MSc thesis in Geomatics

Selective image region focus for efficient 3D building reconstruction using SAM in oblique aerial imagery

L.C. Boertjes

2025

TU Delft GEODELTA

MSc thesis in Geomatics

Selective image region focus for efficient 3D building reconstruction using SAM in oblique aerial imagery

Lars Boertjes

June 2025

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

Lars Boertjes: *Selective image region focus for efficient 3D building reconstruction using SAM in oblique aerial imagery* (2025)

Supervisors:	Azarakhsh Rafiee
	Ken Arroyo Ohori
	Annemieke Verbraeck
Co-reader:	Weixiao Gao

Abstract

High-quality 3D mesh models are increasingly used in urban applications ranging from planning and simulation to environmental monitoring. While aerial imagery provides a practical balance between resolution and coverage, conventional Structure from Motion (SfM) and Multi-View Stereo (MVS) pipelines apply uniform processing to all image regions, often overlooking their geometric relevance. This research investigates whether reconstruction efficiency can be improved by selectively focusing dense matching efforts on image regions most critical to mesh quality.

We explore the use of the Segment Anything Model (SAM) to guide dense matching in oblique aerial imagery, applying it for building-level segmentation and importance estimation via entropy and edge-based scoring. Initially, we tested SAM for sparse reconstruction guidance but encountered poor performance. We then shifted to evaluating whether SAM-derived importance maps could enable region-aware thresholding to improve mesh reconstruction. These methods were benchmarked against a simpler Canny edge-based distance approach across varying thresholds and scenes.

Results show that SAM-based methods can reduce memory usage while maintaining mesh quality at low to moderate thresholds (up to 0.4). However, contrary to expectations, Canny edge detection consistently outperformed SAM across most quality and efficiency metrics, offering better spatial coverage, lower computational overhead, and more stable performance. While SAM-based thresholding led to file size reductions of up to 16% and marginal runtime gains during dense matching (2.3%), these were offset by the costly preprocessing pipeline required to generate segmentation masks and importance maps.

Overall, this thesis contributes an evaluation pipeline for image region-specific reconstruction strategies and highlights that while SAM shows potential for memory-efficient modeling, simpler methods like Canny edge detection may offer better trade-offs for scalable, time-sensitive workflows. Future research should focus on faster importance estimation techniques and pipeline adaptation for more complex, city-scale datasets.

Acknowledgements

Thanks to everyone involved in supporting me with this thesis. Special thanks to Azarakhsh for meeting with me long before we even started the thesis (and for the Omani sweets). Ken, for waking up early to hear my weekly updates on Friday mornings. And to Annemieke, for pointing out that the results don't go in the discussion section. Of course, also a big thank you to all the Geodelta employees, who were incredibly approachable and created a great place for me to work. Finally, thanks to Esra for all the support throughout the year and for encouraging me to pick up my master's again two years ago. And of course, the front page :)

Contents

Acronyms

1. Introduction

1.1. Motivation

Accurate 3D mesh models of urban environments are used in a wide range of geomatics related applications, such as urban planning, disaster response, environmental monitoring, and immersive virtual simulations [Biljecki et al., 2015]. They can be derived from various image data sources, such as satellite imagery, aerial imagery and mobile mapping stations [Remondino and El-Hakim, 2006].

Among the available data sources, aerial imagery offers a promising balance between resolution and coverage. It typically provides higher spatial resolution than satellite imagery and is capable of covering larger areas more efficiently than ground-based methods such as terrestrial laser scanning or mobile mapping [Nex and Remondino, 2014; Fraser and Cronk, 2009].

3D mesh models from aerial images are typically generated through Structure from Motion (SfM) and Multi-view stereo (MVS) pipelines, which estimate camera poses and reconstruct dense point clouds from overlapping images [Remondino and El-Hakim, 2006]. Recently, deep learning-based approaches have also been introduced to these pipelines, particularly in dense matching and depth estimation [Liu et al., 2023; Yao et al., 2018].

However, converting aerial imagery into high-quality 3D meshes remains computationally intensive. These pipelines tend to distribute computational resources uniformly across the entire image, regardless of whether a region contains meaningful structural detail. For example, flat surfaces such as walls or roofs may require fewer vertices for mesh creation, whereas architectural features like windows, balconies, or other ornaments demand greater point density to make an effective mesh.

This research investigates whether the creation of 3D meshes from oblique aerial imagery can be made more efficient by optimizing the dense matching process, by focusing computational resources on parts of buildings that are important for mesh quality.

To achieve this, we explore the potential of SAM, the Segment Anything Model developed by Meta AI [Kirillov et al., 2023], as a tool for guiding adaptive point cloud densification. SAM is a general-purpose segmentation model capable of producing high-quality segmentation masks. In this study, we experiment with SAM in two ways:

- Segmenting buildings from oblique aerial images: we use SAM to extract instance-specific building masks from an oblique image dataset, enabling object-level reconstruction.
- Zooming in on each segmented building: We analyze each building mask to identify geometrically complex surfaces or features that contribute more to overall mesh quality.

By combining SAM-based segmentation with an entropy metric and an edge-based decay function, we aim to identify the most important parts in the images for mesh reconstruction. This approach is explorative and built on the hypothesis that SAM favors visually and structurally salient features. While not guaranteed, this bias aligns with our aim of targeting areas of high geometric complexity.

We apply implicit surface reconstruction techniques to convert the adaptively densified point clouds into watertight 3D meshes, making them suitable for applications in urban planning, simulation, and visualization. These meshes are then compared to reference meshes generated using traditional dense matching pipelines without resource allocation. Ultimately, our goal is to maintain comparable mesh quality while significantly reducing both the computational cost of the reconstruction process and the size of the output meshes.

1.2. Geodelta

This research is conducted in collaboration with Geodelta, an engineering and consulting firm specializing in geoinformation. Geodelta provides technical advice and software solutions. Their office is located in the historic Geodesy building, close to the Architecture faculty. For more information, visit their website at geodelta.com.

1.3. Research objectives

1.3.1. Objectives

The main research question for this thesis is:

How can SAM-based building segmentation and importance estimation in oblique aerial imagery be used to improve the efficiency of 3D mesh reconstruction by selectively focusing on important image regions for reconstruction quality?

The goal of this research is to explore the potential of SAM for 3D mesh reconstruction from oblique aerial imagery, while optimizing computational resources by focusing reconstruction efforts on the most important image regions. To achieve this, the following sub-questions will guide the investigation:

- How can SAM be applied to accurately and efficiently segment individual buildings in oblique aerial imagery?
- How can SAM be used to identify image regions that are most important for achieving high-quality 3D mesh reconstruction?
- How can the effectiveness of image region importance driven resource allocation be evaluated in terms of mesh quality, and computational efficiency?
- How does selective masking of images based on importance threshold affect quality and efficiency of 3D mesh reconstruction across different building datasets?

1.3.2. Scope of research

The main focus of this research is on producing a mesh of comparable quality to one generated through dense reconstruction.

This research aims to develop a reconstruction algorithm using segmentation at two levels:

1. At the oblique image level
2. At the individual building level

For the oblique image level, we used SAM to extract building masks from full-scene images in the area of interest described in Section 3.2.

Initially, our goal was to reconstruct all buildings individually within that area and apply our selective image region-based mesh reconstruction method to a few of them. However, as shown in Section 5.3, reconstruction based solely on building masks produced poor results.

As a result, the focus of the second research phase shifted to the Geodelta drone dataset, where we could test importance-based thresholding for dense matching at the building level. This adjustment was also necessary due to the high computational cost and time required to evaluate many dense matching configurations.

2. Related work

This chapter reviews prior work in four areas relevant to our pipeline: 3D reconstruction from aerial images, segmentation methods, entropy-driven geometry processing, and surface modeling approaches. We also reflect on how some early directions, such as point cloud simplification, shaped our understanding, even if we ultimately did not end up using them for the core pipeline.

2.1. 3D Reconstruction from oblique aerial imagery

Recently, several frameworks have emerged to address the unique challenges of 3D reconstruction from oblique aerial imagery, which often suffers from occlusions, scale variability, and inconsistent features due to varying viewpoints.

Ada-MVS [Liu et al., 2023] is a deep multi-view stereo framework designed to adaptively aggregate view weights, mitigating issues typical in oblique images. It leverages spatial pyramid pooling and a memory-efficient ConvGRU module, enabling accurate point cloud generation without the computational overhead of full 3D convolutions.

Building upon this, Liu et al. [Liu and Ji, 2020] introduced a recurrent encoder-decoder architecture tailored for aerial images with wide depth ranges. They also proposed the WHU dataset, a large-scale aerial dataset containing diverse oblique views, specifically curated to facilitate research in multi-view stereo for aerial applications. Although efficient, this method is outperformed by Ada-MVS in recent evaluations.

Other approaches, like [Yu et al., 2021], focus on symbolic 3D reconstruction by generating simplified building models, typically box, shaped footprints combined with height data. While these models are scalable and computationally lighter, they lack detailed surface geometry and are less suitable for applications requiring mesh-level precision.

Most methods rely on the full set of images for reconstruction, reflecting the need to integrate multiple viewpoints for robust 3D modeling.

2.2. Segmentation in oblique aerial images

Segmenting visual content into meaningful regions has long been a key preprocessing step in computer vision, supporting applications ranging from object detection to scene understanding. Before the rise of deep learning, segmentation typically relied on low-level cues such as intensity, color, edge, continuity, or region homogeneity. Common classical techniques include thresholding methods like Otsu's algorithm [Otsu, 1979], edge-based segmentation

using gradient operators, and region-growing methods [Adams and Bischof, 1994]. Clustering techniques such as k-means and mean shift [Comaniciu and Meer, 2002] have also been widely used, particularly for segmenting images into regions of similar texture and color.

Modern AI-based techniques, using convolutional neural networks (CNNs), support semantic, instance, and panoptic segmentation [Long et al., 2015; He et al., 2017; Kirillov et al., 2019]. These methods typically require large labeled datasets and extensive training, and their performance can degrade when tested on domains outside their training distribution.

In our work, we explore the recently introduced Segment Anything Model (SAM) [Kirillov et al., 2023], which promises supporting zero-shot segmentation, a paradigm where the model is capable of segmenting new objects or scenes without having seen them during training. SAM achieves this by using a large-scale, promptable segmentation model trained on over 1 billion masks (SA-1B dataset).

Although SAM was not explicitly trained on oblique aerial imagery, we found it promising enough to integrate into our pipeline. Specifically, we use SAM not for classical segmentation, but rather to assist in identifying image regions that are most relevant for 3D mesh reconstruction.

2.3. Geometric entropy

Techniques such as vertex curvature entropy [Xing and Hui, 2013] and related point cloud simplification methods [Shi et al., 2022] quantify local geometric significance in point clouds and meshes, typically to drive simplification by preserving important features. Our objective, however, is to invert this approach: rather than reducing existing data, we use entropy to guide the selective generation of data from the outset. Similarly, feature-aware terrain filtering [Yu et al., 2021] prioritizes LiDAR points based on curvature and elevation discontinuities. Although developed for digital terrain models (DTMs), this principle of focusing reconstruction effort on high-information regions aligns conceptually with our method.

Initially, we aimed to estimate entropy directly on sparse point clouds derived from segmentation masks and add points adaptively in geometrically significant regions. However, this approach introduced several challenges (discussed in more detail in Section 3.4): First, the need for a pre-existing dense point cloud contradicted our goal of minimizing computational cost. Secondly, the quality of entropy estimates was coupled to the quality of the initial sparse point cloud. Finally, this pipeline excluded SAM, which we sought to explore for entropy detection. As a result, we shifted to an image-based strategy that allows entropy estimation without requiring dense point cloud reconstruction upfront. We seek this entropy using the following works.

The first draws on the notion of entropy from information theory, where it quantifies the uncertainty or unpredictability in a system [Shannon, 1948]. We apply this to detecting the uncertainty of SAM masks appearing within a local kernel, which we convolve over the images.

The second method relies on edge information extracted using the Canny edge detector [Canny, 1986]. This algorithm identifies significant intensity transitions in an image through a multi-stage process involving noise reduction, gradient calculation, edge thinning, and threshold-based edge linking.

We leverage these edge maps to estimate geometric entropy by computing local edge density. This measure guides image filtering in dense matching, focusing computational resources on regions of high geometric complexity relevant for accurate reconstruction.

2.4. Implicit vs. explicit surface modeling

In 3D reconstruction, surface modeling approaches are broadly divided into explicit and implicit methods, based on how the surface geometry is represented and computed. Explicit surface modeling directly encodes geometry using discrete elements such as vertices, edges, and faces [Bloomenthal et al., 1997]. Examples of explicit mesh reconstruction include Delaunay triangulation, which constructs meshes by connecting points in a way that maximizes minimum angles, or mesh reconstruction methods that directly operate on point clouds. Often marching cubes [Lorensen and Cline, 1987] is used on implicit mesh representations, to extract an explicit mesh from it, in order to visualize it.

Implicit surface modeling, on the other hand, represents surfaces as level sets of continuous functions, such as signed distance fields or indicator functions. These models allow for smooth, watertight surface generation and are often preferred in learning-based and volumetric approaches.

An example of implicit method is Poisson surface reconstruction [Kazhdan et al., 2006], which is also the technique we use in our pipeline. It reconstructs a surface by solving a Poisson equation to compute an indicator function over space, from which the surface is extracted as an isosurface (usually at the 0.5 threshold). This method is known for its robustness to noise and ability to produce smooth surfaces, though it may oversmooth sharp features.

Other modern learning-based implicit methods include ImpliCity [Stucker et al., 2022], which uses neural networks to predict occupancy fields from stereo imagery, and DeepMesh [Guillard et al., 2022], which reformulates the surface extraction step, normally a fixed, non-trainable process, into a differentiable operation. This means the step that converts learned volumetric data into a mesh, such as extracting an isosurface, is redesigned so that its parameters can be adjusted during training using gradients. As a result, the entire pipeline, from raw image input to final mesh output, can be optimized.

Our approach uses Poisson reconstruction, but focuses its innovation earlier in the pipeline. Rather than modifying the reconstruction algorithm itself, we attempt to improve efficiency by filtering out the important regions in the input imagery, so that only the most informative views are processed. This entropy-guided thresholding reduces unnecessary computation while still preserving mesh reconstruction quality.

3. Methodology

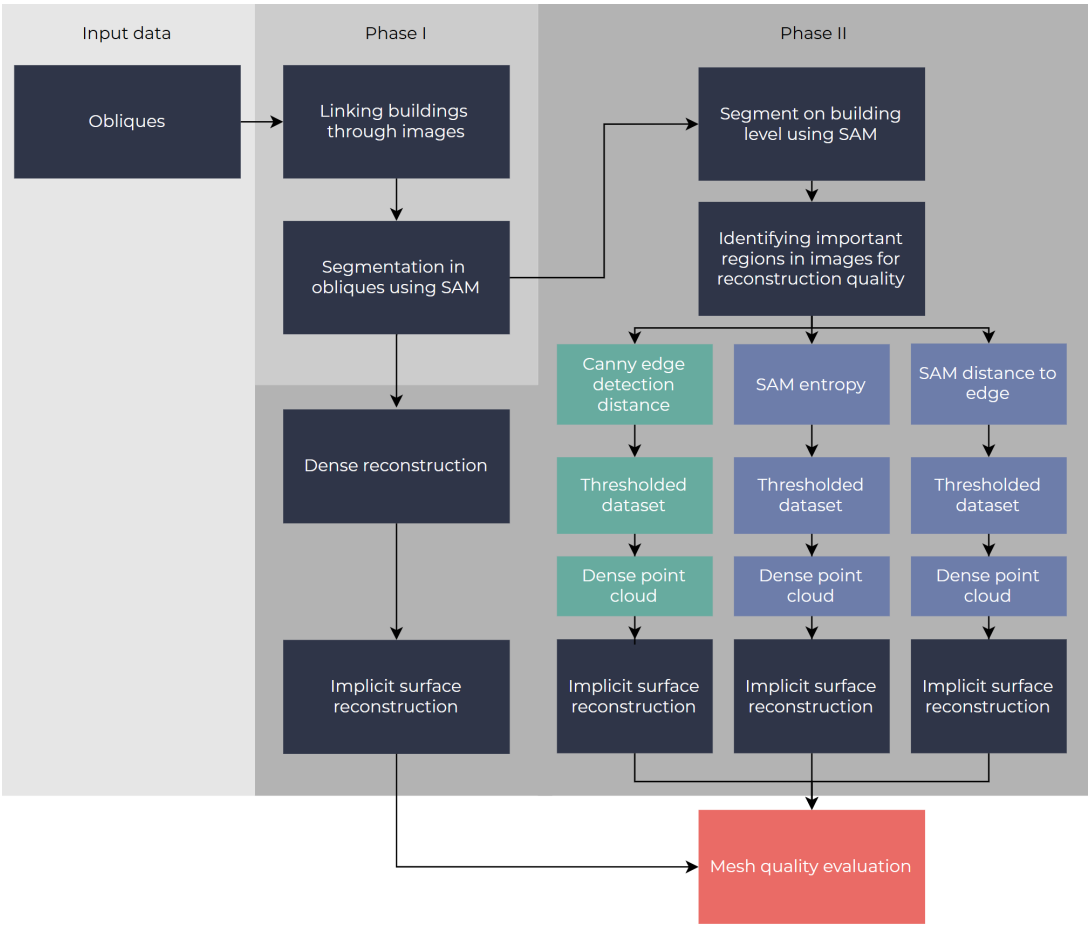


Figure 3.1.: Flowchart of the methodology

Figure 3.1 presents a flowchart of the proposed process, which will be explained in more detail in the following sections. The workflow is divided into two main phases: Phase I involves segmenting each building within the Area of Interest (AOI) across all oblique images in which it is visible. Phase II focuses on generating a mesh for each building using only those points deemed important for mesh reconstruction quality. Ideally, this process uses the segmentation masks obtained in Phase I. However, as discussed in Chapter 4, the nature of the segmentation masks made accurate reconstruction difficult. Therefore, Phase II was also evaluated on an additional drone dataset, better suited for this part of the research.

