\varepsilon_{xy}$.

Clustering the input point cloud $\mathcal{P}_{world}$ yields $M \geq 1$ disjoint clusters

$$\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M,$$

together with a set of outlier points $\mathcal{O}_{\text{world}}$. Each cluster satisfies

$$|\mathcal{P}_j| \geq 4, \quad \forall j = 1, \ldots, M,$$

and the partitioning condition holds:

$$\bigcup_{j=1}^{M} \mathcal{P}_j \;\cup\; \mathcal{O}_{\text{world}} \;=\; \mathcal{P}_{\text{world}}, \quad \mathcal{P}_i \cap \mathcal{P}_j = \emptyset \;\text{ for } i \neq j, \quad \mathcal{P}_j \cap \mathcal{O}_{\text{world}} = \emptyset.$$

### 3.1.2 BPA

As discussed in Section 2.6, triangulation-based surface reconstruction methods are suitable for texel camera data. This work employs BPA (Section 2.5.1) because it consistently produces a 2D manifold—without non-manifold vertices or edges. Furthermore, BPA generates a smoother mesh compared to other greedy Delaunay-based methods. By selecting an appropriate radius, BPA can produce a smooth surface even in the presence of point-wise noise, since the ball naturally rolls over the underlying geometry.

BPA is applied independently to each cluster to generate a smooth surface. After clustering, the point density within each cluster may still vary slightly, but it remains sufficiently uniform to justify estimating the BPA radii from the statistical properties of each cluster.

A simple heuristic is proposed to estimate a suitable radius for each cluster. Given the

$j$-th cluster, $\mathcal{P}_j$, for each point $p \in \mathcal{P}_j$, let $p_3$ be its third nearest neighbor, identified using a k-d tree. The radius $r_j$ for cluster $\mathcal{P}_j$ is then defined as the maximum distance from any $p \in \mathcal{P}_j$ to its third nearest neighbor:

$$r_j = \max_{p \in \mathcal{P}_j} \|p - p_3\|_2, \tag{3.5}$$

Although each cluster point cloud $\mathcal{P}_j$ is fairly uniform, the BPA radius is chosen to reach the sparsest regions while suppressing point-wise noise in denser areas. In very dense regions, BPA may fail to form triangles due to the extended Delaunay criterion, which requires that the ball contain only the three vertices of a triangle $\tau = (v_1, v_2, v_3) \subset \mathcal{P}$ and no other points of $\mathcal{P}$. To correct this, after an initial BPA pass on each cluster, all unused vertices are temporarily removed, and BPA is rerun with the same radius to fill holes in dense regions. Unused vertices are given by the following:

$$\mathcal{U}_j = \{v \in \mathcal{P}_j \mid v \notin \tau \ \ \forall \tau \in \mathcal{T}(\mathcal{P}_j)\}, \tag{3.6}$$

where $\mathcal{T}(\mathcal{P}_j)$ is the set of generated triangles for cluster $\mathcal{P}_j$.

The result of BPA on the clustered point cloud is a collection of triangulated surfaces $\{S_1, S_2, \ldots, S_M\}$, one for each cluster $\mathcal{P}_j$, together with a corresponding set of unused vertices $U_j \subseteq \mathcal{P}_j$. Both $S_j$ and $U_j$ may be empty; for example, if $\mathcal{P}_j$ contains only nearly colinear points, then no triangles can be formed and thus all points belong to $U_j$. Conversely, if the triangulation includes all points from $\mathcal{P}_j$, then $U_j$ is empty. Each surface $S_j$ is defined by its vertex set $\mathrm{Vert}(S_j) \subseteq \mathcal{P}_j$ and triangle set $\mathcal{T}(\mathcal{P}_j)$, where each $\tau \in \mathcal{T}(\mathcal{P}_j)$ is represented by three indices into the point cloud $\mathcal{P}_j$. The union of all used vertices, unused vertices, and outlier points recovers the original point cloud:

$$\bigcup_{j=1}^{M} \big(\mathrm{Vert}(S_j) \cup U_j\big) \ \cup \ \mathcal{O}_{\mathrm{world}} \ = \ \mathcal{P}_{\mathrm{world}}. \tag{3.7}$$

Finally, each $\tau \in \mathcal{T}(\mathcal{P}_j)$ is mapped to three indices into the original point cloud $\mathcal{P}_{\mathrm{world}}$. The union of all surfaces, unused points, and outlier points yields the surface in the world frame

of reference:

$$S_{\text{world}} = \bigcup_{j=1}^{M} (S_j \cup U_j) \cup \mathcal{O}_{\text{world}} = (\mathcal{P}_{\text{world}}, \mathcal{T}(\mathcal{P}_{\text{world}})). \tag{3.8}$$

### 3.1.3 Camera-View Stitching

$S_{\text{world}}$ is a triangulated surface that may contain unused points and disjoint triangulations. The disjoint triangulations arise from missing information in the original scanned data, while unused points typically result from point-wise noise or registration errors. To stitch together the disjoint geometries in $S_{\text{world}}$, constrained Delaunay triangulation (CDT) is applied to fill gaps that are visible from each camera view.

A point $p \in S_{\text{world}}$ is *visible* from camera $j$ if, in the world frame of reference, the ray from the camera center ($c_j$of $[R_j \mid t_j]$ to $p$ does not intersect any other point or triangle of $S_{\text{world}}$ at a smaller depth value.

To determine visibility of points, raycasting [51] is used. For each point $p \in S_{\text{world}}$, a ray

$$r_{j,p}(t) = c_j + t(p - c_j), \quad t \geq 0,$$

is cast from every $j$th camera center $c_j \in \mathcal{C}_{\text{world}}$ toward every $p \in S_{\text{world}}$. If any such ray intersects another point (unlikely in real data) or triangle of $S_{\text{world}}$ at a smaller depth than the target point, then the point is considered hidden from that camera view, otherwise, it is considered visible.

A triangle $\tau \in S_{\text{world}}$ is *visible* from camera $j$ if, when projected into the normalized image plane of $[R_j \mid t_j]$ , no other point or triangle of $S_{\text{world}}$ projects to the same image region with a smaller depth value when mapped back into $\mathbb{R}^3$.

To determine triangle visibility, the sweepline algorithm (Appendix B) is applied. The world point cloud $P_{\text{world}}$ is first rotated into the $j$th camera space using (2.2) and then projected into the normalized image plane via (2.3). Within this 2D image plane, the sweepline algorithm detects triangle intersections and computes the corresponding intersection points. Each intersection point is back-projected to 3D according to Appendix C, and the triangle with the smaller range measurement is identified as visible, while the other is marked as

occluded.

The process for camera-view stitching is outlined in Algorithm 3. First, triangles that are hidden from all views are removed. Next, the algorithm performs constrained Delaunay triangulation (CDT) for each camera view and retains only the triangles that do not contain Steiner points (see Section 2.5.1). This selection ensures that the stitching process uses only triangles that are fully visible from the corresponding camera view. Finally, the list of obscured views is constructed.

**Algorithm 3:** Camera-View Stitching

**input :** $\mathcal{P}_{\text{world}}$, $\mathcal{C}_{\text{world}}$, $S_{\text{world}}$
**output:** Updated $S_{\text{world}}$ and obscured sets $\mathcal{V}_{\text{obscured}}(\tau)$
**for** $i \leftarrow 1$ **to** Size($\mathcal{C}_{world}$) **do**
  rotate $\mathcal{P}_{\text{world}}$ into camera space $i$ using (2.2);
  project into the normalized image plane using (2.3) to obtain $\mathcal{P}^{(i)}_{\text{proj}}$;
**end**
**for** $i \leftarrow 1$ **to** Size($\mathcal{C}_{world}$) **do**
  identify hidden points from camera $i$ to obtain $\mathcal{P}^{(i)}_{\text{hidden}}$;
  identify hidden triangles from camera $i$ to obtain $\mathcal{T}^{(i)}_{\text{hidden}}$;
**end**
identify points hidden from *every* view as $\mathcal{P}_{\text{hidden}} \leftarrow \bigcup_{i=1}^{n} \mathcal{P}^{(i)}_{\text{hidden}}$; remove from $\mathcal{S}_{world}$;
identify triangles hidden from *every* view as $\mathcal{T}_{\text{hidden}} \leftarrow \bigcup_{i=1}^{n} \mathcal{T}^{(i)}_{\text{hidden}}$; remove from $\mathcal{S}_{world}$;
**for** $i \leftarrow 1$ **to** Size($\mathcal{C}_{world}$) **do**
  perform CDT on $S_{\text{world}}$ using $\mathcal{P}^{(i)}_{\text{proj}}$;
  insert new triangles that contain no Steiner vertices into $S_{\text{world}}$;
**end**
**for** $i \leftarrow 1$ **to** Size($\mathcal{C}_{world}$) **do**
  identify hidden triangles from camera $i$ and record in $\mathcal{V}_{\text{obscured}}(\tau)$;
**end**

After camera-view stitching, the surface reconstruction pipeline is complete. The resulting $\mathcal{S}_{\text{world}}$ consists of clusters triangulated using BPA and stitched together through per-view CDT. Every triangle in $\mathcal{S}_{\text{world}}$ is visible from at least one camera view, and $\mathcal{V}_{\text{obscured}}$ records, for each triangle, the set of views from which it is obscured.

## 3.2 TDSM Creation

In previous work (Section 2.4.2), a TDSM was generated from each $\mathcal{L}$-sized window of texel images (Figure 2.6). The underlying assumption of the $\mathcal{L}$ parameter is that any point observable in texel image $i$ is also visible within the range of images from $i - \mathcal{L}$ to $i + \mathcal{L}$. However, the original approach constructed the TDSM using only the current window of texel images. This could result in incompleteness: the first image in the window might miss points that appear in the preceding $\mathcal{L}$ images, and the last image might similarly miss points that appear in the subsequent $\mathcal{L}$ images.

To address this limitation, the TDSM creation is extended by including $\mathcal{L}$ images on either side. To achieve this, the original SBA is modified such that the TDSM is not created immediately after the current texel images are optimized. Instead, its creation is delayed by one iteration, allowing the TDSM to be constructed from the present texel images together with $\mathcal{L}$ images in the past and $\mathcal{L}$ images in the future, as illustrated in Figure 3.2. In this figure, the example TDSM would be created one SBA iteration in the future, so that the past, present, and future texel images are all fully optimized.

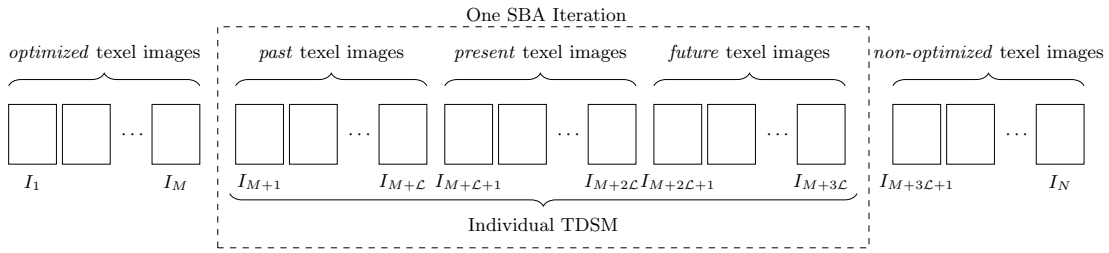



Fig. 3.2: Modified SBA Window

By extending the TDSM creation to include $\mathcal{L}$ images on either side, TDSMs now overlap with each other as shown in Fig. 3.3. If we generate a sequence of TDSMs $(\text{TDSM}_1, \text{TDSM}_2, \ldots, \text{TDSM}_N)$, each $\text{TDSM}_i$ for $i \neq 1$ includes identical points from $2\mathcal{L}$ Texel images that overlap with $\text{TDSM}_{i-1}$, and for $i \neq N$ also includes identical points from $2\mathcal{L}$ Texel images that overlap with $\text{TDSM}_{i+1}$. Similarly, each $\text{TDSM}_i$ with $i \neq 1, 2$ includes

identical points from $\mathcal{L}$ Texel images overlapping with TDSM$_{i-2}$, and for $i \neq N, N-1$ also includes identical points from $\mathcal{L}$ Texel images overlapping with TDSM$_{i+2}$. No other TDSMs overlap.

The overlap between TDSMs results from using the $\mathcal{L}$ parameter to ensure that all visible points are included in each TDSM. Although this overlap is not detrimental to TDSM quality, it introduces redundancy during surface reconstruction.

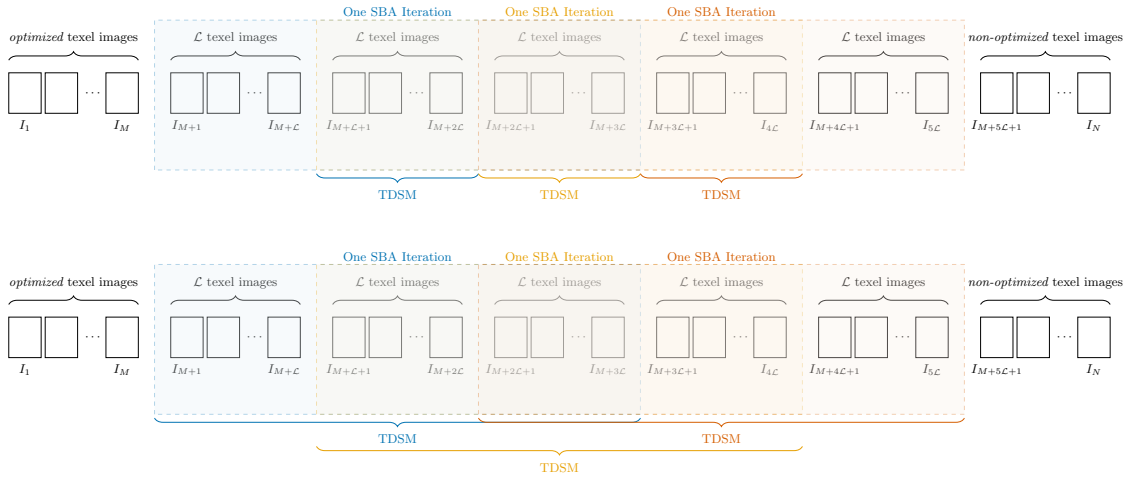



Fig. 3.3: Comparison of Old TDSM Creation (top) and New TDSM Creation(bottom).

In previous work, the texturing process selected a texture for each triangle using a cost function developed by Khatiwada [22]. This cost function determined which camera view offered the best texture for each triangle.

As discussed in Section 3.1.3, the new surface reconstruction pipeline collects, for each triangle, a list of camera views from which that triangle is obscured. In this work the texturing process, doesn't consider any camera views that cannot see a given triangle. This ensures that triangles receive textures that accurately correspond to their true 3D location.

# CHAPTER 4

# RESULTS

This chapter presents the results of the new surface reconstruction pipeline. First, the experimental setup is described. Because no ground-truth surface exists for the real-world data, no quantitative error metrics are used. Instead, qualitative visual comparisons are presented across multiple datasets. Finally, a computational performance comparison is provided.

## 4.1 Experimental Setup

Data were collected using the Texel Camera (see Section 2.1) mounted on a small Unmanned Aerial Vehicle (sUAV). Both flights used a helicopter-type (multirotor) platform, which tilts to generate forward motion to change direction.

The first flight was conducted over the Utah State University (USU) campus at an altitude of about 50 m, with a total duration of 4 minutes. The flight path is shown in Fig. 2.3.

The second flight was performed over a building north of the USU campus and the surrounding area, at an altitude of about 50 m with a total duration of 10 minutes.

The pipeline registers and optimizes the data as described in Chapter 2, incorporating the modifications in Section 3.2. After the registration and optimization, the result is a point cloud. The same point cloud is used to generate a Textured Digital Surface Map (TDSM) using (i) the previous method - projection to the $xy$ plane followed by 2D Delaunay triangulation - and (ii) the proposed method - clustering, Ball-Pivoting Algorithm (BPA), and camera view stitching. After the triangulation is generated, both methods used the same texturing process described in Section 2.4.2.

## 4.2 Visual Comparison

In this section, visual comparisons are presented. Each scene includes a textured point cloud, a wireframe reconstruction using both the proposed method and the Delaunay method, and finally, a TDSM generated by each approach.

Figures 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 present results from Flight 1.

Figures 4.1 and 4.2 show USU Natural Resources building. When reconstructed with the Delaunay method, the building edge is incorrectly triangulated with the ground, producing many thin triangles that span between the ground and the roof and result in a spiky appearance. In contrast, the proposed method generates a continuous surface that connects the building edge to the ground. This produces a better representation of the structure and enables more accurate texturing in the TDSM, where the wall texture is mapped correctly to the building.

Figure 4.3 shows another building and its surrounding area. In this case, the roof extends significantly beyond the ground footprint, and the Texel camera didn't capture texture on the building's side. The Delaunay method triangulates the rooftop to the ground even within the edge of the roof, while the proposed method forms a single surface from the edge of the roof to the ground. The new method shows an improved surface for the roof and the ground, respectively; however, it is unable to texture the walls of the building well.

Figures 4.4 and 4.5 present a courtyard along with the surrounding trees and buildings. Here, the proposed method produces smoother building and tree edges. Although the building's side texture is somewhat dark, the method still yields a more accurate surface and places the texture correctly on the building's walls.

Finally, Figure 4.6 presents the taller Huntsman building on the USU campus. This example again shows that the new method is better able to represent the underneath overhangs of the building, including a large texture on the side of the building. This example also includes trees and shrubs, which are smoother.

Figures 4.7, 4.9, and 4.10 present results from Flight 2.

Figure 4.7 shows two cars parked side by side. The Delaunay method produces triangles between the cars and the ground, even in regions that are not at the car edges. In contrast, the proposed method reconstructs smooth car surfaces and creates a more accurate connection to the ground.

Figure 4.9 depicts a small shed on the left. To the right of the shed are a picnic table and the drone pilot, followed further right by a light pole and a parking lot with several cars. The proposed method extends the ground surface beneath the shed overhang and connects it appropriately to the roof. It also produces more accurate surfaces for small objects such as the table and the light pole.

Figure 4.10 presents a row of cars in a parking lot adjacent to a tree. The proposed method reconstructs the tree canopy as a smoother object, avoiding the repeated stretched triangles that the Delaunay method introduces between the canopy and the ground. The cars exhibit a similar improvement.

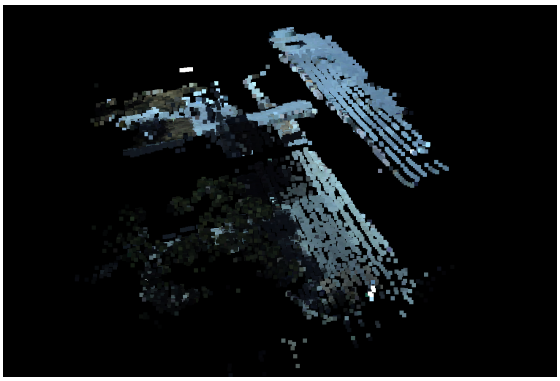
Finally, Figure 4.8 shows a building along with a tree. The edges of the building in the Delaunay reconstruction are jagged while the new reconstruction creates a better representation where the overhangs of the building are not connected to the ground in places where they should not be.

### 4.3 Computational Comparison

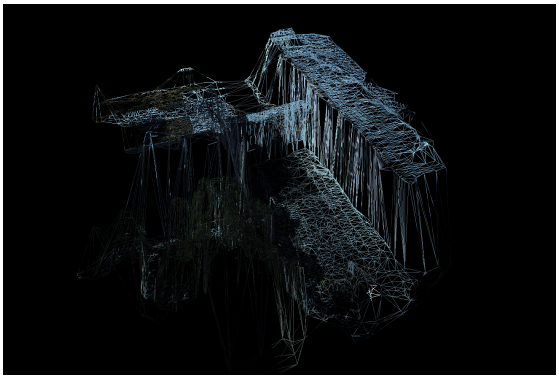
This section will present a computational comparison between the previous surface reconstruction method, and the proposed method.