We will also discuss how our methodology changed throughout the research. In phase II,

the focus is on adding points to a mesh that are beneficial for mesh reconstruction quality. Initially, we aimed to build a sparse point cloud from the segmentation masks, then iteratively add points in areas of high geometric entropy, derived through point cloud analysis. Eventually, we shifted towards an approach that identifies the most important regions in an image for dense reconstruction.

3.1. Input Data

The main input for Phase I (segmentation in oblique images) consists of aerial imagery from the city of Utrecht, including only oblique views. The AOI is shown in Figure 3.3a. For the segmentation step at image level, we focus only on the oblique views. Later, we will also apply our methods to the nadir datasets to evaluate how scalable our approach is. All spatial data is referenced in the Amersfoort / RD New coordinate system (EPSG:28992). Additional inputs include flight plans and camera metadata, such as interior and exterior orientation parameters.

For Phase II, we used a separate drone dataset, comprising 200 unreferenced images of the Geodelta office. Unlike the oblique imagery, these images offer closer views, increased overlap, and no available orientation parameters. Importantly, the segmentation was performed manually rather than using SAM. The nature of this dataset was much more suited for experimentation and evaluation in Phase II.

3.2. Linking buildings through images

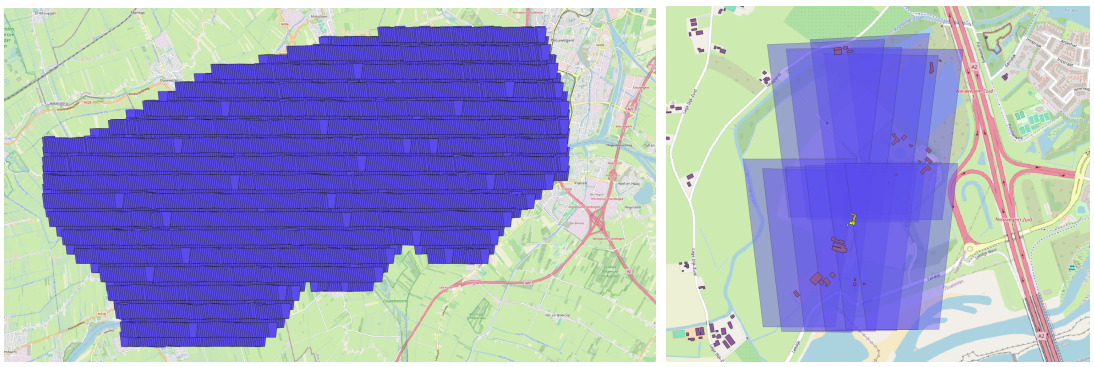
To reconstruct a mesh of an individual building, we must first isolate it in all the oblique images in which it appears. Our initial objective was to do this for every Basisregistratie Adressen en Gebouwen (BAG) building object within the defined AOI. Although the final implementation for phase II primarily used the drone dataset of the Geodelta office due to better reconstruction outcomes, this section outlines the original methodology using the oblique dataset.

We developed a method that leverages the BAG ID as a unique building identifier, along with 3D building world coordinates from the BAG. Using the orientation data from the oblique images, we reproject the 3D world coordinates into image space. This way we know: which images contain a given building, where the building appears in the image, and how to assign each segmented building mask back to its corresponding BAG ID.

We propose the following approach:

1. We use the BAG dataset to identify all the *bag objects* within our AOI (see Figure 3.3a).
2. We use the flight plan datasets, which contains footprint polygons for all aerial images, to determine which buildings appear in which images (see Figure 3.2a).
3. Using the internal and external camera parameters of the images, we reproject the bounding box and centroid of the *bag objects* onto the image plane.

By doing so, we determine the locations of all *bag objects* on the image planes and associate them with their corresponding *bagid*, which we can then use for segmentation (see Figure 3.3b).



(a) Partial flightplan example Utrecht

(b) Corresponding obliques

Figure 3.2.: (a) Partial flight plan for Utrecht, showing the footprints of all oblique images. (b) Subset of oblique image footprints intersecting with the geometry of a single BAG object.

(a) Overview of all BAG objects within the AOI, located within the Leidsche Rijn district of Utrecht. Green polygons represent building footprints. The yellow boundary shows the oblique dataset coverage.

(b) Reprojection of BAG building coordinates onto the image plane. Green boxes indicate projected locations with a 200-pixel x-offset applied to account for building height.

Figure 3.3.

3.3. Phase I: Segmentation in obliques using SAM

3.3.1. SAM modes

Once we have identified the position of the BAG IDs in the image planes where we need to extract a mask, we can use different modes of SAM prompting. Below is a brief overview of the different modes and how we use them to extract building masks from the obliques:

- **Point-based prompting**: A single point is provided to SAM, located at the centroid of the BAG object. This point is obtained by reprojecting the 3D centroid from world to image coordinates (Figure 3.4a).
- **Box-based prompting**: A bounding box around the BAG object is computed via reprojection of its footprint. To account for building height and camera perspective, we apply a buffer of 20 pixels on all sides and an additional 250 pixels vertically (Figure 3.4b).
- **Foreground/background point prompting**: Here, the reprojected building envelope is filled with foreground points, while background points are placed in a buffered region around the building (Figure 3.4c).
- **Full image segmentation**: Using the `SamAutomaticMaskGenerator` class, we segment the entire image without explicit prompts. Masks overlapping with reprojected building coordinates are selected post hoc (Figure 3.4d).

3.3.2. Testing different building classes

To evaluate SAM's performance across different building classes, we will test the previously described modes on various building classes.

During this research phase, we will apply these methods to three aerial images, for each specific building class. We will then evaluate performance based on:

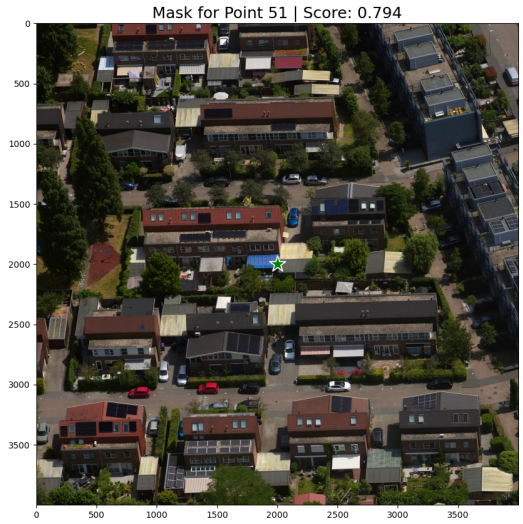
1. The average confidence score assigned by SAM to each mask.
2. The percentage of buildings with a confidence score higher than 0.85.

The *confidence score* in SAM represents the model's certainty about a given segmentation mask. It is a normalized score between 0 and 1, where higher values indicate greater confidence that the detected region accurately corresponds to an object boundary.

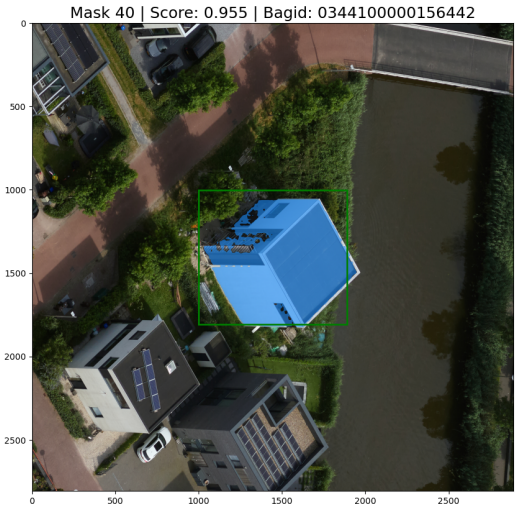
SAM generates multiple possible masks for a given input prompt (e.g., points, boxes), and it assigns a confidence score to each mask based on how well the segmentation aligns with the model's learned representations. For our use case we use only use and process the mask with the highest score.

The building classes we will examine are:

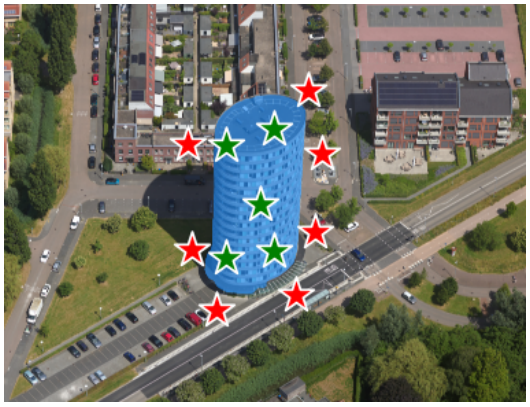
- Detached houses (see Figure 3.5c).
- Buildings taller than 30 meters (see Figure 3.5a).
- Buildings with a surface area greater than 300 square meters (see Figure 3.5b).



(a) Point prompt in SAM at the centroid of a *bag object*.



(b) Box prompt in SAM on the bounding box of a *bag object*.



(c) Building envelope prompt: foreground and background points.

(d) Full image segmentation using SAM.

Figure 3.4.: Different segmentation prompting methods using SAM.

(a) Buildings higher than 30 meters

(b) Buildings with surface larger than 300 square meters

(c) Detached houses

Figure 3.5.: Different building classes used for segmentation testing.

3.4. Phase II: Initial plans and methodology Shift

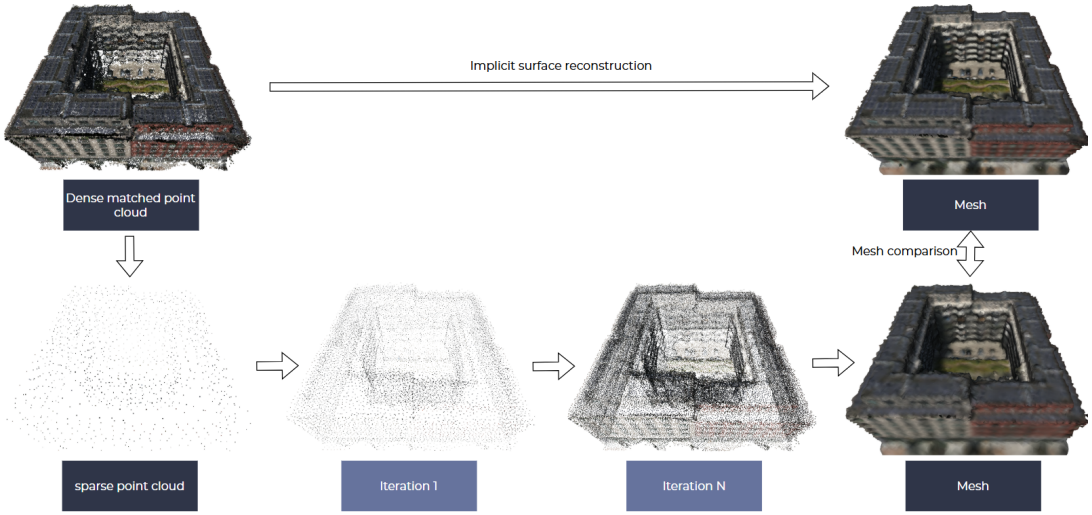


Figure 3.6.: Initial plan for adding geometrically important points for mesh reconstruction quality

Our initial objective was to create a sparse point cloud for each building, matched across different oblique images (see Figure 5.1), using segmentation masks. This sparse point cloud would serve as the foundation for computing geometric entropy and identifying regions where additional points could be added to improve reconstruction quality (see Figure 3.6).

At this stage, we asked ourselves: *"How can we adaptively add points to the sparse point cloud such that only those most important for mesh reconstruction quality are included?"* We explored point cloud analysis methods based on curvature entropy, edge features, and density features, inspired by the work of Shi et al. [2022] (see Appendix B).

In regions with high combined scores from these features, we explored sampling additional points using the following strategies:

- **Surface fitting:** Fit surfaces (e.g., via least squares) to high-scoring points and sample additional points from them.
- **Normal-based sampling:** For high-entropy points, sample along the normal vector to obtain new surface points.
- **Hierarchical sampling:** Use octree-based subdivision to increase sampling resolution in high-scoring regions.

However, this presented an issue: *where should we sample from*? If we sample arbitrarily in high-entropy regions, the new points may not correspond to the actual building surface. Thus, we needed to sample from a dense point cloud that had to be reconstructed beforehand, undermining our goal of reducing computational cost and mesh size.

Another limitation: the initial sparse point cloud plays a major role in determining where high-scoring regions are. If those scores are misleading due to poor initial coverage, we may

be reinforcing errors with each iteration. This issue could be addressed by simplifying the full dense point cloud using curvature, edge, and density features. However, this would turn into a point cloud simplification problem, and we would still need to first perform dense matching on the full point cloud.

Finally, this approach did not involve SAM, which we really wanted to explore, particularly its potential for capturing geometric entropy.

Eventually we came up with a methodology shift towards an image-based point addition method (see Section 3.6). In this approach, we hypothesize three strategies for identifying the most important parts of an image for mesh reconstruction quality. We still used sparse reconstruction (Section 5.3), now on the thresholded images to explore which configurations (threshold, method) were suitable for generating a mesh, and as a basis for the dense matching.

3.5. Phase II: Segmentation on building Level

Now that we have the segmented building masks, we zoom in further and segment these masks as well. In contrast to the oblique segmentation step, where we used a box prompt, we now apply the full *SamAutomaticMaskGenerator* mode from SAM. These segmentations of the individual building masks form the basis for our next step: identifying important regions in images for reconstruction. Two out of the three strategies we propose rely directly on this SAM based segmentation.

The goal here is to reduce the runtime and size of the dense matching process. To do that, we want to leave out less important parts of the images and keep only the relevant areas for matching. We are interested in seeing if segmentation from SAM can help indicate which parts of an image are important for reconstruction.

In later sections, we explain how we go from SAM outputs to normalized entropy maps that reflect importance. But for now, the main point is that we use SAM in this step. In our implementation, we also test different configurations of the full-image segmentation to find the best setup.

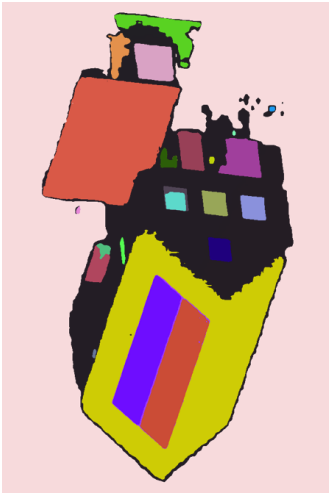


Figure 3.7.: Example of building-level segmentation.

3.6. Phase II: Identifying important image regions for reconstruction quality

(a) SAM mask kernel entropy (b) SAM mask edge distance (c) Canny edge distance

Figure 3.8.: Comparison of edge detection and entropy computation methods.

The goal of this part is to identify areas in images that are important for mesh reconstruction quality. For example, large planar surfaces on buildings typically offer little geometric information, while more complex regions such as ridges, corners or window details are more valuable for reconstruction. If we can identify and keep only the parts of the image that contribute the most, we can reduce both memory usage and computation time.

To estimate geometric importance or complexity within an image, we explore three strategies. Two of these are based on segmentations from SAM, and one uses Canny edge detection as a comparison.