Previous work: Let $n = |\mathcal{P}|$ be the number of points. The baseline method consists of three steps:

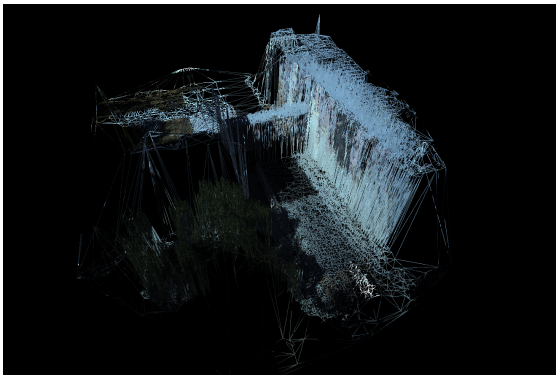
$$
\begin{aligned}
T_{\text{proj}}(n) &= a_1 n, && \text{(project to } xy\text{-plane)} \\
T_{\text{Del}}(n) &= a_2\, n \log n, && \text{(2D Delaunay)} \\
T_{\text{lift}}(n) &= a_3 n, && \text{(map back to 3D)}
\end{aligned}
$$



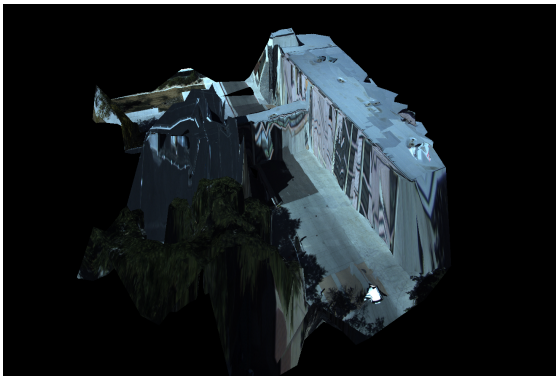
(a) Textured Point Cloud



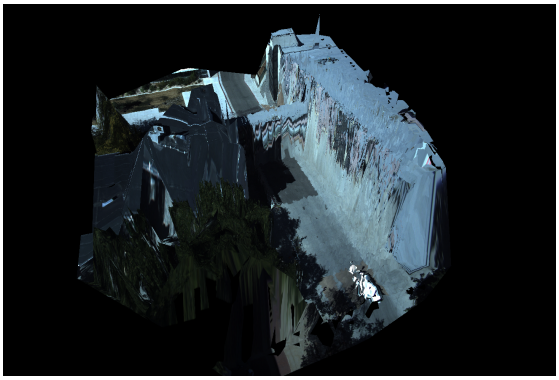
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

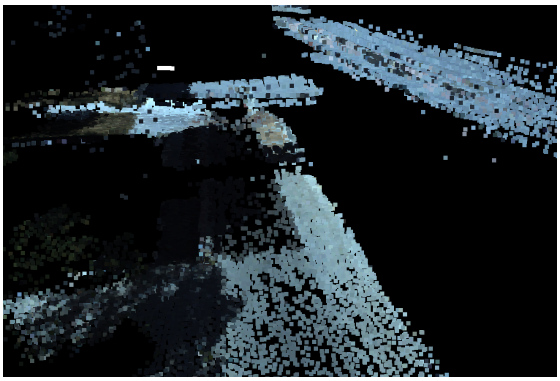


(d) New Method TDSM

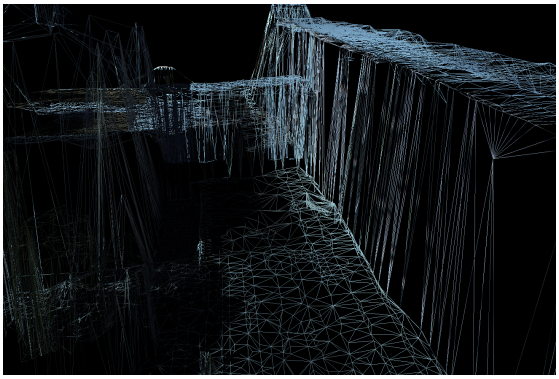


(e) Delaunay TDSM

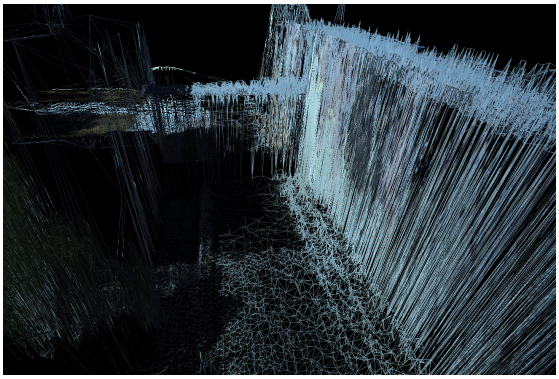
Fig. 4.1: Flight 1 Scene 1 Reconstruction



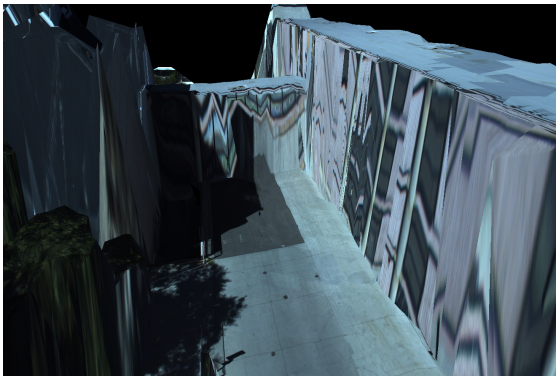
(a) Textured Point Cloud



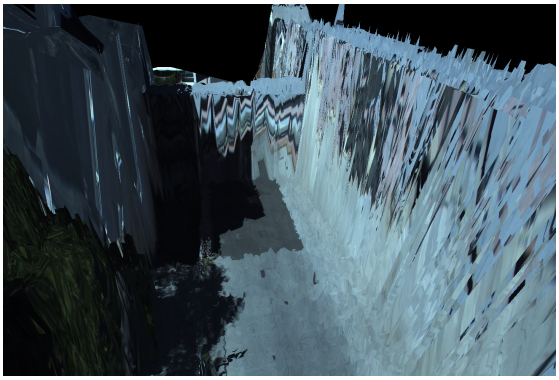
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

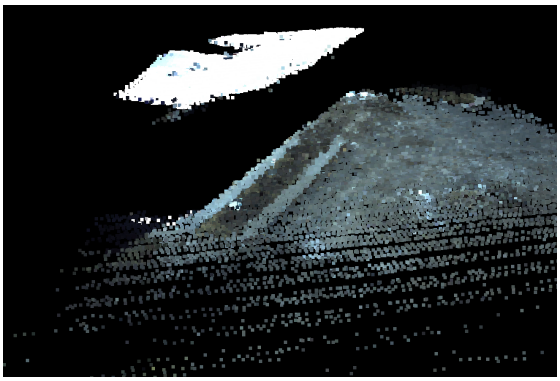


(d) New Method TDSM

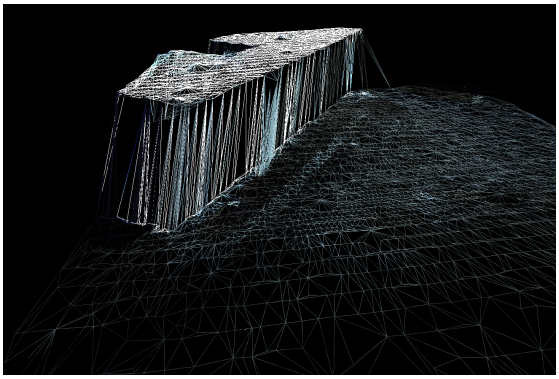


(e) Delaunay TDSM

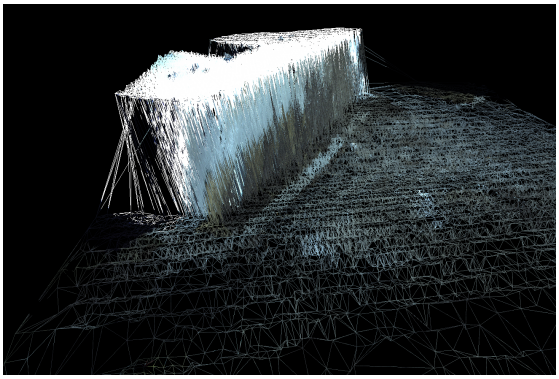
Fig. 4.2: Flight 1 Scene 1 Reconstruction (Different Perspective)



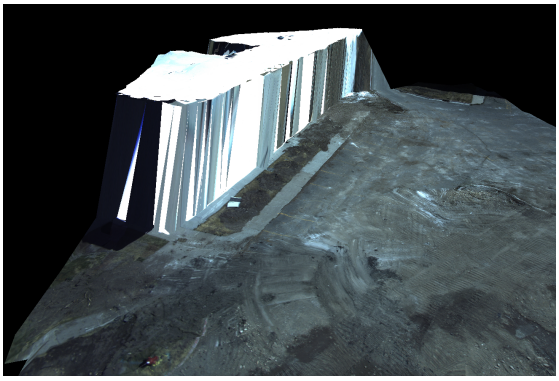
(a) Textured Point Cloud



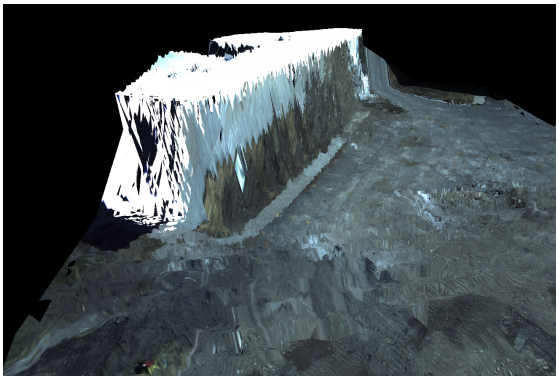
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

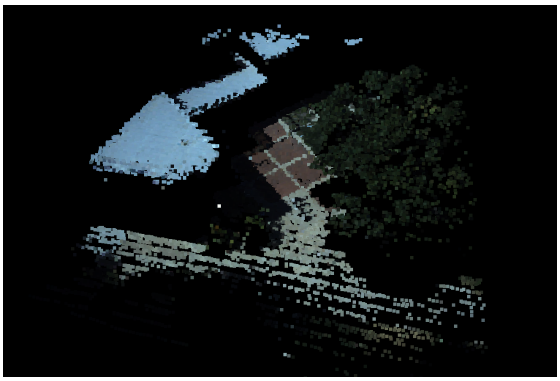


(d) New Method TDSM

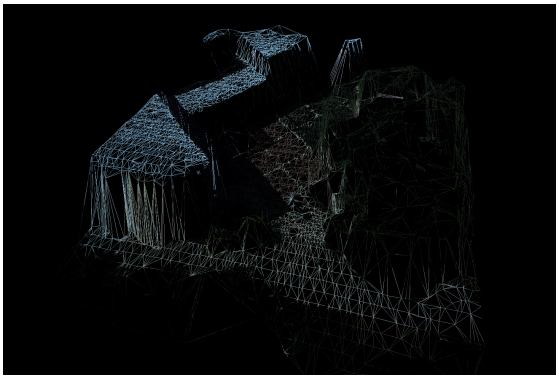


(e) Delaunay TDSM

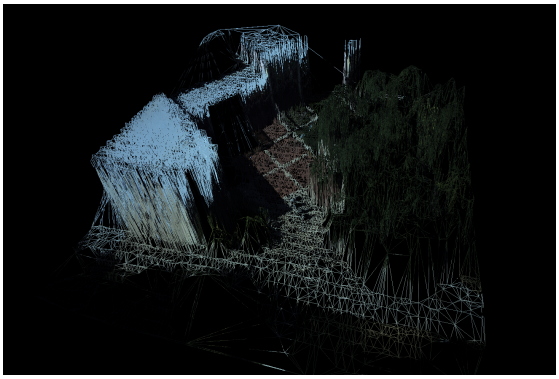
Fig. 4.3: Flight 1 Scene 2 Reconstruction



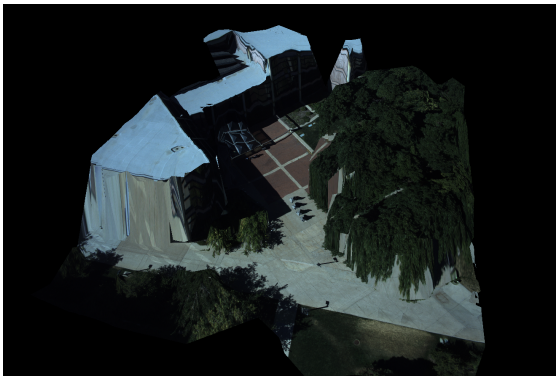
(a) Textured Point Cloud



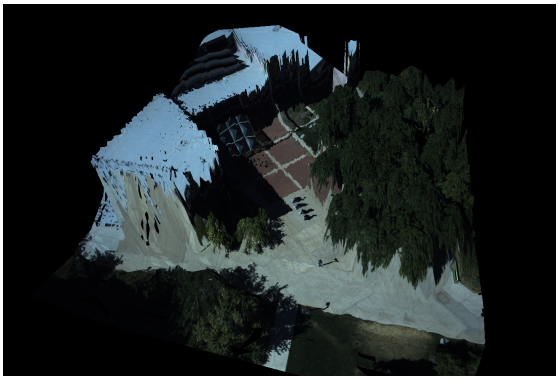
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

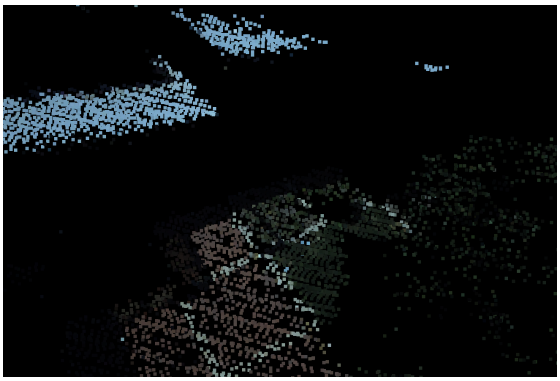


(d) New Method TDSM

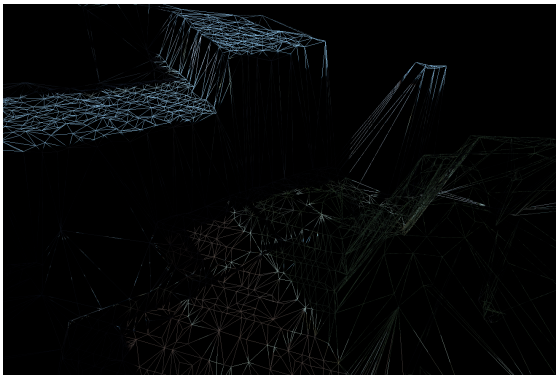


(e) Delaunay TDSM

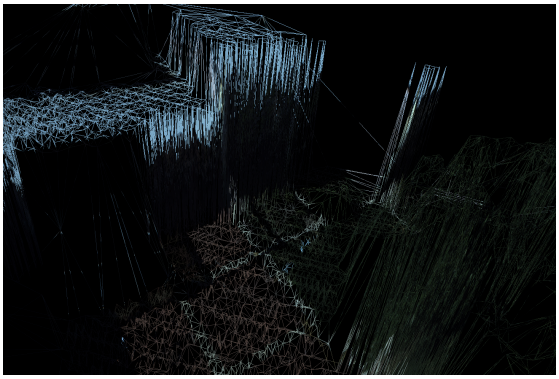
Fig. 4.4: Flight 1 Scene 3 Reconstruction



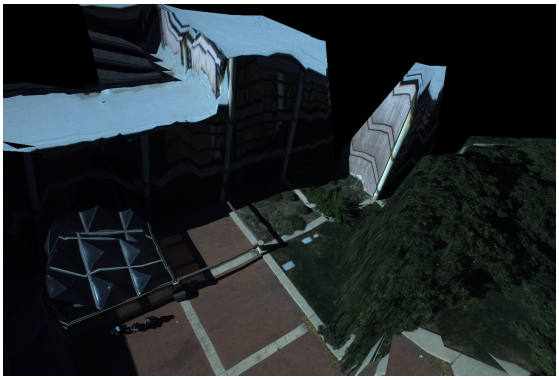
(a) Textured Point Cloud



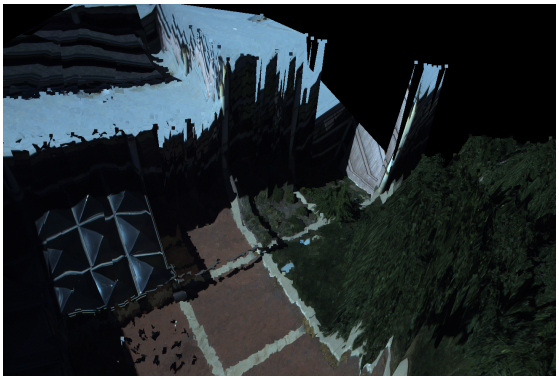
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

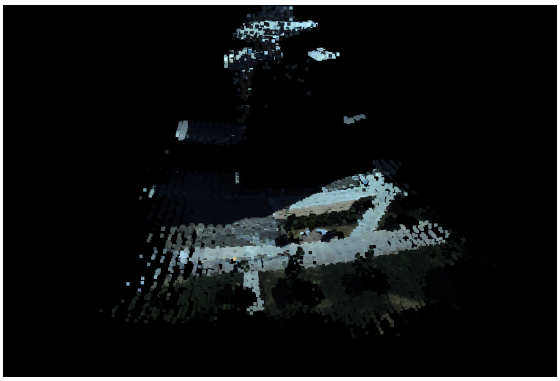


(d) New Method TDSM

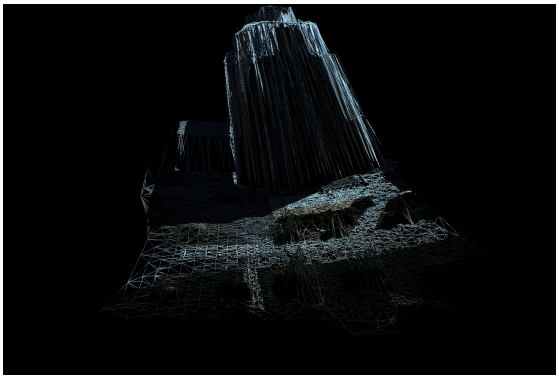


(e) Delaunay TDSM

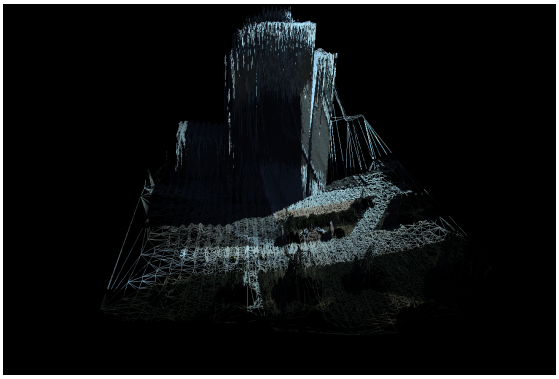
Fig. 4.5: Flight 1 Scene 3 Reconstruction (Different Perspective)



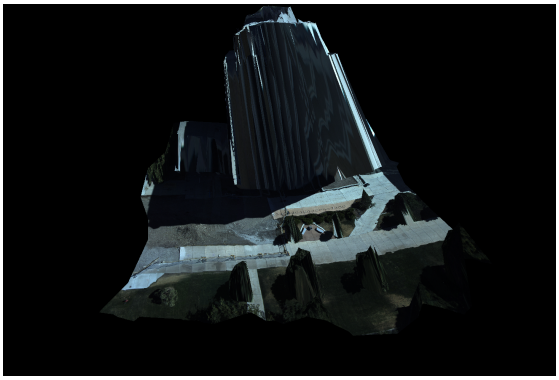
(a) Textured Point Cloud



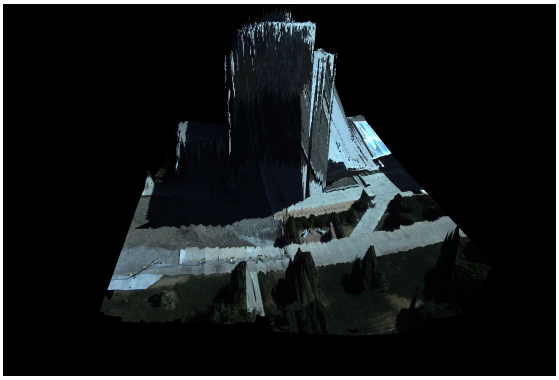
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