The first SAM-based strategy uses a sliding kernel to compute local entropy values over the segmentation masks. The assumption is that regions with more overlapping or diverse masks indicate higher geometric complexity, while flat areas produce fewer masks and lower entropy.

The second strategy also starts from the SAM masks, but focuses on the distance from mask edges. The idea is that SAM tends to outline meaningful building features such as windows, doors, and rooflines. We apply a decaying distance function that assigns higher importance to pixels near these edges.

Finally, we use Canny edge detection directly on the building-masked images. This serves as a baseline to compare against the SAM-based approaches and to evaluate whether SAM adds any meaningful advantage over traditional edge-based methods.

3.6.1. SAM mask kernel entropy

Our first SAM-based metric uses a kernel-based entropy approach applied to segmented mask images. The steps are as follows:

1. **Segmentation:** We use the building-level segmentation masks (Section 3.5) as input.
2. **Kernel Convolution:** We slide a fixed-size kernel over the segmented image and analyze the diversity of segment labels within each kernel.
3. **Entropy Computation:** For each kernel position, we compute the entropy using:

$$H_G(x,y) = -\sum_{j=0}^{n} p_j \log p_j \tag{3.1}$$

where p_j is the proportion of pixels belonging to segment j within the local window:

$$p_j = \frac{\text{count of pixels from segment } j}{\text{total pixels in the window}} \tag{3.2}$$

This method accounts for both the number and distribution of unique segment labels, offering a more informative metric than simply counting unique segments.

3.6.2. SAM mask edge distance entropy

The second metric leverages the distance from the edges of segmentation masks. Since mask boundaries typically align with building features, such as windows and doors, these regions are often geometrically significant. First, we run the segmentation again on the building mask, and then we apply the following formula:

$$H_{edge} = H_{mask} \cdot e^{-\alpha \cdot \text{distance}} \tag{3.3}$$

Where:

- α is a constant that controls the rate of fading.
- H_{mask} is the mean entropy value derived from the kernel-based geometric entropy. Alternatively, $H_{\text{mask}} = 1$ if the effect is to be purely based on the distance from the edge of the mask.

3.6.3. Image-based Canny Edge Entropy

Additionally, we also used Canny Edge detection directly on the building mask. This was to have a comparison against SAM to see if it adds any value.

3.7. Phase II: Targeted point addition based on KPI

In this section, we use the importance maps from the previous section to create a sparse reconstruction based on image regions from those importance maps.

We begin by performing sparse reconstructions on all building datasets listed in Table 4.3. This serves two purposes: (1) to assess the general effectiveness of the reconstruction pipeline on each dataset, and (2) to identify which datasets are suitable for further dense reconstruction. As shown in the results (Section 5.3), many oblique aerial datasets with SAM-based segmentation struggle to reconstruct a meaningful sparse point cloud, even when using unthresholded masks. Applying thresholding in such cases is unlikely to help and may worsen the outcome. Therefore, we limit dense reconstruction efforts to those datasets where sparse reconstruction indicates promising potential.

To prepare the inputs, we apply pixel-wise thresholding to the original RGB images using normalized importance maps (values in [0, 1]). Pixels below the threshold are removed (set to black), while higher-entropy regions are preserved. This allows us to evaluate how selective masking affects reconstruction quality across datasets and threshold levels.

- The method is applied to seven building datasets, as detailed in Table 4.3.
- For each dataset, we evaluate all three importance map methods: SAM-based kernel entropy, SAM-based edge distance, and Canny edge distance.
- For each method-dataset combination, we apply nine different threshold levels in the range `[0.0, 0.1, ..., 0.9]`.

We then use COLMAP to perform sparse 3D reconstruction on each filtered dataset. Our objective here is to analyze how different combinations of threshold level, importance map method, and dataset affect sparse reconstruction quality. We can use these results to see which datasets we want to test for the mesh quality reconstruction, using dense matching later. The following reconstruction metrics are reported:

- **Threshold**: Original pixel values are retained if the normalized entropy map value (ranging from 0.0 to 1.0) is higher than this threshold.
- **Extraction time**: Time taken to extract features from all images in the dataset.
- **Matching time**: Time taken to match features across all images in the dataset.
- **Reconstruction time**: Time taken to incrementally perform SfM.
- **# Mean observations/Image**: average number of 2D feature observations per registered image.
- **# Images**: Number of registered images used in the SfM reconstruction.
- **# Points**: Number of reconstructed 3D points.

- **Mean track length**: Average number of images in which each 3D point is observed.
- **Mean reprojection error**: Average pixel error between observed image points and their reprojected 3D positions.

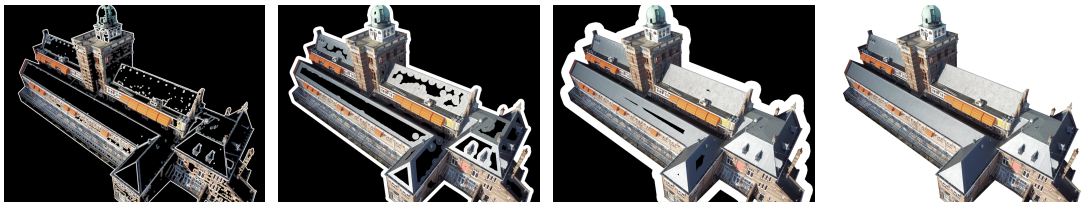
Table 3.1.: Overview of building datasets and their configurations

Building	Number of images	Capture method	Mask extraction method	bag-id
1_detached_house	28	Aerial	SAM	0344100000157740
1_detached_house	45	Aerial	Manual	0344100000157740
2_tall_building	29	Aerial	SAM	0344100000080618
3_apartment_block	39	Aerial	SAM	0344100000149906
3_apartment_block	70	Aerial	Manual	0344100000149906
4_bouwpub_groundbased	53	Phone	Manual	0503010000044548
5_geodelta_drone	200	Drone	Manual	503010000003065

3.7.1. Threshold levels

To exclude pixels that, according to our entropy estimation, are less informative for 3D reconstruction, we thresholded the original images using the normalized entropy maps (values in the range [0, 1]). Pixels with entropy values above a given threshold were kept, while the rest were removed.

We applied a range of thresholds: `[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]`. A value of 0.0 keeps all original pixels, while higher values keep increasingly smaller regions with higher entropy. Figure 3.9 illustrates the effect of different thresholds.



(a) Threshold: 0.9 (b) Threshold: 0.6 (c) Threshold: 0.3 (d) Threshold: 0.0

Figure 3.9.: Entropy thresholding results at various levels.

3.7.2. Building types datasets

We ran the pipeline on different buildings and under several configurations to test various aspects and better understand the pipeline. Three of the datasets originate from Phase I, segmented from the oblique aerial imagery of Utrecht. Two additional datasets—the Bouwpub (captured with a phone) and the Geodelta office (captured by drone), were included because they contain significantly more images and offer greater viewpoint overlap. See Table 4.3 for dataset details.

Since we did not apply SAM-based segmentation to the Bouwpub and Geodelta_drone datasets (because our algorithm described in Section 3.2 requires a reprojected BAG bounding box), we manually masked the building from the rest of the image. To see if the segmentation

method affects the result (as opposed to factors like the number of images or the close-up nature of the data), we also manually masked a few images from the 1_detached_house and 4_apartment_block datasets.

To investigate whether the number of images significantly impacts reconstruction quality (since the Geodelta drone dataset contains significant more images (200) than the others), we ran the pipeline on that dataset using subsets of [10, 40, 75, 150, 200] random picked images. This helps isolate the effect of image count from other factors like distance to the subject, segmentation method, or building complexity (e.g., the Geodelta building has many distinctive features that may aid in feature matching).

3.7.3. Sparse reconstruction

We performed feature extraction, feature matching, and sparse reconstruction using COLMAP. Each reconstruction yielded a sparse point cloud, which we evaluated using the following metrics:

- **Threshold**: Original pixel values are retained if the normalized entropy map value (ranging from 0.0 to 1.0) exceeds this threshold.
- **Extraction time**: Time required to extract features from all images in the dataset.
- **Matching time**: Time required to match features across image pairs.
- **Reconstruction time**: Time required for incremental Structure-from-Motion (SfM).
- **# Mean observations/image**: Average number of 2D feature observations per registered image.
- **# Images**: Number of registered images used in the SfM reconstruction.
- **# Points**: Number of reconstructed 3D points in the sparse model.
- **Mean track length**: Average number of images in which each 3D point appears.
- **Mean reprojection error**: Average pixel error between observed and reprojected image points.

3.8. Phase II: Implicit mesh creation

In this phase, we generate 3D surface meshes from dense point clouds. The dataset selected for this purpose is the Geodelta office drone dataset, which demonstrated the highest reconstruction potential in the previous phase based on metrics such as the number of registered images, number of sparse points, and mean reprojection error. Using dense point clouds generated from this dataset, we apply Poisson surface reconstruction in the open-source software `Meshlab` to produce a mesh for each combination of importance map method and threshold. These meshes form the basis for the mesh quality evaluation presented in Section 3.9.

3.8.1. Poisson surface reconstruction

To produce watertight and smooth 3D meshes, we use the Poisson surface reconstruction method, see Section 5.4.

We explored how varying three key parameters of the Poisson reconstruction affects mesh quality and complexity:

- **Reconstruction Depth:** depth of the octree, meaning higher values offer more detail at a computational cost. Only an upper bound, as it adapts to the sampling density.
- **Minimum Number of Samples per Node:** minimum number of sample points per octree node, with higher values providing smoother, more noise-resistant reconstruction.
- **Interpolation Weight:** determines the importance of point sample interpolation, with higher values capturing more detail.

Table 3.2 shows the parameter ranges we tested for each parameter.

Table 3.2.: Tested parameter ranges for Poisson surface reconstruction

Parameter	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Reconstruction Depth	5	6	7	8	9	10	11	12	13
Minimum # Samples	1	2	3	5	10	15	25	50	100
Interpolation Weight	1	2	4	8	16	32	64	128	256

3.8.2. Mesh quality evaluation configuration

For each parameter, we performed a sweep while fixing the others, recording both visual outputs and complexity metrics (vertex/face counts). Based on this analysis, we selected an optimal configuration, which was then used to reconstruct meshes for all 27 dense point clouds from the Geodelta office drone dataset (9 thresholds × 3 importance methods). In addition, we generated a reference mesh using the full, unmasked images, which serves as the ground truth for evaluation. These meshes form the basis for the quality comparison presented in Section 3.9.

3.9. Phase II: Mesh quality evaluation

To evaluate the geometric quality of the reconstructed meshes, we compared each mesh against a reference mesh generated through full-image dense matching, which serves as the ground truth. This evaluation was conducted across all three importance methods and a range of threshold values.

3.9.1. Mesh quality distance metrics

We evaluated the quality of each mesh by computing vertex-to-face distances from every vertex in the evaluated mesh to the nearest face on the ground truth mesh. The following distance-based metrics were computed:

- **Mean distance:** The average Euclidean distance from each vertex to the closest face of the ground truth mesh.
- **Median distance:** The median of the distance distribution.
- **Hausdorff distance:** The maximum distance from any vertex in the evaluated mesh to the nearest surface of the reference mesh.

In addition to geometry error, we will also evaluate several metrics related to computational efficiency and mesh complexity.

- **# Vertices (input):** The number of vertices in the dense point cloud prior to surface reconstruction
- **# Vertices / Faces (mesh):** The vertex and face count of the reconstructed mesh.
- **Dense matching time:** The processing time required to generate the dense point cloud using COLMAP.

3.9.2. High vs. low density

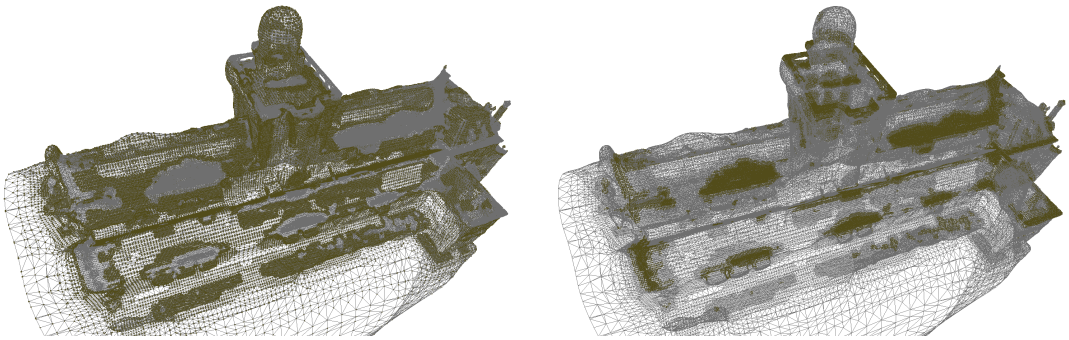
Our hypothesis is that thresholding based on image entropy will retain image regions that contribute most to mesh reconstruction quality, such as regions with more geometric detail, while discarding regions like large planar surfaces. As a result the reconstructed mesh should be denser in areas retained in the thresholded images and sparser where pixels have been discarded.

Because the dense matching algorithm uses the same image content as input, we expect the reconstructed geometry in high-density regions to closely follow the ground truth mesh. On the other hand, sparsely reconstructed regions may be less accurate, but ideally not by a large amount if they correspond to uninformative image areas for mesh reconstruction.

To test this hypothesis, we further subdivide each evaluated mesh into two subsets based on local vertex density (see also Figure 3.10):

- **Top 50% density vertices** (denser regions of the mesh, corresponding to retained parts of the input images)
- **Bottom 50% density vertices** (sparser regions, corresponding to discarded parts of the input images)

We compute the same distance metrics (mean, median, Hausdorff) for each subset individually, comparing them against the reference mesh as described in Section 3.9.1.



(a) Bottom 50% density points (b) Top 50% density points

Figure 3.10.: Top 50% density points

4. Implementation

4.1. Input Data

The input data used for the retrieval of building masks in phase I, consist of oblique aerial images, flight plans, and camera interior and exterior parameters (see Table 4.1). All spatial data is referenced in the Amersfoort / RD New coordinate system (EPGS:28992).

Dataset	Type	Amount	Size (GB)
Oblique	.jpg	2,562 files, 4 folders	456
Nadir	.jpg	620 files, 1 folder	262
Flight plans	.shp	1 file	-
Camera int./ext.	.opt/.txt	1 file	-

Table 4.1.: Input data

The image data is stored on Geodelta's internal server. The oblique dataset contains four folders: *Back*, *Fwd*, *Left*, and *Right*. It also contains the corresponding `.xyz` files with image positions, and `.opt` files with camera specifications and orientation data. Examples of the `.opt` and `.xyz` file structure can be found in Figure 4.2. The flight plan shapefile contains the projected footprints of each aerial image (see Figure 3.2).

4.2. Linking buildings through images

Database structure

All orientations are delivered with the image dataset in an `.opt` file. Figure 4.2 shows a few lines as an example. A custom Python script was developed to extract interior and exterior orientation parameters from the `.opt` file. The output is split into separate CSV files for camera and image parameters, which are then imported into the PostgreSQL database.

Shapefiles of the flight plans and AOI were imported using *shp2pgsql* command-line tool that comes with PostGIS. The database structure can be seen in Figure 4.1.

This database supports the segmentation phase. Where we use a script that iterates over the images in the dataset and retrieve information from this database to determine which BAG IDs are present in each image and to obtain the orientation data needed to reproject the BAG coordinates into image space. This then tells us where to segment, and allows each segmentation mask to be linked back to a corresponding BAG ID.