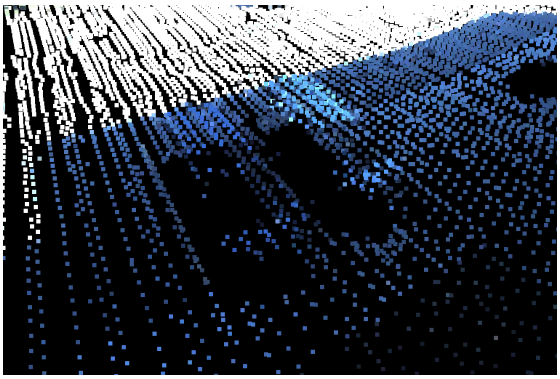


(d) New Method TDSM

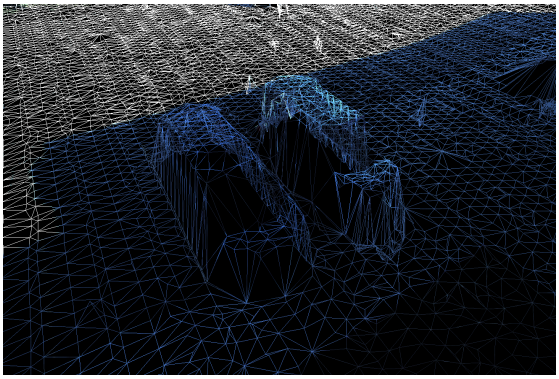


(e) Delaunay TDSM

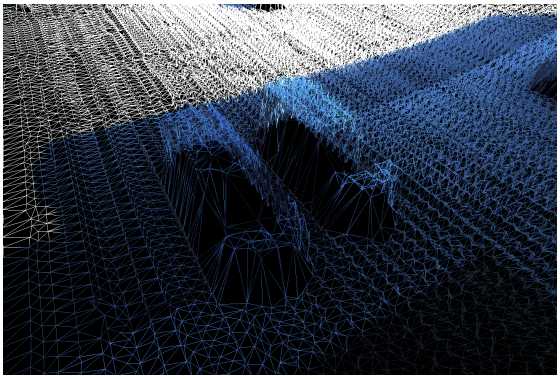
Fig. 4.6: Flight 1 Scene 4 Reconstruction



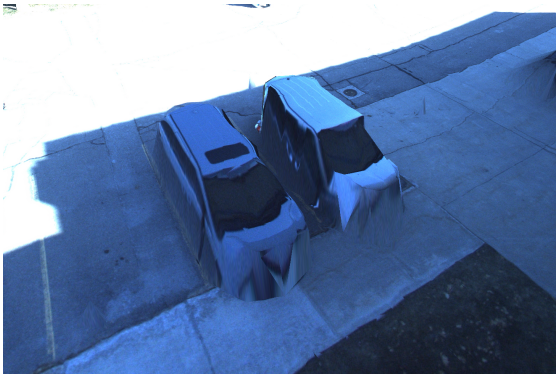
(a) Textured Point Cloud



(b) New Method Wire-frame



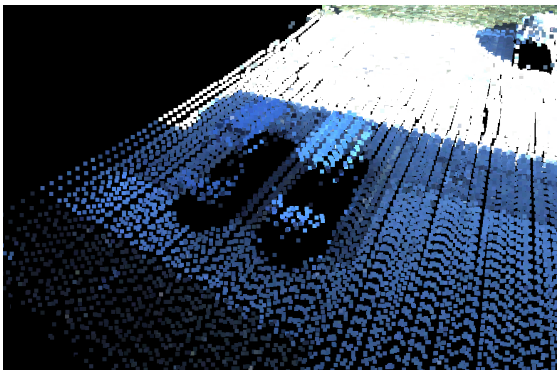
(c) Delaunay Wire-Frame



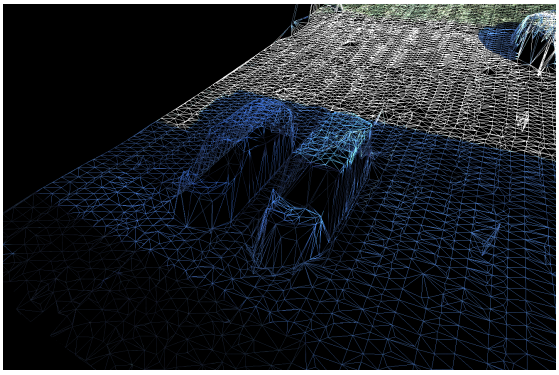
(d) New Method TDSM

(e) Delaunay TDSM

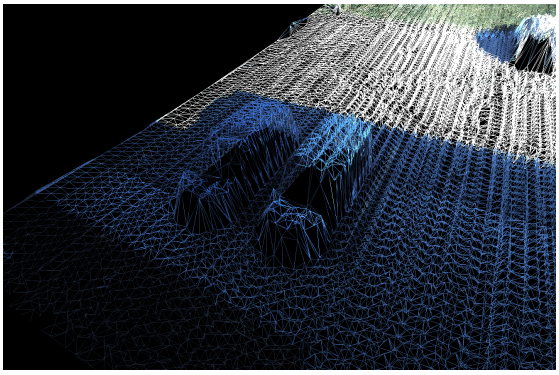
Fig. 4.7: Flight 2 Scene 1 Reconstruction



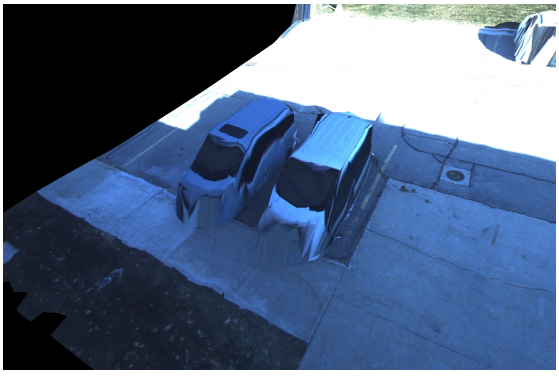
(a) Textured Point Cloud



(b) New Method Wire-frame



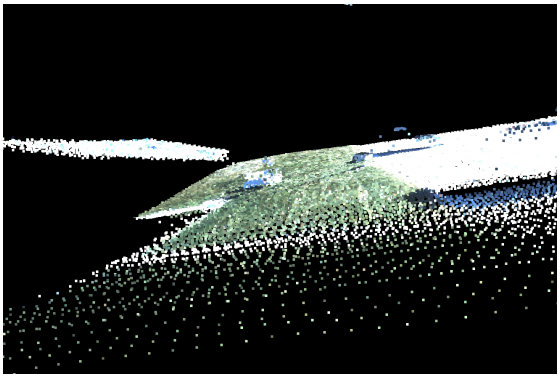
(c) Delaunay Wire-Frame



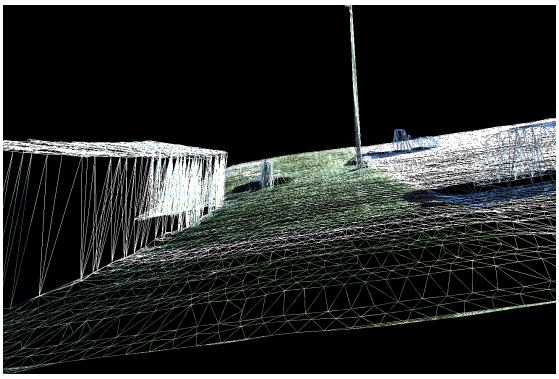
(d) New Method TDSM

(e) Delaunay TDSM

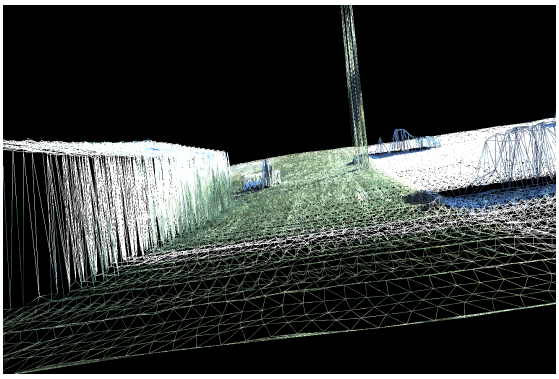
Fig. 4.8: Flight 2 Scene 1 Reconstruction (Different Perspective)



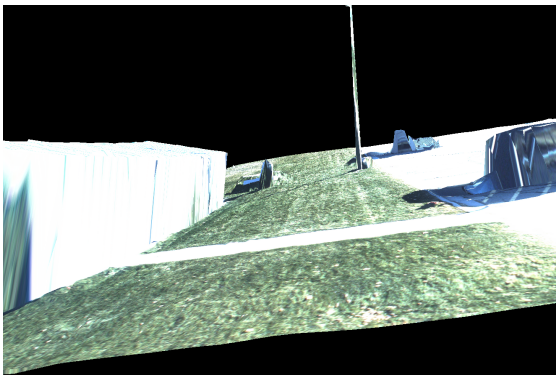
(a) Textured Point Cloud



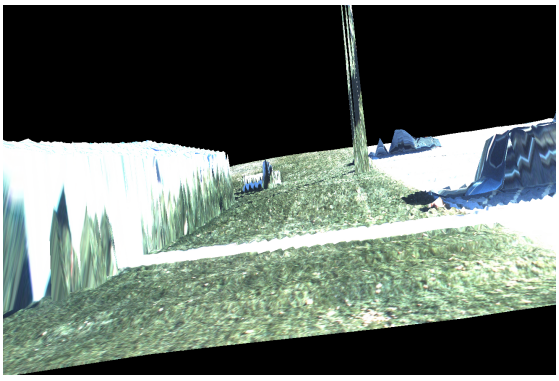
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

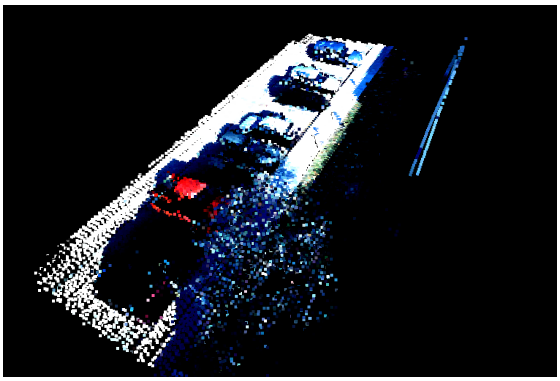


(d) New Method TDSM

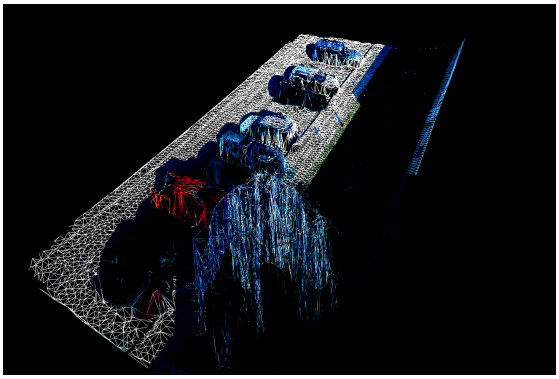


(e) Delaunay TDSM

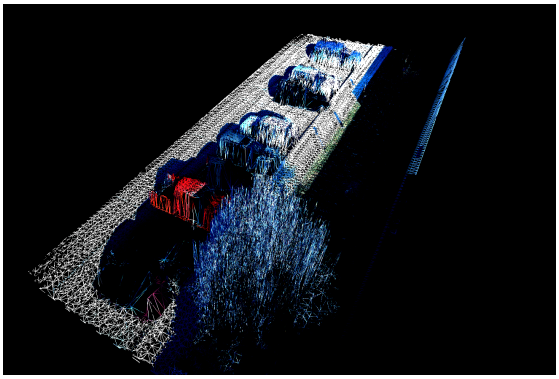
Fig. 4.9: Flight 2 Scene 2 Reconstruction



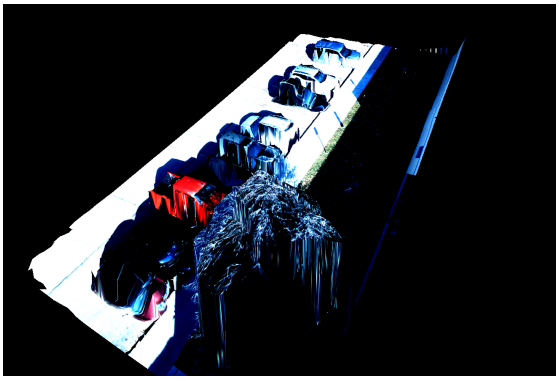
(a) Textured Point Cloud



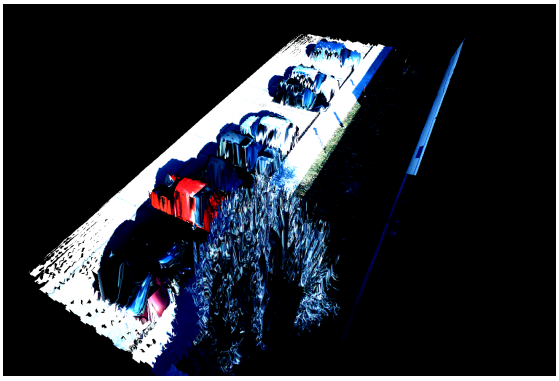
(b) New Method Wire-frame



(c) Delaunay Wire-Frame

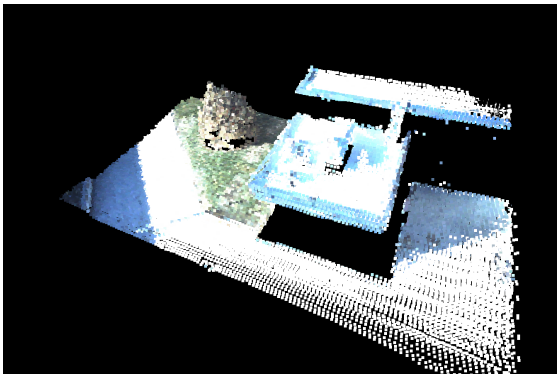


(d) New Method TDSM

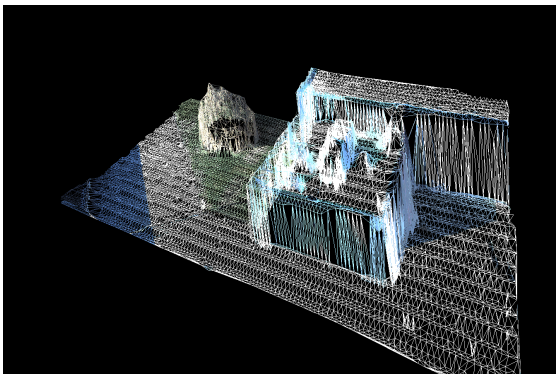


(e) Delaunay TDSM

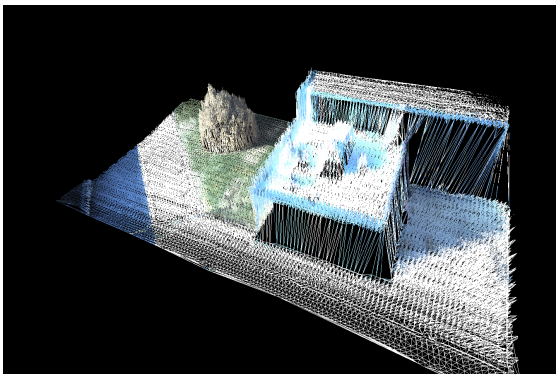
Fig. 4.10: Flight 2 Scene 3 Reconstruction



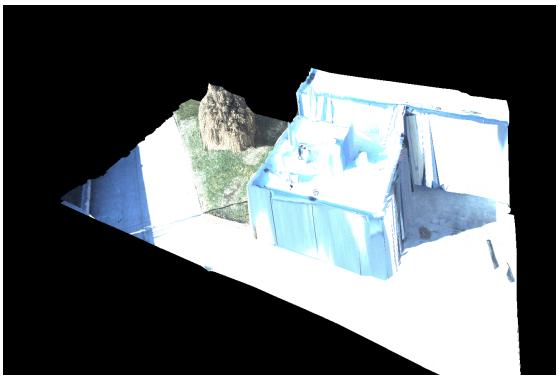
(a) Textured Point Cloud



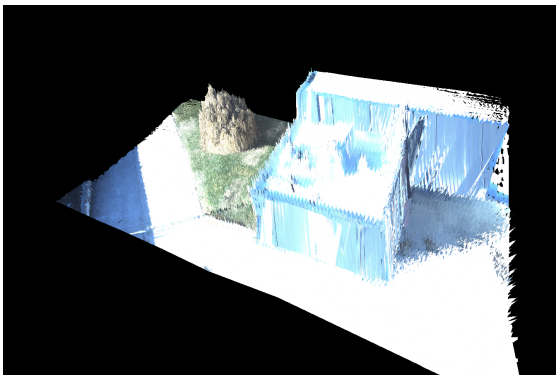
(b) New Method Wire-frame



(c) Delaunay Wire-Frame



(d) New Method TDSM



(e) Delaunay TDSM

Fig. 4.11: Flight 2 Scene 4 Reconstruction

so that

$$\begin{aligned} T_{\text{old}}(n) &= a_1 n + a_2 n \log n + a_3 n \\ &= a_2\, n \log n + (a_1 + a_3) n \\ &= \Theta(n \log n). \end{aligned}$$

This work: Let $n = |\mathcal{P}|$ be the number of points, $c$ the number of camera views, and $v_i$ the number of intersections detected in view $i$. Then:

$$\begin{aligned} T_{\text{clus}}(n) &= b_1 n, && \text{(clustering)} \\ T_{\text{bpa}}(n) &= b_2 n, && \text{(BPA on clusters)} \\ T_{\text{proj}}(n, c) &= b_3 c n, && \text{(projection to } c \text{ views)} \\ T_{\text{swp}}(n, v_i) &= 2 b_4 c (n + v_i) \log n, && \text{(sweepline, } 2c \text{ times)} \\ T_{\text{ray}}(n, c) &= 2 b_5 c n \log n, && \text{(raycasting per view)} \\ T_{\text{Del}}(n, c) &= b_6 c n \log n, && \text{(Delaunay per view)} \end{aligned}$$

Summing all contributions gives

$$\begin{aligned} T_{\text{new}}(n, c, \{v_i\}) = {} & b_1 n + b_2 n + b_3 c n \\ & + 2 b_4 \left( c n + \sum_{i=1}^{c} v_i \right) \log n \\ & + (2 b_5 + b_6)\, c n \log n. \end{aligned}$$

Thus,

$$T_{\text{new}} = \mathcal{O}(n + cn) + \mathcal{O}\left((cn + \textstyle\sum_i v_i) \log n\right).$$

If $\sum_i v_i = \mathcal{O}(cn)$, this simplifies to

$$T_{\text{new}} = \mathcal{O}(cn \log n).$$

In practice, although BPA has a computational complexity of $\mathcal{O}(n)$, the choice of radius has a significant impact on performance. When the registration and point-wise error are low, BPA generally performs better. Appendix A discusses a method for parallelizing BPA.

# CHAPTER 5
# CONCLUSION

## 5.1 Contributions

This thesis presented a new surface-reconstruction pipeline and updates to Streaming Bundle Adjustment (SBA) that refine point selection for Textured Digital Surface Map (TDSM) creation and the texturing process. The approach addresses challenges in Texel Camera data—specifically missing data, non-uniform point density, and concave features.

The effectiveness of the proposed pipeline is demonstrated through visual comparisons across multiple datasets. The results show improved fidelity for low-cost aerial LiDAR data, particularly in scenes with large vertical discontinuities and concave structures.