The `image_specs` and `camera_specs` tables store the external and internal parameters, respectively. The `flightplans` table contains the geometry of each image footprint. This

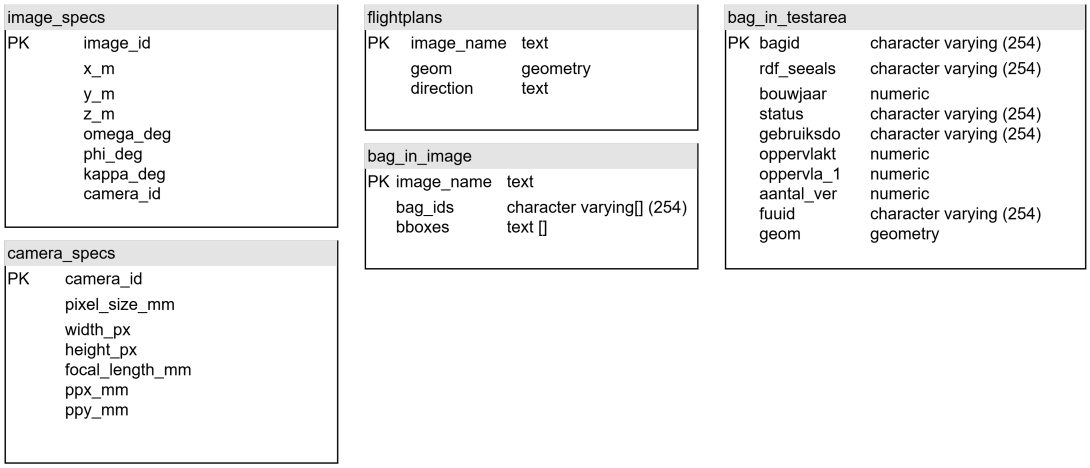


Figure 4.1.: Database structure used for linking image data with building geometries. The `image_specs` table stores exterior orientation parameters (x, y, z, omega, phi, kappa) and references the `camera_id` in `camera_specs`, which holds the internal camera parameters. The `flightplans` table contains the footprint geometry of each image. The `bag_in_image` table lists, for each image, the BAG objects IDs present and their bounding boxes in world coordinates. The `bag_in_testarea` table, contains full geometry and metadata for each building project.

footprint can be intersected with the geometries in the `bag_in_testarea` table, which contains all the BAG IDs in our AOI, to determine which buildings appear in which images. The `bag_in_image` table then stores, for every image, all BAG IDs present and their corresponding bounding boxes.

We used the 3D BAG coordinates and the known camera orientations to determine which buildings appear in which images, as well as their locations within those images. Based on this information, we perform segmentation in the following section.

CameraId	PixelSize [µm]	Width [px]	Height [px]	Focal [mm]	PPX [mm]	PPY [mm]	
Back	3.76	14204	10652	108.694	-0.045	-0.048	
Fwd	3.76	14204	10652	108.752	0.06	-0.154	
Left	3.76	14204	10652	108.836	-0.146	0.227	
Right	3.76	14204	10652	108.807	0.033	-0.076	
ImageId	X [m]	Y [m]	Z [m]	Omega [deg]	Phi [deg]	Kappa [deg]	CameraId
262000203_0067_01_0110_P00_01	131187.73650	457238.56580	427.39680	-0.20540	44.80360	269.42430	Back
262000204_0067_01_0109_P00_01	131137.40660	457238.13440	429.09200	-0.19240	44.81710	269.36400	Back
262000205_0067_01_0108_P00_01	131087.09660	457237.63790	431.21260	-0.22510	44.83760	269.44480	Back
262000206_0067_01_0107_P00_01	131036.71030	457237.31560	433.30000	-0.13140	44.85400	269.47040	Back
262000207_0067_01_0106_P00_01	130986.29360	457237.65140	435.29080	-0.56930	44.80520	269.04450	Back

Figure 4.2.: .opt file example

Reprojecting 3D bounding boxes onto Image Planes

To project building geometries onto image planes, we first identify which buildings intersect each image footprint. This is done by intersecting the image footprints in the `flightplans` table with the building geometries in the `bag_in_testarea` table using a spatial join in PostGIS.

For each matching building-image pair, the 3D terrain points are reprojected into pixel coordinates using the collinearity equations implemented in the `Geodelta.Photogrammetry` library. This transformation uses both the internal camera parameters (from `camera_specs`) and the external orientation parameters (from `image_specs`). A 2D bounding box is then computed from the projected points and stored in the database in WKT polygon format.

These bounding boxes are stored in a table that links each image to the visible BAG IDs and their pixel-space locations (table `bag_in_image`). When performing segmentation on the oblique images using SAM (as described in Section 4.3), we retrieve this information by querying the PostgreSQL database through a connection established with the `psycopg2` package.

4.3. Segmentation in obliques using SAM

4.3.1. SAM configuration

The SAM operates in two modes: full-image segmentation and prompted segmentation. Full-image segmentation is performed using the *SamAutomaticMaskGenerator* class, whereas the *SamPredictor* class is used for handling point, box, and other types of prompts, as described in Section 3.3.1.

SAM offers three model checkpoint variants: ViT-H (default), ViT-L, and ViT-B. These correspond to different *Vision Transformer (ViT) architectures*, with ViT-H being the largest and ViT-B the smallest. In our process, we used the ViT-B model. This made it more efficient to run on a large number of images.

We ran the ViT-B model on an NVIDIA GeForce RTX 3050 Ti laptop GPU with CUDA support, see Table 4.2 for more information.

System Information	Details
Operating System	Windows 11 Pro 64-bit (10.0, Build 22631)
System Manufacturer	Dell Inc.
System Model	Vostro 7620
BIOS	1.23.0
Processor	12th Gen Intel(R) Core(TM) i7-12700H (20 CPUs)
Memory	16,384 MB RAM
DirectX Version	DirectX 12

Table 4.2.: System specifications of the Dell Vostro 7620.

Segmentation using SAM

Segmentation is performed in a Jupyter Notebook using the `SamPredictor` box prompts.

In our methodology (Section 3.3.1), we explored four different ways of prompting SAM. Ultimately, we opted for the box prompt in our implementation. The point prompt often produced high-quality masks but generally lacked awareness of the building's scale. As a

Figure 4.3.: Reprojected bounding box (red square, also used as the box prompt) within the cropped image. The image is the result of adding a 250-pixel buffer on each side of the original bounding box.

result, it frequently returned accurate masks of smaller, uniform parts of the building, such as a solar panel or window, rather than the entire structure. The bounding box prompt consistently produced better segmentation results by guiding the model to focus on the full building.

We also experimented with using the building envelope as a prompt, along with foreground and background points. However, this approach introduced significant complexity and instability. It required reprojection not only of the bounding box but of the full building envelope, including appropriate perspective corrections. Finally, we chose not to use the automatic full-image segmentation mode because it often failed to identify the target building, attempting instead to segment all prominent objects in the image indiscriminately.

The segmentation pipeline proceeds as follows:

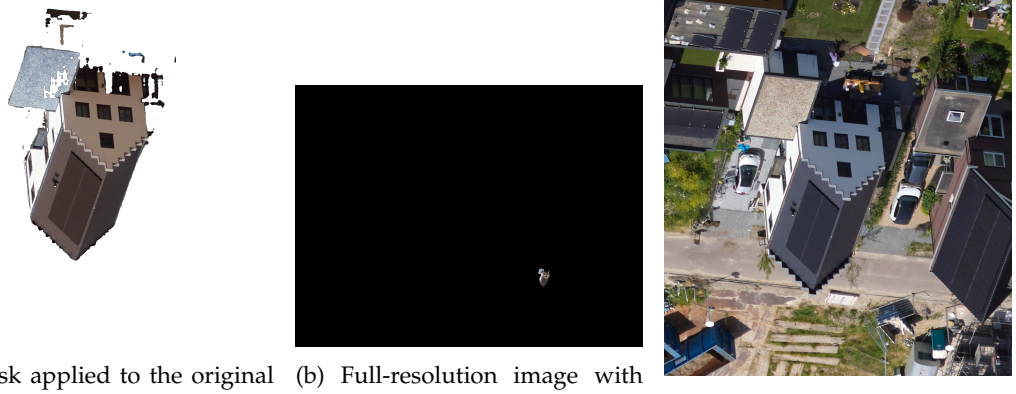
1. **Data retrieval:** For each oblique image, we query the PostgreSQL database to retrieve the associated `bag_ids` and bounding boxes. These bounding boxes are preprocessed and stored in image coordinate space.

2. **Image Loading and Cropping:** Images are loaded using the OpenCV library Bradski [2000]. To reduce computational load, we avoid prompting on the full-resolution image. Instead, we crop the image around each bounding box, adding a buffer of 250 pixels to ensure the entire building is included. The box prompt is then applied to the cropped image. This approach is significantly faster because the image input to the SAM model is much smaller, reducing the encoding time.

Note: This buffer is applied during segmentation for computational efficiency and differs from the earlier buffer (100 pixels on each side and 250 pixels on the height side) that was used to convert the reprojected BAG bounding box into a prompt box for the `SamPredictor`.

3. **Segmentation:** The cropped image and bounding box are passed to the `SamPredictor`, which generates a mask for the building (see Figure 3.4b).

4. **Output:** For each image, we saved three outputs: the mask applied to the original image, the full-resolution image with masks, and the cropped image version, see Figure 4.4.



(a) Mask applied to the original image

(b) Full-resolution image with masks

(c) Cropped image version

Figure 4.4.: Saved segmentation outputs: (a) mask on original, (b) full-resolution with masks, and (c) cropped image used for prediction.

4.4. Segmentation on building level

We applied the `SamAutomaticMaskGenerator` for full-image segmentation to identify detailed regions within building images. The results of this segmentation are used as input for the next step, where we aim to detect important regions in the images that contribute to dense matching and mesh reconstruction quality. This section outlines the method and configuration process used to select suitable parameter values. Evaluation results and visual comparisons can be found in Section 5.2.

4.4.1. Segmentation process

Different configurations of parameters for the `SamAutomaticMaskGenerator` were tested on the Geodelta drone image set. This dataset consists of 200 drone images that are manually segmented, and which is also used in following sections for mesh reconstruction. More details in Table 4.3.

For each parameter, we varied its value while keeping all other parameters at their default settings. To ensure consistency, each configuration was tested on the same 10 images selected from the dataset. For each run, we recorded:

- Mean runtime per image (in seconds),
- Mean predicted IoU score (mask quality),
- Mean number of masks generated per image.

4.4.2. Parameters Evaluated

- `points_per_side`
- `points_per_batch`
- `pred_iou_thresh`
- `stability_score_thresh`
- `stability_score_offset`
- `box_nms_thresh`

4.4.3. Evaluation on different building types

Once the final configuration of the `SamAutomaticMaskGenerator` was selected, we applied it across different building types as defined in Table 4.3. For each building, we report:

- Mean number of masks per image,
- Mean predicted IoU,
- Mean stability score.

Table 4.3.: Overview of building datasets and their configurations

Building	Number of Images	Capture Method	Mask Extraction Method	BAG-ID
1_detached_house	28	Aerial	SAM	0344100000157740
1_detached_house	45	Aerial	Manual	0344100000157740
2_tall_building	29	Aerial	SAM	0344100000080618
3_apartment_block	39	Aerial	SAM	0344100000149906
3_apartment_block	70	Aerial	Manual	0344100000149906
4_bouwpub_groundbased	53	Phone	Manual	0503010000044548
5_geodelta_drone	200	Drone	Manual	0503010000003065

4.5. Identifying important image regions for reconstruction quality

In this section, we describe the workflow used to compute the three importance maps that identify important regions for mesh reconstruction.

4.5.1. Overview

Our pipeline takes as input building-level segmentation masks (produced by SAM according to Section 4.4) for the two SAM-based methods, or the original building masks in the Canny edge detection method. They then produce normalized importance maps highlighting what the method thinks is the important part of the image for mesh reconstruction quality. The process is implemented in Python using OpenCV [Bradski, 2000], SciPy [Virtanen et al., 2020], and scikit-image [van der Walt et al., 2014] libraries.

4.5.2. SAM mask kernel entropy

The SAM building-level segmentation outputs an RGB image with a different colour for each mask. We convert them to grayscale here, where each grayscale pixel value corresponds to a unique segment.

- We then use a sliding window with a kernel size set to 1/8 of the image's largest dimension. A kernel too small would only capture one edge of a mask at a time, basically recreating the segmentation masks again. A kernel too big would smooth the output map too much.
- At each kernel position, compute the entropy of the segment labels within the window, according to equation 3.1.
- We normalize the resulting entropy map to [0, 1].

We used `scipy.ndimage.generic_filter` with a custom entropy function to compute local entropy. Alternatively, for more intense data, a faster approximation using scikit-image's `entropy` function with a disk-shaped structuring element can be applied.

4.5.3. SAM mask edge distance

As with the SAM mask kernel entropy, the SAM building-level segmentation outputs were also converted to grayscale.

- Extract the edges of the masks.
- Compute the Euclidean distance transform from each pixel to the nearest edge.
- Calculate an exponentially decaying map from the mask edges using equation 3.3.
- Normalize the resulting map to [0,1].

The decay parameter α controls how quickly the importance decreases with distance from the edges; we set it to 0.01. When desired (as we did), the kernel-based entropy term H_{kernel} can be replaced with a constant 1 to generate a purely distance-based map.

4.5.4. Image-based Canny edge distance

For the Canny edge distance maps, we used the raw building masks converted to grayscale instead of the SAM outputs.

- Apply Canny edge detection directly on the grayscale building masks using thresholds 260 and 300, which control the sensitivity for detecting edges: the lower threshold (260) marks strong edges, and the higher threshold (300) helps suppress noise by filtering out weaker edges.
- Compute the Euclidean distance transform to the nearest detected edge.
- Generate the importance map using the same exponential decay formula as above with $\alpha = 0.01$.
- Normalize the resulting map to the range [0, 1].

4.6. Targeted point addition based on building complexity

To ensure reproducibility, we summarize the core implementation steps for filtering input images and reconstructing sparse point clouds using COLMAP. The pipeline consists of two main components:

4.6.1. Image thresholding

We applied different thresholding levels on the original image pixels using the importance maps. This was done using a custom Python script.

- **Inputs:** Raw RGB images and corresponding normalized entropy maps (grayscale, values in [0.0–1.0]).
- **Thresholding logic:** Pixels with entropy values $\geq$ threshold are retained; others are set to black.
- Thresholds: `[0.0, 0.1, ..., 0.9]`

4.6.2. Reconstruction with COLMAP

We performed sparse 3D reconstruction using the COLMAP GUI for each combination of dataset, importance map method, and threshold level. The reconstruction settings were as follows: data type set to *individual images*, quality set to *high*, with both *shared intrinsics* and *shared intrinsics per sub-folder* disabled. Only sparse models were generated (no dense reconstruction).

The procedure involved loading the thresholded images into COLMAP, running the full reconstruction pipeline (feature extraction, matching, and incremental SfM), and extracting statistics via the built-in `Show Model Statistics` tool.

In cases where COLMAP produced multiple disconnected models, we retained only the model with the highest number of registered images. Reconstructions with fewer than 20%

of the input images successfully registered were considered uninformative and excluded from further analysis. This explains why some result tables in Section 5.3 do not include all threshold levels.

The following metrics were recorded for each valid reconstruction:

- Feature extraction time
- Feature matching time
- Reconstruction time
- Number of registered images
- Number of reconstructed 3D points
- Mean track length
- Mean reprojection error
- Average number of observations per image

Table 4.4.: Overview of building datasets and their configurations

Building	Number of images	Capture method	Mask extraction method	bag-id
1_detached_house	28	Aerial	SAM	0344100000157740
1_detached_house	45	Aerial	Manual	0344100000157740
2_tall_building	29	Aerial	SAM	0344100000080618
3_apartment_block	39	Aerial	SAM	0344100000149906
3_apartment_block	70	Aerial	Manual	0344100000149906
4_bouwpub_groundbased	53	Phone	Manual	0503010000044548
5_geodelta_drone	200	Drone	Manual	0503010000003065

4.7. Implicit mesh creation

4.7.1. Dense matching

For the Geodelta office drone dataset, we performed dense matching for all 27 image subsets, corresponding to 9 threshold values across 3 importance map methods (SAM-entropy, SAM-edge distance, and Canny edge detection). Additionally, a dense reconstruction using the full (unmasked) image set was generated to serve as the ground truth. All dense matching was carried out using the COLMAP graphical user interface.

4.7.2. Poisson configuration

To determine optimal Poisson surface reconstruction parameters, we conducted a parameter sweep on the full-image dense point cloud. Meshes were generated using the open-source application `MeshLab`, which implements the Poisson surface reconstruction method based on [Kazhdan et al., 2006].

For each parameter sweep (see Table 4.5), we recorded the following metrics:

- Number of mesh vertices

- Number of mesh faces

- Number of input points in the dense cloud

Table 4.5.: Tested parameter ranges for Poisson surface reconstruction

Parameter	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Reconstruction Depth	5	6	7	8	9	10	11	12	13
Minimum # Samples	1	2	3	5	10	15	25	50	100
Interpolation Weight	1	2	4	8	16	32	64	128	256

4.7.3. Final meshes generation

Based on the configuration experiment, we selected the following reconstruction parameters: depth 10, minimum number of samples 50, and interpolation weight 1. These parameters provided a balanced trade-off between mesh detail and processing efficiency.

Using this configuration, we applied Poisson surface reconstruction in `MeshLab` to all 27 dense point clouds from the Geodelta dataset. The resulting meshes were exported in ASCII-format `.PLY` files to be used in the mesh quality evaluation described in Section 3.9.