## 5.2 Future Work

This work employs a modified Density-Based Spatial Clustering (DBSCAN) for clustering with two neighborhood radii, $\varepsilon_{xy}$ (horizontal) and $\varepsilon_z$ (vertical). Although these parameters are estimated, achieving the highest TDSM quality can still require manual tuning. Future work should aim to reduce or eliminate this by adopting self-tuning or parameter-free clustering methods.

Current work uses a fixed parameter $\mathcal{L}$ assuming that points visible in Texel image $i$ are also visible within $[i - \mathcal{L},\, i + \mathcal{L}]$. Future work should make $\mathcal{L}$ *time-varying* and tie it to platform velocity and acquisition settings (Section 2.1). This should give longer look windows during fast motion and shorter windows when hovering or flying slowly, which will improve optimization in the Streaming Bundle Adjustment and the quality of the final TDSM.

Previous work generated disjoint TDSMs at each iteration of SBA. In contrast, this work produces TDSMs with overlap across SBA iterations. Future work could implement

a surface reconstruction strategy that incrementally maintains a single mesh. The strategy could initialize a mesh in the first iteration of SBA, then at each subsequent iteration add newly optimized points, re-triangulate only affected regions, and update textures accordingly.

Another direction is to investigate approximation-based surface reconstruction. While the triangulation-based pipeline in this and prior work produces a mesh from the optimized point cloud, an alternative is to define an implicit function from the optimized points and camera poses, and then extract a triangulation from that implicit representation. Although such an implicit function would not guarantee that the point cloud present in the final TDSM equals the input optimized point cloud, it has benefits. This approach could reduce computation relative to the current work, and there is existing research on updating an implicit field given new points. This aligns with creating an initial mesh and then updating it with each iteration of SBA.

# REFERENCES


[1] Z. Huang, Y. Wen, Z. Wang, J. Ren, and K. Jia, "Surface reconstruction from point clouds: A survey and a benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, pp. 9727–9748, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:248525170

[2] Q. Yang, L. Wang, R. Yang, H. Stewénius, and D. Nistér, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 492–504, 2009.

[3] F. Isaksson, J. Borg, and L. Haglund, "3D rapid mapping," in *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications V*, D. J. Henry, Ed., vol. 6946, International Society for Optics and Photonics. SPIE, 2008, p. 69460D. [Online]. Available: https://doi.org/10.1117/12.777616

[4] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed. Berlin, Heidelberg: Springer-Verlag, 2000. [Online]. Available: http://www.cs.uu.nl/geobook/

[6] C. Gomez, Y. Hayakawa, and H. Obanawa, "A study of japanese landscapes using structure from motion derived dsms and dems based on historical aerial photographs: New opportunities for vegetation monitoring and diachronic geomorphology," *Geomorphology*, vol. 242, pp. 11–20, 2015, geomorphology in the Geocomputing Landscape: GIS, DEMs, Spatial Analysis and statistics.

[7] K. Anderson and K. J. Gaston, "Lightweight unmanned aerial vehicles will revolutionize spatial ecology," *Frontiers in Ecology and the Environment*, vol. 11, no. 3, pp. 138–146, 2013.

[8] K. Lim, P. Treitz, M. Wulder, B. St-Onge, and M. Flood, "Lidar remote sensing of forest structure," *Progress in Physical Geography*, vol. 27, no. 1, pp. 88–106, 2003.

[9] M. K. Jain and V. P. Singh, "Dem-based modelling of surface runoff using diffusion wave equation," *Journal of Hydrology*, vol. 302, no. 1, pp. 107–126, 2005.

[10] M. Bisson, B. Behncke, A. Fornaciai, and M. Neri, "Lidar-based digital terrain analysis of an area exposed to the risk of lava flow invasion: the zafferana etnea territory, mt. etna (italy)," *Natural Hazards*, vol. 50, no. 2, pp. 321–334, Aug 2009.

[11] K. Latham, W. Hurst, N. Shone, A. El Rhalibi, and Z. Pan, "A case study on the advantages of 3d walkthroughs over photo stitching techniques," in *2016 International Conference on Virtual Reality and Visualization (ICVRV)*, 2016, pp. 364–371.

[12] J. Wang, S. Wang, D. Zou, H. Chen, R. Zhong, H. Li, W. Zhou, and K. Yan, "Social network and bibliometric analysis of unmanned aerial vehicle remote sensing applications from 2010 to 2021," *Remote Sensing*, vol. 13, no. 15, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/15/2912

[13] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[14] R. Hartley and A. Zisserman, *3D Reconstruction of Cameras and Structure*. Cambridge University Press, 2004, p. 262–278.

[15] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver, "Google street view: Capturing the world at street level," *Computer*, vol. 43, no. 6, pp. 32–38, 2010.

[16] Y. Wang, V. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon, "Detr3d: 3d object detection from multi-view images via 3d-to-2d queries," 2021. [Online]. Available: https://arxiv.org/abs/2110.06922

[17] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 420–425.

[18] A. Thakur and P. Rajalakshmi, "Lidar and camera raw data sensor fusion in real-time for obstacle detection," in *2023 IEEE Sensors Applications Symposium (SAS)*, 2023, pp. 1–6.

[19] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, 2020.

[20] A. M. d. S. Neto and A. F. Roseli Romero, "Sensor fusion of lidar and stereo camera for forest mapping," in *2024 Latin American Robotics Symposium (LARS)*, 2024, pp. 1–6.

[21] T. Bybee and S. Budge, "Method for 3-d scene reconstruction using fused lidar and imagery from a texel camera," *IEEE Transactions on Geoscience and Remote Sensing*, vol. PP, pp. 1–11, Jul 2019.

[22] B. Khatiwada, "Super-resolution textured digital surface map (dsm) formation super-resolution textured digital surface map (dsm) formation by selecting the texture from multiple perspective texel images taken by a low-cost small unmanned aerial vehicle (uav)," Ph.D. dissertation, Utah State University, 2023.

[23] T. Bybee and S. Budge, "Method for 3-D scene reconstruction using fused lidar and imagery from a texel camera," *IEEE Transactions on Geoscience and Remote Sensing*, vol. PP, pp. 1–11, 07 2019.

[24] S. E. Budge and N. S. Badamikar, "Calibration method for texel images created from fused flash lidar and digital camera images," *Optical Engineering*, vol. 52, no. 10, p. 103101, 2013. [Online]. Available: https://doi.org/10.1117/1.OE.52.10.103101

[25] T. Welch, C. Coopmans, and S. Budge, "Development of a small unmanned aerial system-mounted texel camera," Utah State University, Tech. Rep., May 2019.

[26] B. M. Boldt, S. E. Budge, R. T. Pack, and P. D. Israelsen, "A handheld texel camera for acquiring near-instantaneous 3d images," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, 2007, pp. 953–957.

[27] N. I. Jensen, "Gps-denied navigation using location estimation and texel image correction," Master's thesis, Utah State University, 2024, all Graduate Theses and Dissertations, Fall 2023 to Present, 213. [Online]. Available: https://digitalcommons.usu.edu/etd2023/213

[28] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, 1985, pp. 245–254.

[29] D. M. Henderson, "Shuttle program: Euler angles, quaternions, and transformation matrices working relationships," NASA Johnson Space Center, Technical Report JSC-12960, 1977.

[30] B. Khatiwada and S. E. Budge, "Three-dimensional image reconstruction using bundle adjustment applied to multiple texel images," in *Laser Radar Technology and Applications XXI*, ser. Proceedings of SPIE, M. D. Turner and G. W. Kamerman, Eds., vol. 9832. International Society for Optics and Photonics, 2016, p. 98320S, online.

[31] T. C. Bybee, "An automatic algorithm for textured digital elevation model formation using aerial texel swaths," Master's thesis, Utah State University, 2016. [Online]. Available: https://digitalcommons.usu.edu/etd/4593

[32] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, ser. Lecture Notes in Computer Science, vol. 3951. Springer, 2006, pp. 404–417.

[33] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[34] R. M. Bolle and B. C. Vemuri, "On three-dimensional surface reconstruction methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, pp. 1–13, 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:206418929

[35] S. P. Lim and H. Haron, "Surface reconstruction techniques: a review," *Artificial Intelligence Review*, vol. 42, pp. 59 – 78, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:1695942

[36] 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, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:5902682

[37] C. C. You, S. P. Lim, S. C. Lim, J. S. Tan, C. K. Lee, and Y.-M. J. Khaw, "A survey on surface reconstruction techniques for structured and unstructured data," *2020 IEEE Conference on Open Systems (ICOS)*, pp. 37–42, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229374456

[38] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 551–559, 1983.

[39] T. K. F. Da and D. Cohen-Steiner, "Advancing front surface reconstruction," in *CGAL User and Reference Manual*, 6.0.2 ed. CGAL Editorial Board, 2025. [Online]. Available: https://doc.cgal.org/6.0.2/Manual/packages.html#PkgAdvancingFrontSurfaceReconstruction

[40] F. Hurtado, M. Noy, and J. Urrutia, "Flipping edges in triangulations," *Discrete & Computational Geometry*, vol. 22, no. 3, pp. 333–346, 1999. [Online]. Available: https://doi.org/10.1007/PL00009464

[41] L. J. Guibas, D. E. Knuth, and M. Sharir, "Randomized incremental construction of delaunay and voronoi diagrams," *Algorithmica*, vol. 7, no. 1, pp. 381–413, 1992. [Online]. Available: https://doi.org/10.1007/BF01758770

[42] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi," *ACM Trans. Graph.*, vol. 4, no. 2, p. 74–123, Apr. 1985. [Online]. Available: https://doi.org/10.1145/282918.282923

[43] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1, pp. 153–174, 1987. [Online]. Available: https://doi.org/10.1007/BF01840357

[44] J. Ruppert, "A delaunay refinement algorithm for quality 2-dimensional mesh generation," *Journal of Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.

[45] L. P. Chew, "Guaranteed-quality mesh generation for curved surfaces," in *Proceedings of the Ninth Annual Symposium on Computational Geometry*, ser. SCG '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 274–280. [Online]. Available: https://doi.org/10.1145/160985.161150

[46] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, May 1996, vol. 1148, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.