4.8. Mesh Quality Evaluation

4.8.1. Dense matching and Poisson surface reconstruction

We performed minor manual pre-cleaning of the input point clouds in MeshLab. Points not belonging to the building, such as trees or other foreground elements, were removed. These would otherwise interfere with the Poisson surface reconstruction and cause artifacts, as shown in Figure 4.5. In all cases, this involved only a small number of points (around 100–200) out of point clouds containing between 1 and 7 million points.

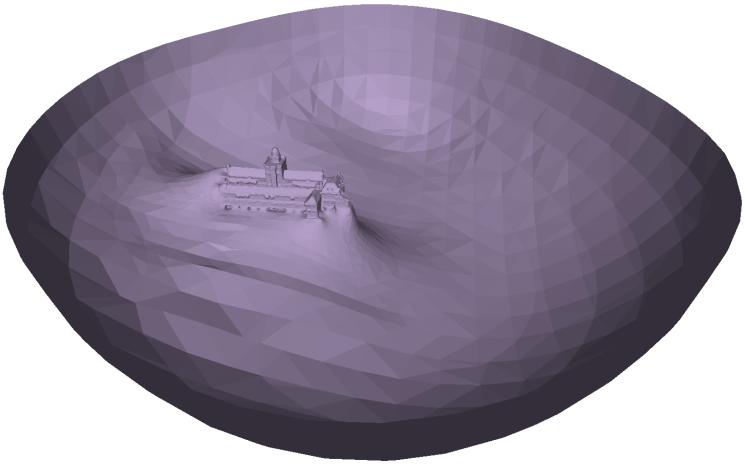


Figure 4.5.: Poisson reconstruction on point cloud with outliers

For the mesh quality evaluation, we then performed dense matching using COLMAP for all 27 mesh configurations (i.e., combinations of importance map type and threshold value) from the Geodelta drone office dataset. Subsequently, we generated the surface meshes in `MeshLab` using the Poisson surface reconstruction settings described in Section 4.7.3.

4.8.2. Mesh-alignment

Because the image datasets lack a fixed global reference frame, the different models exhibit arbitrary rotation, translation, and scale. An example of this misalignment is shown in Figure 4.6. To properly compare the two meshes using distance measures, they must be precisely aligned. We used the `meshlab-alignment` tool for this purpose, which allows you to select at least four corresponding points on both meshes and computes the rotation matrix accordingly. For each mesh, we selected ten points (to ensure accuracy) and applied the computed transformation to align the compared mesh with the ground truth mesh, see Figure 4.7. In all 27 cases, the ground truth mesh was kept fixed, and the evaluated mesh was aligned and scaled to match it

Figure 4.6.: Misalignment of ground truth mesh and 0.7 Image-based Canny Edge Entropy thresholded mesh.

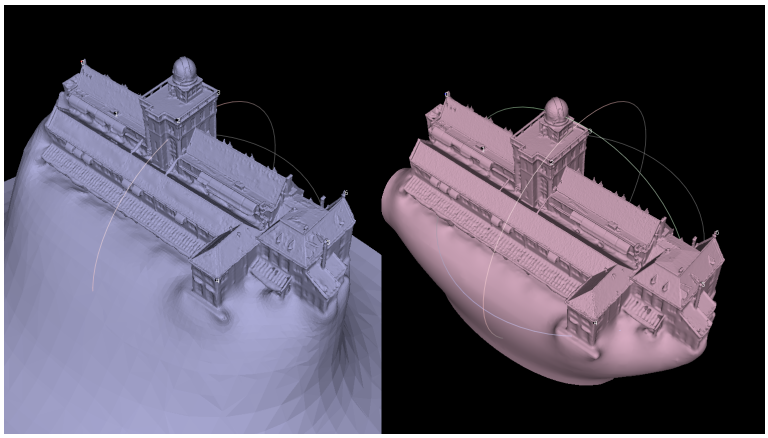


Figure 4.7.: Mesh alignment in Meshlab.

4.8.3. Distances computation

We then implemented a custom comparison pipeline in C# that parses the aligned mesh files in `.PLY` format. For each vertex in the evaluated mesh, we computed the shortest Euclidean distance to the nearest face (triangle) of the ground truth mesh.

To do so, the following steps were performed:

1. The ground truth mesh was loaded and its triangle faces reconstructed from the vertex and face information.

2. For each vertex in the evaluated mesh, the closest face in the ground truth mesh was identified using a spatial index (k-d tree) to accelerate nearest-neighbor queries.
3. The point-to-triangle distance was computed analytically for each evaluated vertex, using standard geometry formulas that calculate the perpendicular distance from a point to a triangle in 3D space.
4. All resulting distances were stored and used to compute the final statistics:
    - **Mean Distance:** the arithmetic average of all distances.
    - **Median Distance:** the median of the sorted distance values.
    - **Hausdorff Distance:** the maximum of the computed minimal distances.

5. Results

5.1. Segmentation in obliques using SAM

5.1.1. Overall performance

In total, **3,522** segmentation masks were generated and evaluated across the selected building samples. The average confidence score assigned by SAM across all masks was 0.746, indicating a generally high level of confidence in the generated masks, especially considering the complexity of oblique aerial imagery.

We further quantified performance by computing the percentage of high, confidence masks, defined as masks with a score greater than 0.85, per building. This threshold was chosen to reflect strong segmentation certainty. The distribution of high-confidence masks varies significantly among buildings, suggesting that SAM's segmentation quality is affected by building-specific characteristics.

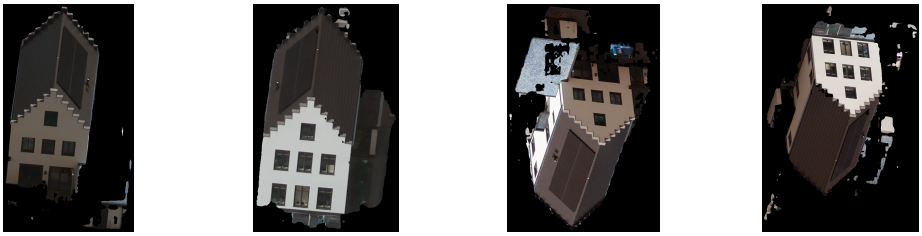
5.1.2. Interpretation of high-confidence Masks

The percentage of high-confidence masks (> 0.85) across buildings shows a wide range:

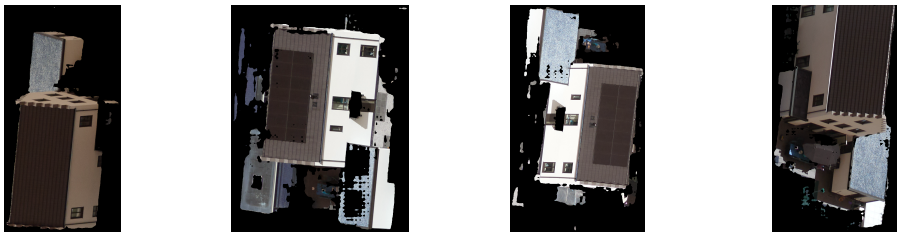
- Some buildings achieved perfect segmentation confidence (e.g., 100% of masks above threshold, such as BAG IDs: 0344100000053378, 0344100000019816).
- Others had poor performance, with no high-confidence masks (e.g., 0344100000017828, 0344100000080200).
- A significant number of buildings achieved over 75% high-confidence segmentation, indicating that SAM performs well for many cases even under oblique conditions.

5.1.3. Examples of segmentation quality

Figure 5.1 provides visualizations of segmentation masks from various viewing angles (front, back, left, right) for a selected building (BAG ID: 0344100000157740). While some views yield masks with tight alignment to building boundaries, others show more uncertainty, which may be reflected in their confidence scores.



(a) Fwd 1 (b) Fwd 2 (c) Back 1 (d) Back 2



(e) Left 1 (f) Left 2 (g) Right 1 (h) Right 2

Figure 5.1.: Selection of masks for a specific building (BAG ID: 0344100000157740)

5.2. Segmentation on building level

Our goal here is to find a configuration of the `SamAutomaticMaskGenerator` that has a balance between runtime, segmentation quality and number of generated masks. We evaluated several configurations of the `SamAutomaticMaskGenerator` on the Geodelta drone dataset. For each parameter, we analyzed its impact on runtime, average predicted IoU, and number of masks. This section includes a table and a figure for each parameter configuration. Note that the table presents average values across 10 images, while the figure shows only the first image in the dataset for illustrative purposes. For each parameter, the default value is indicated in bold.

5.2.1. Effect of `points_per_side` on segmentation quality and runtime

For the `points_per_side` parameter, increasing the value results in greater segmentation detail, but also significantly higher runtime. The runtime grows by factors ranging from approximately 2.3 to nearly 3.8 as the value doubles, while the number of masks increases rapidly up to 32 points per side and more gradually beyond that. This indicates diminishing returns in detail relative to computational cost, making 32 a balanced and efficient choice.

Table 5.1.: Effect of varying points per side (`points_per_side`) on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
8	0.78	0.9673	5.5
16	1.77	0.9685	15.60
32	5.25	0.9692	34.40
64	19.93	0.9690	46.70

(a) 16 (b) 32 (c) 64 (d) 128

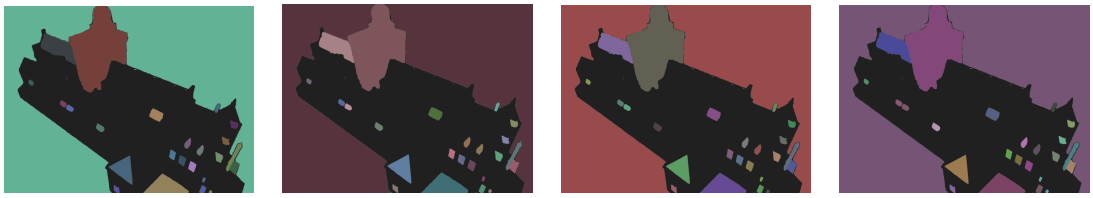
Figure 5.2.: Segmentation masks generated using varying values of points per side (`points_per_side`)

5.2.2. Effect of `points_per_batch` on segmentation quality and runtime

The `points_per_batch` parameter controls how many point prompts are processed in parallel during mask generation. This influences runtime and memory usage but has no effect on mask quality or count. As shown in Table 5.2, varying the value between 16 and 128 results in nearly identical average runtimes and identical mask scores and counts. This indicates that the parameter has minimal impact on performance within this range. Therefore, we will keep this default value of 64 for our configuration.

Table 5.2.: Effect of varying points per batch `points_per_batch` on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
16	5.60	0.9694	34.50
32	5.32	0.9694	34.50
64	5.36	0.9694	34.50
128	5.88	0.9694	34.50



(a) 16 (b) 32 (c) 64 (d) 128

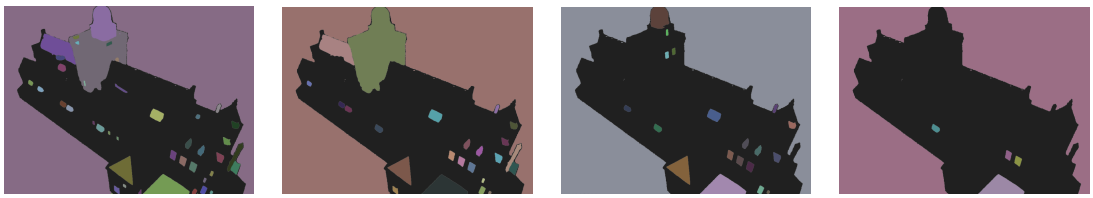
Figure 5.3.: Segmentation masks generated using varying values of points per batch (`points_per_batch`)

5.2.3. Effect of `pred_iou_thresh` on segmentation quality and runtime

The `pred_iou_thresh` parameter controls the minimum confidence (predicted IoU)[1] required to retain a mask. Lower values allow more masks, capturing finer details but with slightly reduced reliability. As shown in Table 5.3, decreasing the threshold from 0.95 to 0.75 increases the average number of masks from 7.2 to 45.4, while the average mask score decreases only slightly. Runtime remains unaffected. Since a lower threshold yields more masks without significantly compromising mask quality or runtime, we opted to reduce the value from the default and selected 0.75 for our configuration.

Table 5.3.: Effect of varying minimum confidence (predicted IoU) `pred_iou_thresh` on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
0.75	5.39	0.9679	45.40
0.88	5.31	0.9694	34.50
0.92	5.31	0.9711	22.40
0.95	5.28	0.9719	7.20



(a) 0.75 (b) 0.88 (c) 0.92 (d) 0.95

Figure 5.4.: Segmentation masks generated using varying values of the minimum confidence value (`pred_iou_thresh`)

[1]Intersection over Union (IoU) measures the accuracy of a predicted mask by comparing its overlap with the ground truth mask. Since the ground truth isn't available during inference, the predicted IoU is the model's estimated confidence score indicating how well the mask likely fits the object.

5.2.4. Effect of `stability_score_thresh` on segmentation quality and runtime

The `stability_score_thresh` parameter sets the minimum stability score required for a mask to be kept, reflecting how consistently the mask appears under small image changes. As the threshold increases, the number of masks decreases. Although runtime decreases slightly and average mask score improves with higher thresholds, the 0.75 thresholds tends to prioritize smaller, less important masks, causing some good masks to be lost. Visual inspection (Figure 5.5) shows that values around 0.85 and 0.95 better capture relevant details compared to the lowest threshold. Considering this, we selected 0.85 for our configuration.

Table 5.4.: Effect of varying the stability score threshold `stability_score_thresh` on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
0.75	7.36	0.9412	51.20
0.85	6.97	0.9516	48.40
0.95	5.40	0.9694	34.50
0.99	4.66	0.6950	1.20

(a) 0.75 (b) 0.85 (c) 0.95 (d) 0.99

Figure 5.5.: Segmentation mask generated using varying values of the stability score threshold (`stability_score_thresh`).

5.2.5. Effect of `stability_score_offset` on segmentation quality and runtime

The `stability_score_offset` shifts the stability scores before applying the `stability_score_threshold`, influencing which masks are considered stable. We chose a lower value of 0.8 because it produces more masks with a similar average mask score and no additional runtime cost, improving detail capture without sacrificing efficiency

Table 5.5.: Effect of varying the stability score offset (`stability_score_offset`) on runtime, average mask score, and number of masks

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
0.8	5.71	0.9723	39.70
1.0	5.85	0.9694	34.50
1.1	5.59	0.9682	32.00
1.3	6.11	0.9657	27.40

(a) 0.8 (b) 1.0 (c) 1.1 (d) 1.3

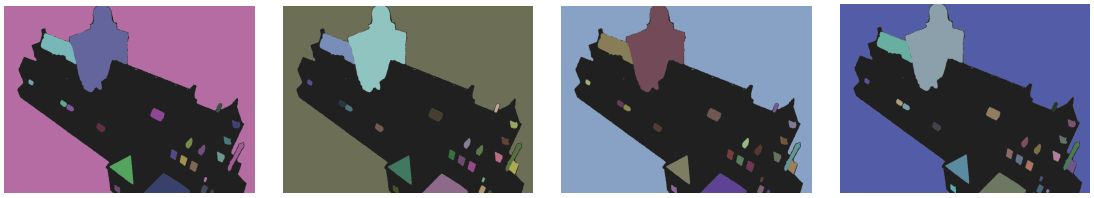
Figure 5.6.: Segmentation masks generated using varying values of the stability score threshold (`stability_score_offset`)

5.2.6. Effect of `box_nms_thresh` on segmentation quality and runtime

The `box_nms_thresh` parameter controls the non-maximum suppression (NMS) threshold for bounding boxes, determining how much overlap is allowed between masks. As shown in Table 5.6, varying this parameter does not change the metric much. Therefore, we kept this at the default value in our configuration.