[47] ——, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry*, vol. 22, no. 1, pp. 21–74, 2002, 16th ACM Symposium on Computational Geometry. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772101000475

[48] S. Loriot, J. Tournois, and I. O. Yaz, "Polygon mesh processing," in *CGAL User and Reference Manual*, 4.11.3 ed. CGAL Editorial Board, 2018. [Online]. Available: http://doc.cgal.org/4.11.3/Manual/packages.html#PkgPolygonMeshProcessingSummary

[49] M. Soucy and D. Laurendeau, "A general surface approach to the integration of a set of range views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 4, pp. 344–358, 1995.

[50] S. Hert and M. Seel, "dD convex hulls and delaunay triangulations," in *CGAL User and Reference Manual*, 6.0.1 ed. CGAL Editorial Board, 2024. [Online]. Available: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgConvexHullD

[51] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[52] J. Digne, J.-M. Morel, C. Lartigue, T. Lahouze, F. Dupont, and R. Chaine, "An analysis and implementation of a parallel ball pivoting algorithm," *Image Processing On Line*, vol. 4, pp. 149–168, 2014.

[53] H. Saffi, A. Ben-Hamadou, and H. Ben-Ahmed, "Auto-bpa: An enhanced ball-pivoting algorithm with adaptive radius using contextual bandits," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 1234–1243.

[54] A. Prithiviraj, R. Santhosh, A. Vignesh, and S. Sujatha, "Optimization of mesh reconstruction using delaunay and ball pivoting algorithm," *International Journal of Engineering Research and Technology*, vol. 12, no. 11, pp. 1–7, 2023.

[55] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," ser. SIGGRAPH '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 71–78. [Online]. Available: https://doi.org/10.1145/133994.134011

[56] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, 2006, pp. 61–70.

[57] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge Discovery and Data Mining*, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:355163

# APPENDICES

# APPENDIX A

## Threading BPA

The Ball Pivoting Algorithm (BPA) is $O(n)$ in computational complexity, but in practice, its runtime depends heavily on the dataset and on the chosen radii. In data generated by the Texel Camera (see Section 2.1), registration errors from inaccurate Inertial Navigation System (INS) poses can produce multi-layered surfaces (e.g., the ground appears several 3D points thick).

The radius-estimation method described in Chapter 3 helps the ball roll on the outer surface, but it also means BPA may need to try many candidate seed triangles before finding a valid starting position and then execute many pivots to traverse the surface, increasing runtime.

To address this issue, the C++ implementation threads BPA. The first level of threading operates at the *cluster* level: as shown in Fig. 3.1, the input point cloud is clustered prior to BPA, so clusters are independent. BPA runs on each cluster concurrently (one thread per cluster), with no inter-cluster synchronization required.

The implementation adds a second level of parallelism inside each cluster. It first computes the mean cluster size $\bar{N}$ (in points) across the scene. Any cluster with size $N > 2\bar{N}$ is subdivided into spatial *grid chunks* in the $xy$ plane. The grid resolution $(n_x, n_y)$ is estimated so that each chunk contains approximately $\bar{N}$ points, i.e., $n_x n_y \approx \lceil N/\bar{N} \rceil$ (chosen to respect the cluster's $xy$ aspect ratio). BPA then runs concurrently on all chunks from that oversized cluster using the same radius set. After the first pass finishes, the method unites the chunk point clouds and performs a *second* BPA pass on the combined points to stitch together chunks.

This strategy accelerates meshing because the first pass of BPA creates a triangulation that doesn't use many points that arise from acquisition/registration error (e.g., duplicate layers from pose drift). With fewer misleading candidates, BPA finds seed triangles faster

and executes fewer pivots. The second pass then operates on a less noisy set of surface points, so the stitching the chunks together is relatively inexpensive.

# APPENDIX B

## Sweepline Algorithm

The *Sweepline Algorithm* (also known as the *Plane Sweep Algorithm*) is a computational geometry technique that reduces the complexity of problems by processing data in a predefined geometric order. Sweepline has been used for detecting line-segment intersections, constructing Voronoi diagrams (and thus Delaunay triangulations), and performing Boolean operations on polygons.

The implementation presented here follows a similar approach to Berg et al. [5]. The algorithm conceptually uses a horizontal line that moves top-down through the plane. This line, called the *sweepline*, advances discretely from one *event point* to the next. An event point is either a segment endpoint or an intersection point.

Two data structures are maintained throughout the algorithm:

- A **priority queue** $Q$ that stores all event points, ordered by decreasing $y$-coordinate then by increasing $x$-coordinate.
- A **balanced binary search tree** $T$ that stores the set of segments currently intersecting the sweepline. These are ordered from left to right according to their x-position, given the y-position of the current sweepline. If two segments share a point at the given y-position, then they are sorted by angle from the positive x-axis. In other words, whichever segment points more to the left after the sweepline is ordered first.

As the algorithm processes each event point $p$, it performs $\mathcal{O}(\log n)$ operations per insertion, deletion, or find in $T$, where $n$ is the number of active segments. Accessing the first element in $Q$ is constant in time, and adding new events (intersections) is $\mathcal{O}(\log n)$ where $n$ is the number of active events. Since the total number of events is $\mathcal{O}(n + k)$, the overall time complexity is $\mathcal{O}((n + k) \log n)$. The full algorithm is outlined in Algorithm 4.

**Algorithm 4:** Segment Intersection via Sweepline

**input :** Set of line segments $\mathcal{S}$
**output:** Set of intersection points $\mathcal{I}$, Set of intersecting segments $\mathcal{X}$
Let $L(p)$ be the set of segments for which $p$ is the lower endpoint;
Let $U(p)$ be the set of segments for which $p$ is the upper endpoint;
Let $C(p)$ be the set of segments for which $p$ is an interior point (initially unknown);
Initialize the event queue $Q$ with all unique segment endpoints, each associated with its $L(p)$ and $U(p)$;
Initialize the balanced binary search tree $T \leftarrow \varnothing$;
Initialize $\mathcal{I} \leftarrow \varnothing$;
Initialize $\mathcal{X} \leftarrow \varnothing$;
**while** $Q$ *not empty* **do**
    Pop the next event point $p$ from $Q$;
    Insert $L(p) \cup U(p) \cup C(p)$ into $\mathcal{X}$ and insert $p$ into $\mathcal{I}$;
    Remove all segments in $L(p) \cup C(p)$ from $T$;
    Update the sweepline position to $p$;
    Insert all segments in $U(p) \cup C(p)$ into $T$;
    **if** $U(p) \cup C(p) \neq \varnothing$ **then**
        Let $s_{\mathrm{p}}$ be the leftmost segment in $U(p) \cup C(p)$;
        Let $s_\ell$ be the left neighbor of $s_{\mathrm{p}}$ in $T$;
        **if** *a valid intersection between* $s_{\mathrm{p}}$ *and* $s_\ell$ *occurs below or to the right of* $p$ **then**
            Insert the intersection point as an event into $Q$;
        **end**
        Let $s_{\mathrm{pp}}$ be the rightmost segment in $U(p) \cup C(p)$;
        Let $s_r$ be the right neighbor of $s_{\mathrm{pp}}$ in $T$;
        **if** *a valid intersection between* $s_{\mathrm{pp}}$ *and* $s_r$ *occurs below or to the right of* $p$ **then**
            Insert the intersection point as an event into $Q$;
        **end**
    **else**
        Let $s_\ell$ and $s_r$ be the left and right neighbors of the gap at $p$ in $T$;
        **if** *a valid intersection between* $s_\ell$ *and* $s_r$ *occurs below or to the right of* $p$ **then**
            Insert the intersection point as an event into $Q$;
        **end**
    **end**
**end**

# APPENDIX C

## Back-projecting a 2D intersection to 3D along segment correspondences

Consider two intersecting image-plane segments

$$s_0 = (u_{i_0},\, u_{j_0}), \qquad s_1 = (u_{i_1},\, u_{j_1}),$$

where each endpoint $u_k \in \mathbb{R}^2$ represents the normalized image-plane projection of a 3D world point $X_k \in \mathbb{R}^3$. Let $C \in \mathbb{R}^3$ denote the camera center in world coordinates, and let $u_* \in \mathbb{R}^2$ denote the intersection point of the two 2D segments.

Each image-plane segment is parameterized linearly, and the scalar parameter $t$ that places the intersection along the segment is obtained. For segment $s_0$,

$$t_0 \;=\; \frac{(u_{j_0} - u_{i_0})^\top (u_* - u_{i_0})}{\| u_{j_0} - u_{i_0} \|_2^2}, \qquad t_1 \;=\; \frac{(u_{j_1} - u_{i_1})^\top (u_* - u_{i_1})}{\| u_{j_1} - u_{i_1} \|_2^2},$$

with $t_0, t_1$ clamped to $[0, 1]$ to enforce on-segment intersections.

The same linear parameters interpolate along the corresponding 3D world segments:

$$P_0 \;=\; X_{i_0} \;+\; t_0\,(X_{j_0} - X_{i_0}), \qquad P_1 \;=\; X_{i_1} \;+\; t_1\,(X_{j_1} - X_{i_1}),$$

which yields two candidate 3D points projecting to $u_*$ along the respective segment pairs.

The squared Euclidean distances of these points to the camera center are then computed:

$$d_0 \;=\; \|P_0 - C\|_2^2, \qquad d_1 \;=\; \|P_1 - C\|_2^2.$$

The point with the smaller value, $\min\{d_0, d_1\}$, identifies the visible intersection along the viewing ray through $u_*$, while the other lies occluded.

CURRICULUM VITAE

# Samuel Kiguthi

## Published Journal Articles

- 50 kW Reflexive Tuning Networks With Low Uncoupled Transmitter Currents for Dynamic Inductive Power Transfer Systems, Shuntaro Inoue, Samuel Kiguthi, Jonathan Newman, Timothy Goodale, Chakradhar Reddy Teeneti, and Bryce Hesterman, *IEEE Open Journal of Power Electronics*, vol. 5, pp. 1–10, 2024.

## Published Conference Papers

- Improving texturing techniques for fused LiDAR images, Samuel Kiguthi and Scott E. Budge, in *Proc. SPIE Defense + Commercial Sensing, Laser Radar Technology and Applications XXX*, vol. 13472, 134720J, Orlando, Florida, United States, May 2025. DOI: 10.1117/12.3053831.