Table 5.6.: Effect of varying (NMS) threshold for bounding boxes (`box_nms_threshold`) on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
0.2	5.28	0.9697	33.40
0.5	5.39	0.9696	33.90
0.7	5.27	0.9694	34.50
0.9	5.33	0.9692	37.40



(a) 0.2 (b) 0.5 (c) 0.7 (d) 0.9

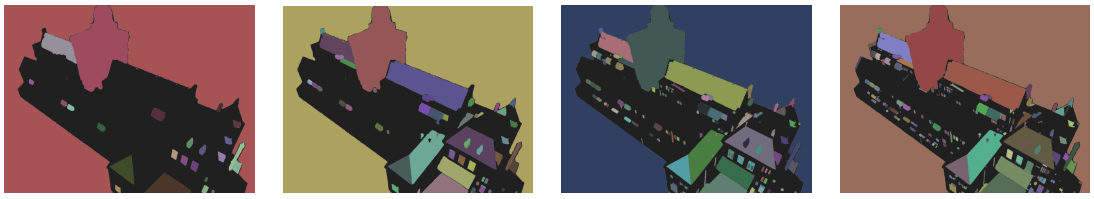
Figure 5.7.: Segmentation masks generated using varying values of (NMS) threshold for bounding boxes (`box_nms_threshold`)

5.2.7. Effect of `crop_n_layers` on segmentation quality and runtime

The `crop_n_layers` parameter enables recursive cropping for a specified number of layers. As shown in Table 5.7 and Figure 5.8, higher values significantly increase the number of masks but also increase runtime. Mask quality remains consistent across values. We opted for one level deeper than the default and chose a value of 1. To offset the runtime increase, we set the `crop_n_points_downscale` parameter to 2, which reduces the density of `points_per_side` in the recursive layers, reducing runtime.

Table 5.7.: Effect of varying the cropping depth (number of recursive layers, `crop_n_layers`) on runtime, average mask score, and number of masks.

Value	Avg. runtime (s)	Avg. mask score	Avg. # masks
0	5.20	0.9728	19
1	24.44	0.9661	83
2	96.96	0.9706	213
3	398.41	0.9740	436



(a) 0 (b) 1 (c) 2 (d) 3

Figure 5.8.: Segmentation masks generated with varying cropping depth values (`crop_n_layers`)

5.2.8. Final building level SAM configuration

To conclude, the parameters settings for the `SamAutomaticMaskGenerator` that we used on our datasets can be found in Table 5.9.

Parameter	Value
points_per_side	32
points_per_batch	64
pred_iou_thresh	0.75
stability_score_thresh	0.85
stability_score_offset	0.8
box_nms_thresh	0.7
crop_n_layers	1
crop_n_points_downscale_factor	2

Figure 5.9.: Configuration parameters used for the segmentation

Figure 5.10.: Final configuration on the image used in testing

5.3. Targeted point addition based on building complexity

In this section, we present the results of feature extraction, matching, and sparse reconstruction for the different building-type datasets, importance maps, and threshold levels.

We report the set of reconstruction metrics described in Section 3.7, including:

- Threshold level
- Extraction, matching, and reconstruction time
- Number of registered images and 3D points
- Mean track length, reprojection error, and observations per image

In this results section, we present the most important result tables, figures, and their interpretations. A complete set of all results, including full tables, is available in Appendix C.

Note: In some cases, COLMAP produced multiple disconnected models—especially at higher thresholds where larger portions of the input images were masked. In these situations, we report results only from the model with the highest number of registered images. This also influenced the reported reconstruction time: occasionally, a lower threshold may result in a longer reconstruction time due to the reconstruction process attempting an additional model.

For our first dataset, we include all reconstruction tables. The first dataset we tested on was the `1_detached_house` dataset, segmented from the oblique images using SAM. Although the results are listed here across all thresholds and entropy methods, not a single valid sparse model was constructed. Even without any thresholding, only 17 images were registered in the reconstruction—resulting in a skewed and incomplete model.

We do observe a decline in extraction, matching, and reconstruction times as thresholding increases and less information is retained from the images. This trend will continue throughout all models. An exception to this is threshold 0.3 in Table C.3, where the reconstruction time does not follow the expected pattern. This is an example where COLMAP attempted

to initiate a separate model from different perspectives, resulting in multiple disconnected models. Similar behavior can be seen in the following datasets.

Another consistent trend across all building datasets is that Canny edge detection typically results in the longest extraction, matching, and reconstruction times. This is due to Canny-thresholded images retaining the highest number of pixels on average.

5.3.1. 1_detached_house (SAM) - 28 input images

Table 5.8.: SfM results for 1_detached_house with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.086	0.005	0.287	12	418.333	1,193	4.20788	0.656676
0.2	0.088	0.004	0.113	7	187.143	384	3.41146	0.536607
0.3	0.089	0.005	0.265	6	179.833	340	3.17353	0.531796
0.4	0.084	0.004	0.030	4	122.25	194	2.52062	0.491187
0.5	0.086	0.003	0.003	2	60	60	2.0000	0.279012
0.6	0.080	0.002	0.009	4	112.5	168	2.67857	0.4359
0.7	0.084	0.001	0.004	2	29	29	2.0000	1.11085
0.8	0.082	0.002	0.000	0	0	0	0	0
0.9	0.081	0.001	0.000	0	0	0	0	0

Table 5.9.: SfM results for 1_detached_house with SAM edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.096	0.005	0.525	14	470.357	1,401	4.70021	0.679753
0.2	0.088	0.006	0.361	4	448.75	670	2.6791	0.473758
0.3	0.086	0.006	0.511	11	164.818	332	5.46084	0.71258
0.4	0.088	0.005	0.157	7	1.14286	4	2.0000	0.000017
0.5	0.085	0.005	0.202	9	365.444	965	3.40829	0.664438
0.6	0.085	0.005	0.030	4	67.75	99	2.73737	0.548408
0.7	0.088	0.005	0.024	4	120	123	2.93848	0.738290
0.8	0.093	0.006	0.025	4	170	222	3.06306	0.802794
0.9	0.083	0.004	0.008	4	140.5	205	2.74146	0.839855

Table 5.10.: SfM results for 1_detached_house with edge-detection

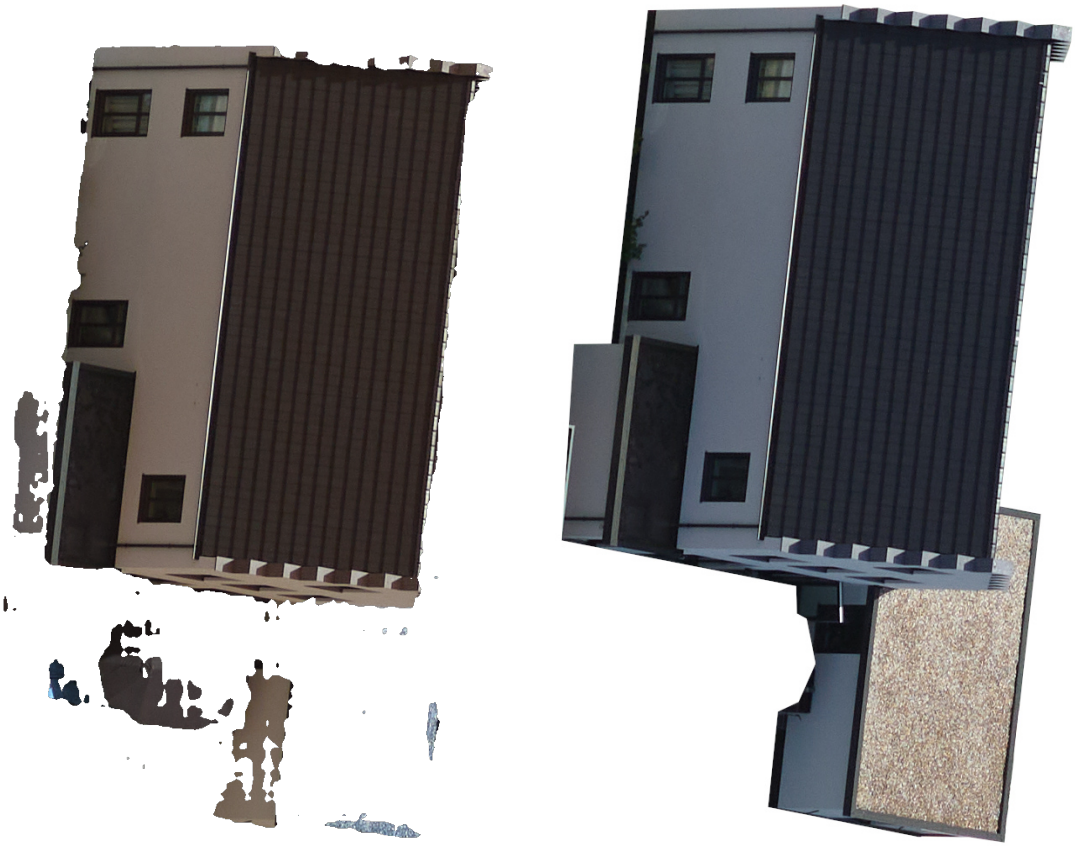
Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.087	0.005	0.486	11	331.545	694	5.25504	0.674564
0.2	0.099	0.006	0.717	12	549	1,487	4.4304	0.792753
0.3	0.092	0.005	0.351	5	368	626	2.9393	0.468019
0.4	0.086	0.006	0.470	17	468.412	1,895	4.20211	0.705464
0.5	0.090	0.006	0.518	11	322.273	697	5.08608	0.693039
0.6	0.089	0.006	0.505	4	389.25	576	2.70313	0.481089
0.7	0.086	0.007	0.375	12	253.083	818	3.71271	0.564703
0.8	0.089	0.008	0.084	5	316.4	440	3.59545	0.617071
0.9	0.086	0.006	0.019	3	292.667	386	2.27461	0.501573

5.3.2. 1_detached_house (manual segmentation) - 45 input images

We also manually segmented the building masks after we saw that the SAM segmented masks were not able of producing a good reconstruction. Therefore, for this same building the number of input images increases from 28 to 45, and also the quality of the masks increases see Figure 5.11. However, even though the number of registered images increased, not a single reconstruction accurately represented the real-world structure.

Table 5.11.: SfM results for 1_detached_house (manual) with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.110	0.037	3.673	35	1576.06	7,253	7.6054	0.911192
0.1	0.055	0.012	1.801	22	681.682	2,837	5.28622	1.02639
0.2	0.052	0.010	1.565	19	498.368	2,042	4.63712	1.10220
0.3	0.052	0.013	0.139	4	1.5	3	2	0.000028
0.4	0.050	0.016	0.089	4	279.75	490	2.28367	0.460264
0.5	0.046	0.009	0.019	3	0	0	0	0
0.6	0.044	0.008	0.006	2	176	176	2	0.600909
0.7	0.042	0.005	0.004	2	52	52	2	0.890843
0.8	0.036	0.003	0.001	2	25	25	2	0.575599
0.9	0.039	0.000	0.000	—	—	—	—	—



(a) SAM-segmented masks (28 input images).

(b) Manually segmented masks (45 input images).

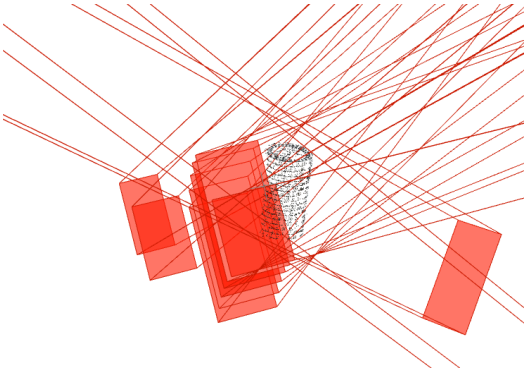
Figure 5.11.: Comparison of reconstruction input masks for the same building. After observing poor reconstructions using SAM-generated masks, we manually segmented the building. This increased the number of input images from 28 to 45 and improved mask quality.

For the dataset with the tall building we observe the same trends. However, we would like to mention here that out of all the oblique imagery datasets, the 0.1 threshold here produced the only reconstruction that was an accurate representation of the real world geometry, we show it in Figure 5.12

Table 5.12.: SfM results for 2_tall_building with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.309	0.064	2.402	15	1,921	4,241	3.12012	0.829121
0.1	0.304	0.106	2.886	10	1,716.4	5,219	3.28875	0.782024
0.2	0.291	0.104	3.444	19	1,721.21	9,782	3.34318	0.702166
0.3	0.264	0.102	6.101	14	1,663.21	7,760	3.00064	0.90227
0.4	0.279	0.098	1.967	13	1,249.69	5,316	3.05606	0.842655
0.5	0.305	0.100	3.205	10	69	267	2.58427	1.10686
0.6	0.296	0.102	2.430	11	200.636	791	2.79014	0.923647
0.7	0.291	0.095	3.539	12	550	2,222	2.9703	0.85611
0.8	0.290	0.088	2.332	6	4.83333	14	2.07143	0.000013
0.9	0.295	0.087	1.402	2	4	4	2	0.954711

(a) Input photo used in the successful reconstruction.



(b) Resulting sparse model from threshold 0.1, aligned with real-world geometry.

Figure 5.12.: Example from the `2_tall_building` dataset. Threshold 0.1 produced the only sparse model closely resembling the real structure.

5.3.3. 3_apartment_block (SAM) - 39 input images

5.3.4. 3_apartment_block (manual segmentation) - 70 input images

Both the SAM segmented and manually segmented datasets of the apartment block did not give any good reconstructions, we only include one table here as an example. All other tables for both datasets can be found in Appendix C.

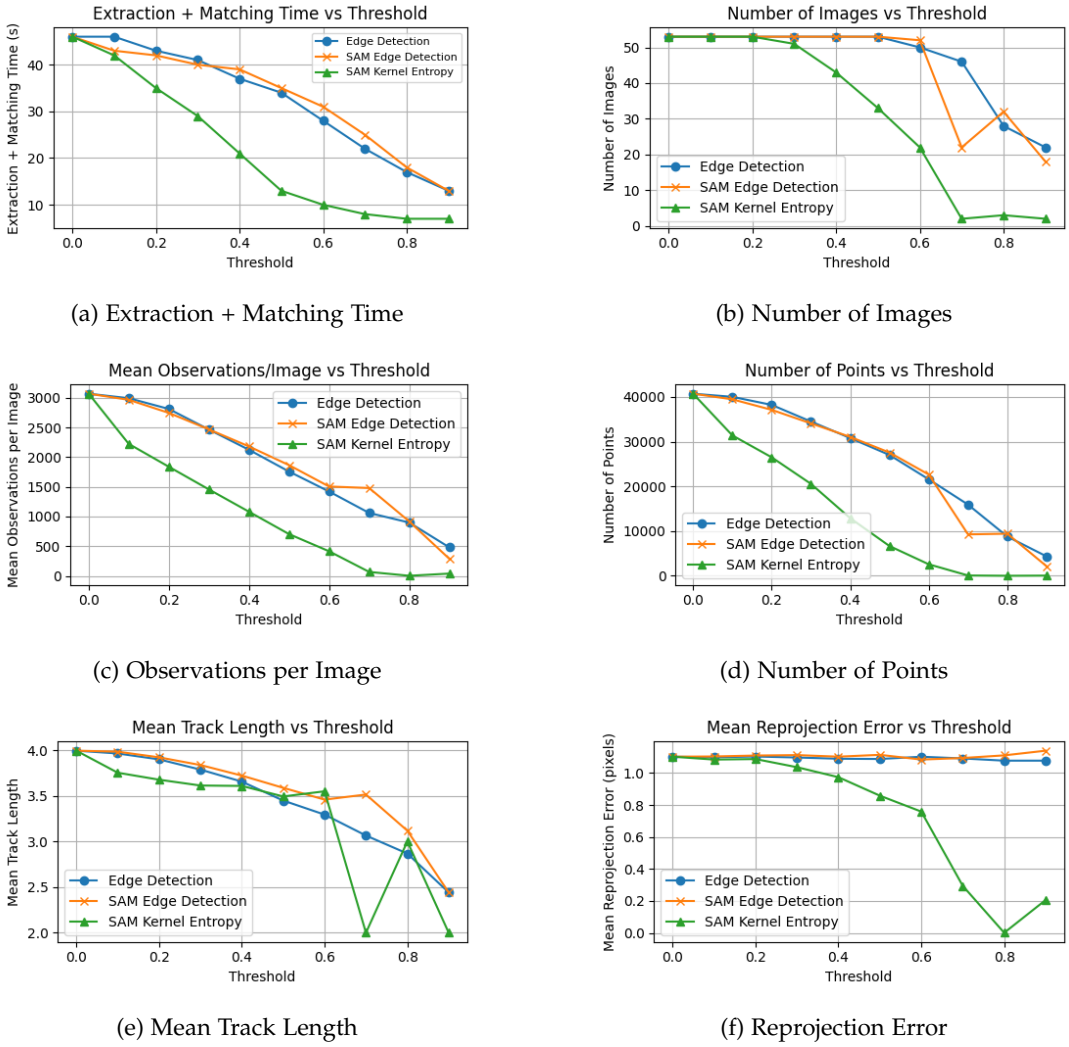
Table 5.13.: SfM results for 3_apartment_block with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.990	0.202	9.274	10	642.7	2,817	2.28151	0.922359
0.1	1.266	0.205	4.720	11	622.091	3,296	2.07615	0.938021
0.2	1.055	0.192	4.964	10	1,213.8	4,639	2.61651	1.152
0.3	0.860	0.180	4.136	15	713	3,422	3.12537	0.681197
0.4	0.833	0.149	3.441	14	997.429	3,900	3.58051	0.855569
0.5	0.670	0.118	2.312	13	1,150.92	3,765	3.97397	0.791224
0.6	0.552	0.066	1.730	11	933.909	2,761	3.72075	0.729669
0.7	0.446	0.025	0.336	9	585.778	1,416	3.72316	0.72576
0.8	0.324	0.009	0.126	6	292.5	602	2.91528	0.74585
0.9	0.326	0.007	0.019	2	61	61	2	0.409424

5.3.5. 4_bouwpub (manual segmentation) - 53 input images

The Bouwpub dataset, which was captured using a smartphone and manually segmented, represents one of the first datasets from which we obtain realistic reconstructions.

Figure 5.13 we can see the results of the bouwpub dataset for the main metrics.



(a) Extraction + Matching Time

(b) Number of Images

(c) Observations per Image

(d) Number of Points

(e) Mean Track Length

(f) Reprojection Error

Figure 5.13.: SfM evaluation metrics for Bouwpub dataset.

We observe in Figure 5.13a that extraction and matching times decrease as the thresholds increase. The SAM kernel entropy method has the lowest extraction and matching time overall, whereas the other two methods show similar performance.

A similar trend is visible in the number of registered images versus threshold, see Figure 5.13b. Up until a threshold of 0.2, all three methods reconstruct using all 53 input images.

After that, SAM kernel entropy starts to drop off, while the other two methods retain all images until around threshold 0.5.

The mean observations per image in Figure 5.13c are significantly lower for SAM kernel entropy compared to the other methods, indicating that this method discards more pixels. This also reflects in the number of points versus threshold in Figure 5.13d.

The mean track length in Figure 5.13e is the most comparable metric across the three methods. However, after threshold 0.6, the reconstruction quality for SAM kernel entropy declines due to insufficient image information.

A similar pattern appears in the mean reprojection error in Figure 5.13f, which decreases for SAM kernel entropy at higher thresholds. However, this is likely due to the reduced number of images and points, rather than improved reconstruction accuracy.

Table 5.14.: SfM results for 4_bouwpub (manual) with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m20s	0m26s	1m36s	53	2,992.75	40,023	3.96312	1.09889
0.2	0m17s	0m26s	1m32s	53	2,811.66	38,224	3.89855	1.10369
0.3	0m17s	0m24s	1m27s	53	2,464.81	34,500	3.78652	1.09756
0.4	0m16s	0m21s	1m19s	53	2,124.09	30,801	3.65498	1.08972
0.5	0m15s	0m19s	1m6s	53	1,757.58	27,019	3.44765	1.08829
0.6	0m14s	0m14s	1m5s	50	1,421.2	21,573	3.29393	1.10230
0.7	0m13s	0m9s	0m36s	46	1,060.2	15,914	3.06453	1.09169
0.8	0m11s	0m6s	1m27s	28	898.464	8,773	2.86755	1.07772
0.9	0m10s	0m3s	0m16s	22	480.545	4,330	2.44157	1.07777

Table 5.15.: SfM results for 4_bouwpub (manual) with SAM edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m18s	0m25s	1m51s	53	2,965.55	39,463	3.98282	1.10395
0.2	0m18s	0m24s	1m53s	53	2,745.38	37,116	3.92092	1.10973
0.3	0m17s	0m23s	1m52s	53	2,467.55	34,088	3.83654	1.11244
0.4	0m17s	0m22s	1m51s	53	2,180.53	31,059	3.72092	1.10269
0.5	0m15s	0m20s	1m25s	53	1,865.47	27,563	3.58706	1.11416
0.6	0m15s	0m16s	1m12s	52	1,505.38	22,637	3.45806	1.08461
0.7	0m14s	0m11s	1m41s	22	1,479.91	9,266	3.51371	1.09432
0.8	0m12s	0m6s	0m36s	32	917.344	9,415	3.11790	1.11087
0.9	0m9s	0m4s	0m24s	18	288.111	2,124	2.44162	1.14011

Table 5.16.: SfM results for 4_bouwpub (manual) with SAM kernel entropy

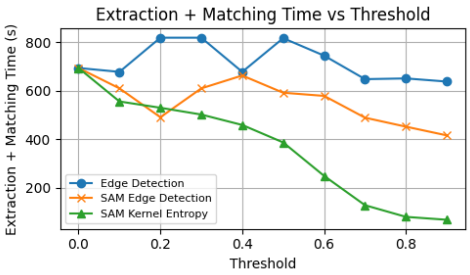
Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m20s	0m22s	1m16s	53	2,223.13	31,398	3.75266	1.08451
0.2	0m17s	0m18s	1m11s	53	1,838.26	26,503	3.67611	1.08779
0.3	0m16s	0m13s	0m55s	51	1,456.43	20,563	3.61222	1.03688
0.4	0m13s	0m8s	0m41s	43	1,081.12	12,885	3.60792	0.974324
0.5	0m9s	0m4s	1m1s	33	704.121	6,653	3.49256	0.858625
0.6	0m8s	0m2s	0m16s	22	414.591	2,570	3.54903	0.758927
0.7	0m7s	0m1s	0m1s	2	67.000	67	2.0000	0.294403
0.8	0m6s	0m1s	0m1s	3	2.000	2	3.0000	0.000600
0.9	0m6s	0m1s	0m1s	2	39.000	39.000	2.0000	0.205755

5.3.6. 5_geodelta_drone (manual segmentation) - 200 input images

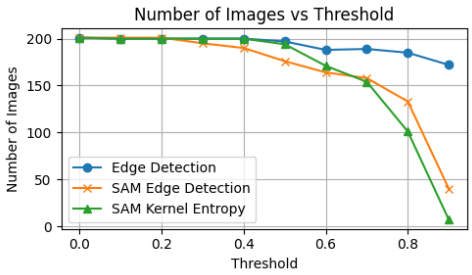
For the GeoDelta Drone Office dataset, we observe a consistent decrease in extraction and matching times as the thresholds increase, particularly for the SAM kernel entropy method,

which becomes significantly faster than the other two beyond threshold 0.4. The SAM edge detection and entropy methods show similar extraction and matching times until threshold 0.3, after which edge detection maintains slightly more consistent durations.

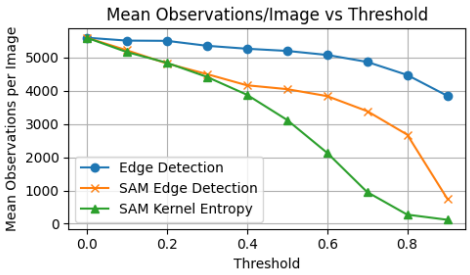
All methods start with 201 registered images, but SAM kernel entropy and edge detection both show a rapid drop in image count beyond threshold 0.5, with kernel entropy dropping to only 7 images at threshold 0.9. This trend also reflects in the number of points and mean observations per image, both of which decline sharply as the image masks become more sparse.



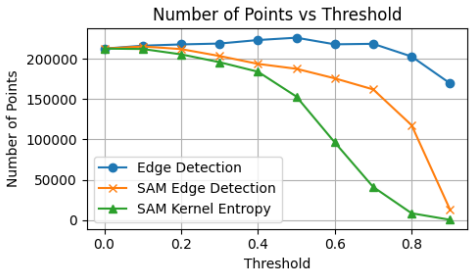
(a) Extraction + Matching Time



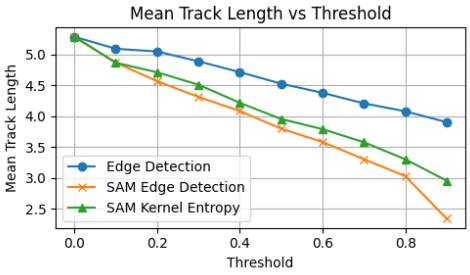
(b) Number of Images



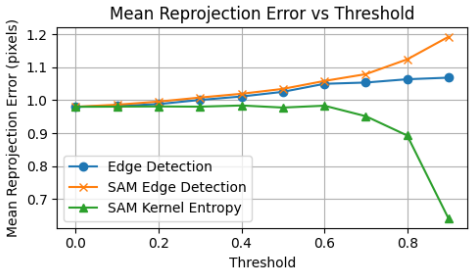
(c) Observations per Image



(d) Number of Points



(e) Mean Track Length



(f) Reprojection Error

Figure 5.14.: SfM evaluation metrics for GeoDelta Drone dataset.

Our initial plan was to explore mesh reconstruction for all datasets, but due to poor SfM results from the oblique-derived datasets, we focused instead on the GeoDelta Drone dataset. With more images and higher overlap, it produced the most complete and consistent recon-

Table 5.17.: SfM results for 5_Geodelta_drone with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	4m39s	6m38s	12m25s	200	5,507.8	216,269	5.09347	0.982634
0.2	6m55s	6m43s	13m33s	200	5,500.65	217,849	5.04997	0.988194
0.3	6m55s	6m43s	11m56s	200	5,352.19	218,915	4.88974	1.00084
0.4	4m36s	6m41s	11m49s	200	5,263.3	223,275	4.71463	1.01127
0.5	6m54s	6m42s	11m3s	197	5,197.9	226,189	4.52713	1.02545
0.6	5m27s	6m57s	13m45s	188	5,076.54	217,971	4.37851	1.04974
0.7	3m36s	7m11s	11m43s	189	4,867.19	218,592	4.20829	1.05362
0.8	4m8s	6m42s	10m59s	185	4,473.37	202,852	4.07969	1.0636
0.9	4m11s	6m26s	13m39s	172	3,847.78	169,543	3.90355	1.0684

Table 5.18.: SfM results for 5_Geodelta_drone with SAM edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	3m28s	6m41s	10m56s	201	5,216.94	215,316	4.87007	0.98635
0.2	3m28s	4m42s	12m24s	201	4,820.04	212,044	4.56900	0.99577
0.3	3m28s	6m41s	16m32s	195	4,499.99	203,484	4.31237	1.00809
0.4	4m46s	6m16s	10m55s	190	4,163.51	193,712	4.08373	1.01948
0.5	3m34s	6m17s	14m37s	176	4,048.31	187,623	3.79752	1.03436
0.6	3m27s	6m11s	12m34s	164	3,838.39	175,753	3.58171	1.05837
0.7	2m43s	5m25s	9m34s	158	3,386.99	162,049	3.30237	1.07905
0.8	2m58s	4m33s	14m0s	133	2,679.07	117,663	3.02828	1.12399
0.9	2m34s	4m21s	8m57s	40	750.975	12,856	2.33657	1.19275

Table 5.19.: SfM results for 5_Geodelta_drone with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	3m9s	6m6s	11m47s	200	5,163.67	212,185	4.86714	0.980572
0.2	3m2s	5m47s	9m55s	200	4,841.91	205,527	4.7117	0.981102
0.3	2m45s	5m36s	9m16s	200	4,416.39	195,804	4.51103	0.980386
0.4	2m16s	5m22s	9m31s	200	3,879.08	183,980	4.21685	0.983975
0.5	2m3s	4m23s	7m10s	194	3,122.97	153,196	3.95478	0.977657
0.6	1m39s	2m29s	5m2s	171	2,133.85	96,284	3.78972	0.983343
0.7	1m19s	0m48s	3m10s	154	952.896	41,018	3.5776	0.951435
0.8	1m6s	0m13s	2m47s	101	274.010	8,383	3.30132	0.892823
0.9	1m2s	0m5s	0m3s	7	119.000	282	2.9539	0.64008

structions, making it better suited for dense point cloud generation and mesh reconstruction.

5.4. Implicit mesh reconstruction

In this section, we present the results of our Poisson surface reconstruction parameter experiments conducted on the full-image dense point cloud. By systematically varying reconstruction depth, minimum number of samples, and interpolation weight, we analyze the impact of each parameter on mesh complexity and visual quality. Based on these findings, we identify an optimal configuration that is later applied to all 27 dense reconstructions. The selected configuration and corresponding reference mesh will serve as a basis for mesh quality evaluation in the following section.

5.4.1. Reconstruction depth

In Figure 5.15, we can visually inspect the different configurations of the reconstruction depth. Table 5.20 and Figure 5.16 show the corresponding number of vertices and faces for each parameter value.

Up until a depth of 7 or 8, there is a noticeable lack of detail. From 9 onwards, finer features such as window frames and the façade become visible. This level of detail continues to improve noticeably up to the final depth tested. Starting at depth 10 and beyond, we also observe some overfitting in the Poisson reconstruction, visible in the appearance of a small blob above the dome.

Considering both the vertex and face counts, a reconstruction depth of 10 appears to offer the best trade-off between visual detail and computational cost. Depths 11 and 12 more than double the memory cost, resulting in meshes that become increasingly difficult to process efficiently.

Table 5.20.: Number of vertices and faces for varying reconstruction depth (min samples = 10, interpolation weight = 4)

Depth	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
# Vertices	525	2083	5153	17368	74962	310334	1225572	4480679	13051596
# Faces	961	4004	10131	34566	149751	620395	2450966	8961366	26103701

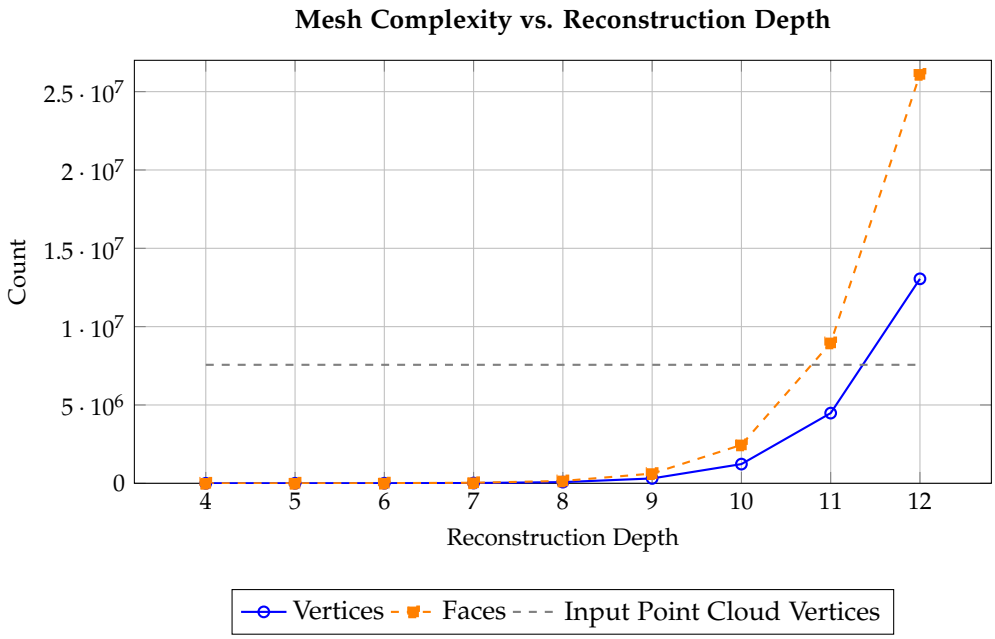
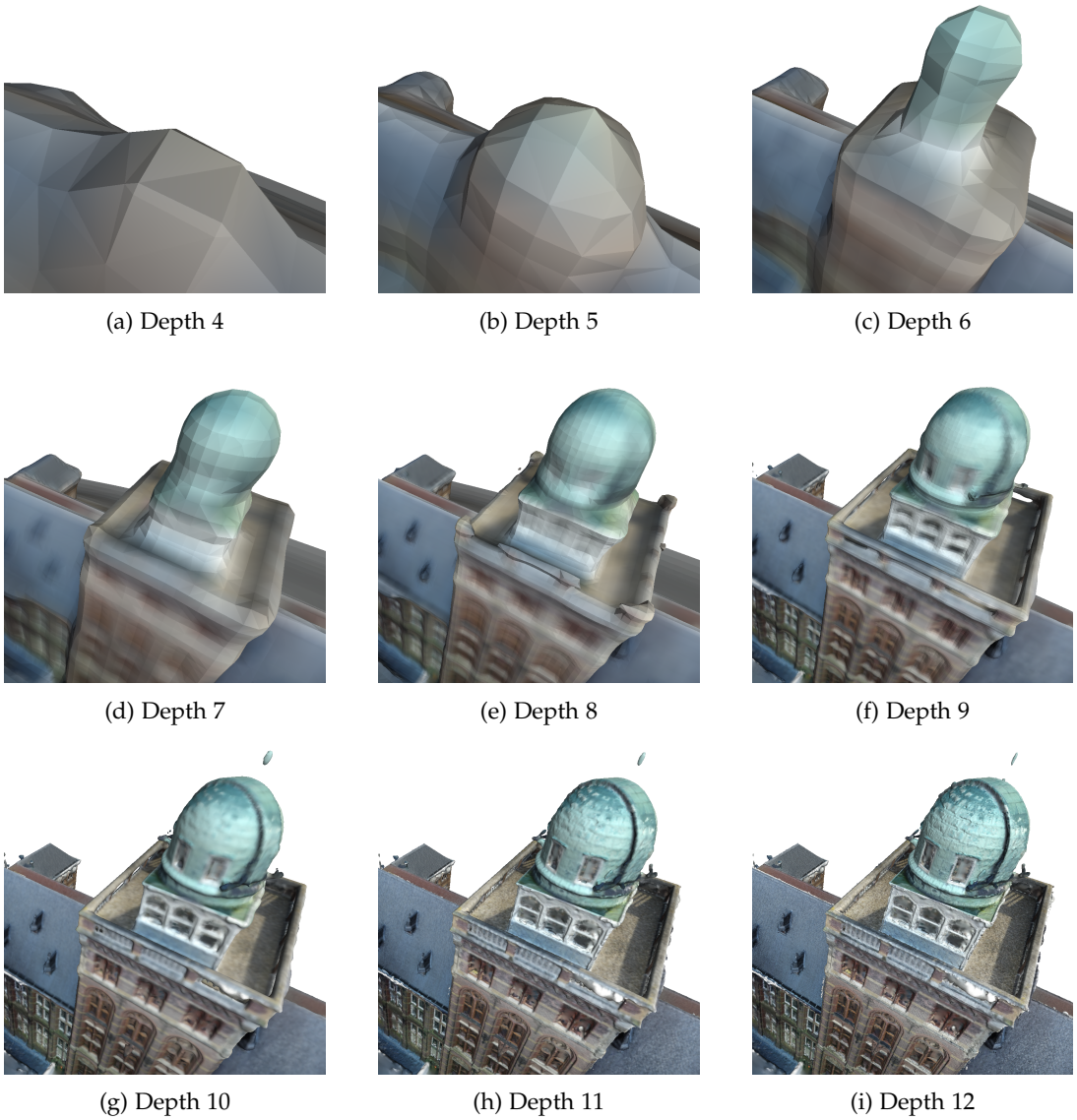


Figure 5.16.: Number of vertices and faces as a function of reconstruction depth (min samples = 10, interpolation weight = 4). The dashed gray line indicates the number of vertices in the original point cloud.



(a) Depth 4 (b) Depth 5 (c) Depth 6

(d) Depth 7 (e) Depth 8 (f) Depth 9

(g) Depth 10 (h) Depth 11 (i) Depth 12

Figure 5.15.: Surface reconstructions for varying reconstruction depth parameters (min samples = 1.5, interpolation weight = 4).

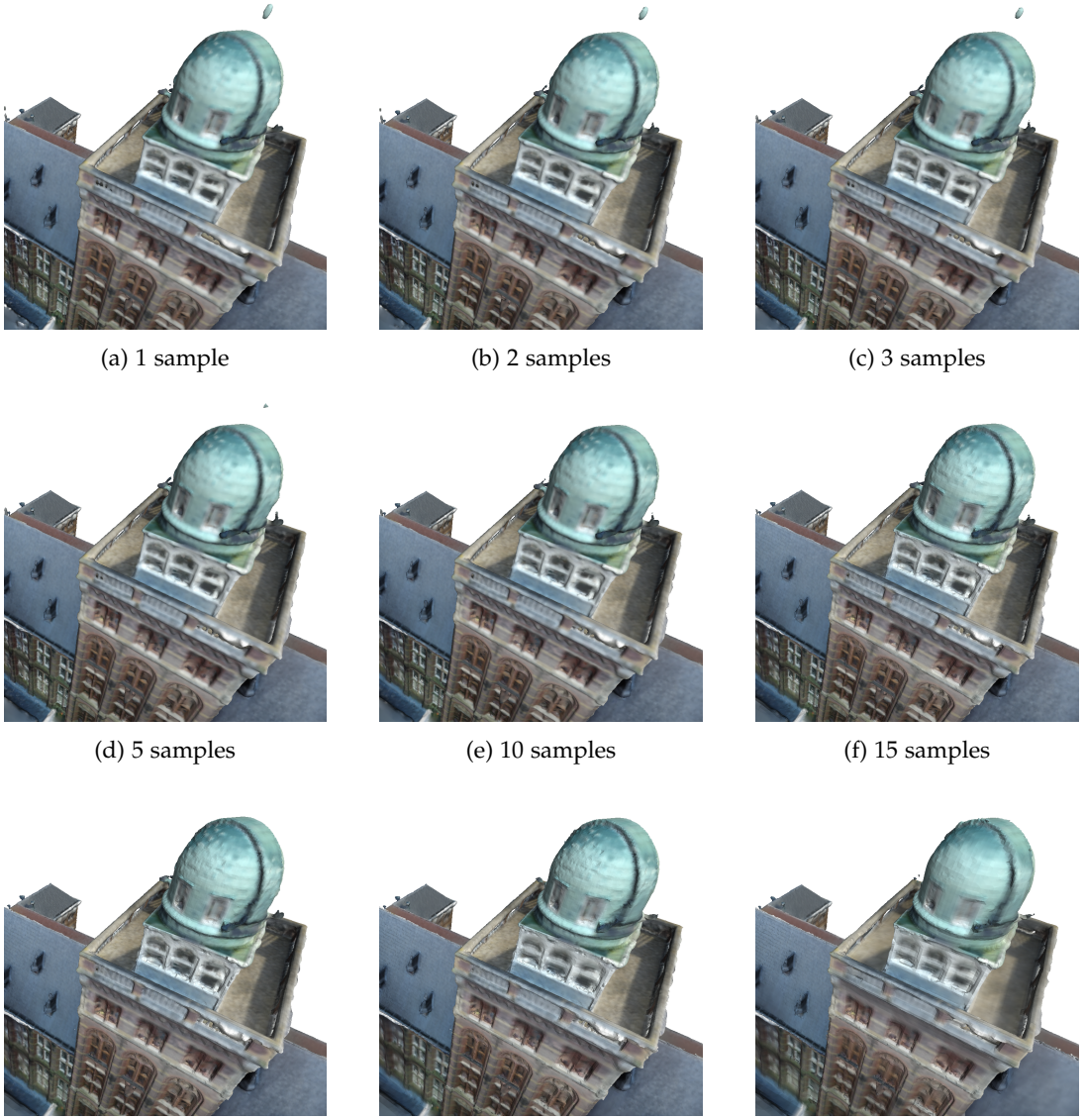
5.4.2. Minimum number of samples

In Figure 5.17, we can visually inspect the different configurations of the minimum number of samples parameter. Table 5.21 and Figure 5.18 show the corresponding number of vertices and faces for each parameter value.

With a small number of samples (1–5), we observe overfitting (with a blob above the dome again) and a higher number of vertices and faces. As the number increases, the mesh becomes smoother. Up to around 25–50 samples, the reconstruction still preserves considerable

detail. However, at 100 samples, there is a noticeable loss of detail.

We want to choose a minimum number of samples that avoids both overfitting with excessive geometric complexity and underfitting with loss of detail. In this case, selecting a value between 5 and 25 appears to offer a good trade-off.



(a) 1 sample (b) 2 samples (c) 3 samples

(d) 5 samples (e) 10 samples (f) 15 samples

(g) 25 samples (h) 50 samples (i) 100 samples

Figure 5.17.: Surface reconstructions for varying minimum number of samples per octree node.

Table 5.21.: Number of vertices and faces for varying minimum number of samples (reconstruction depth = 10, interpolation weight = 4)

Min. Samples	**1**	**2**	**3**	**5**	**10**	**15**	**25**	**50**	**100**
# Vertices	1234454	1217085	1198881	1158782	298373	944975	826041	607630	287818
# Faces	2468562	2434027	2397732	2317656	596828	1890180	1652330	1215448	575809

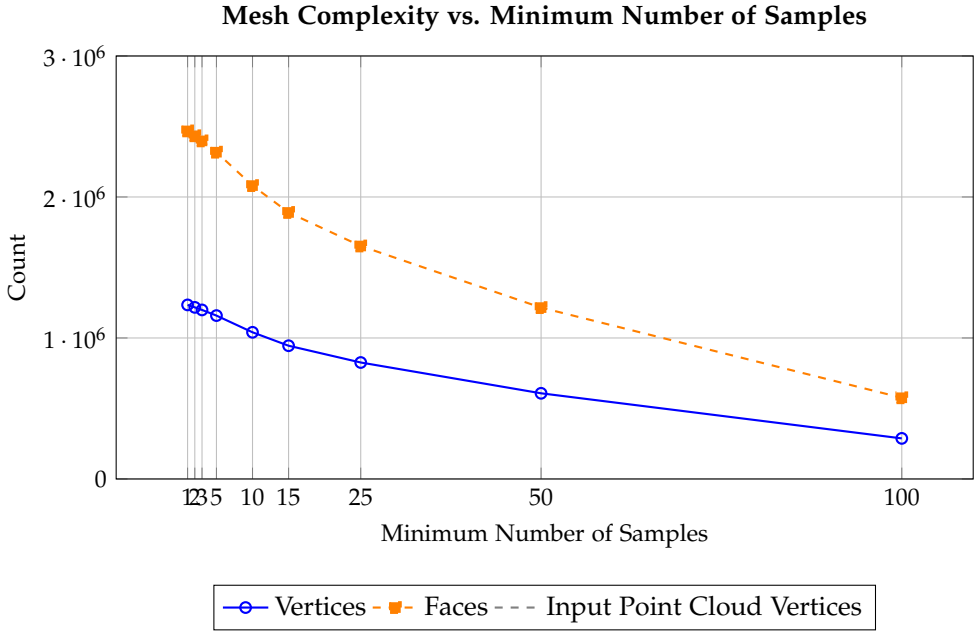


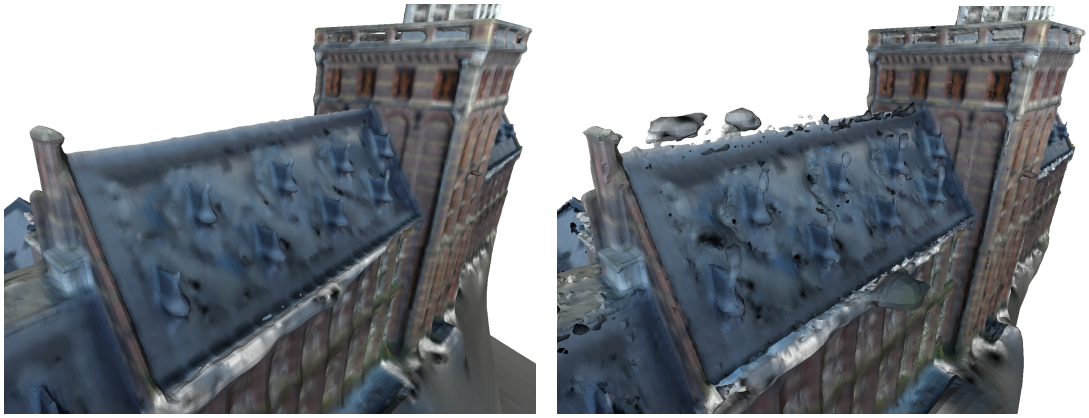
Figure 5.18.: Number of vertices and faces as a function of the minimum number of samples per octree node (depth = 10, interpolation weight = 4). The dashed gray line indicates the number of vertices in the original point cloud.

5.4.3. Interpolation Weight

In Figure 5.20, we can visually inspect the different configurations of the interpolation weight parameter. Table 5.22 and Figure 5.21 show the corresponding number of vertices and faces for each parameter value.

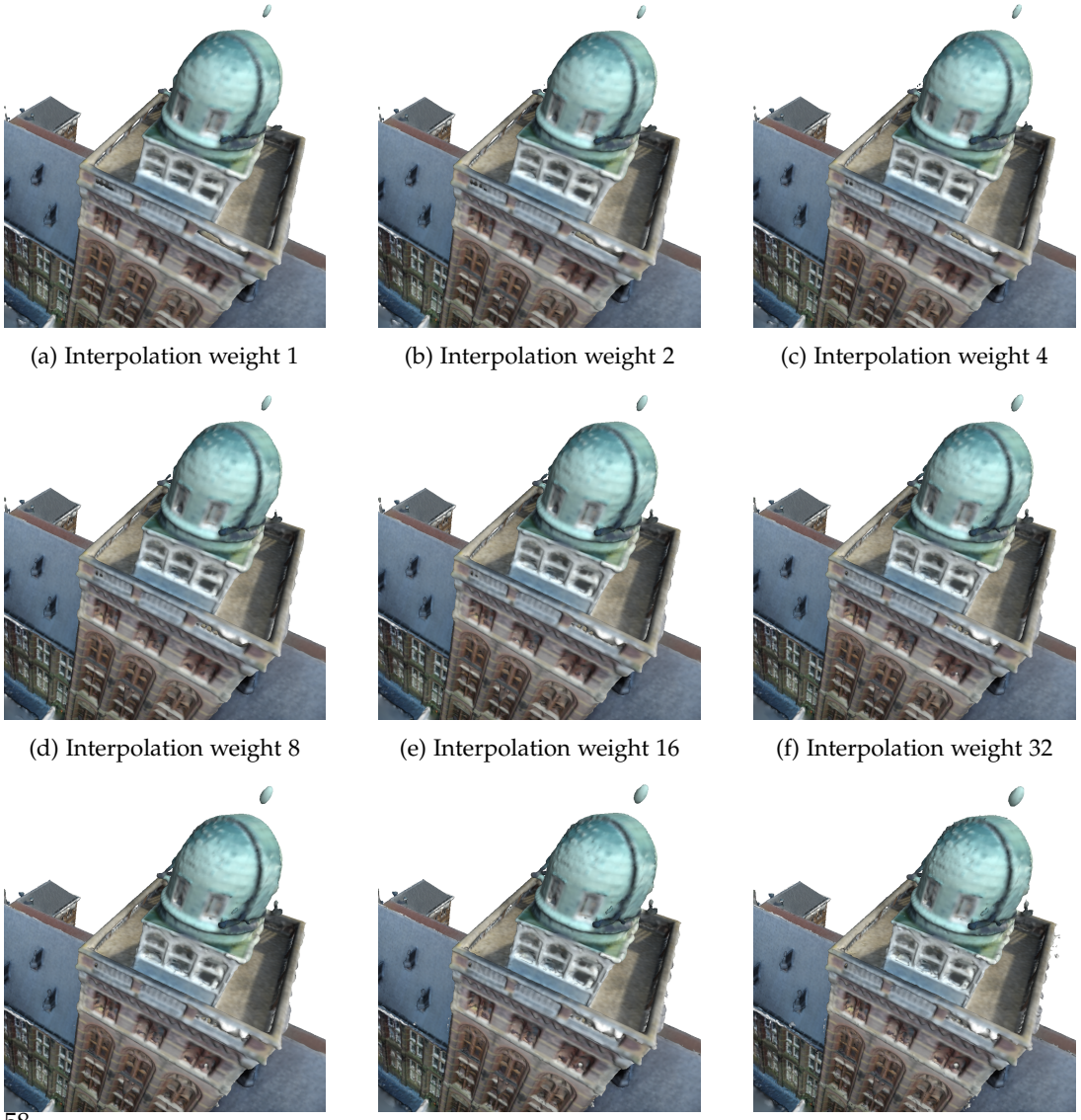
A low interpolation weight smooths the surface, reducing noise and generating a cleaner mesh. In contrast, a high interpolation weight can lead to overfitting, capturing noise from the point cloud. In this case, the lower weight (1) is preferable, producing a smoother and more visually coherent result than the higher weight (50), which introduces visible artifacts.

Figure 5.21 shows that the interpolation weight has a small influence on the number of vertices and faces compared to reconstruction depth and the minimum number of samples. The increase in complexity is only slight as the interpolation weight increases, so the memory cost is not significantly affected.



(a) Low interpolation weight (1) (b) High interpolation weight (50)

Figure 5.19.: Effect of interpolation weight on mesh reconstruction



(a) Interpolation weight 1 (b) Interpolation weight 2 (c) Interpolation weight 4

(d) Interpolation weight 8 (e) Interpolation weight 16 (f) Interpolation weight 32

(g) Interpolation weight 64 (h) Interpolation weight 128 (i) Interpolation weight 256

Figure 5.20.: Surface reconstructions for varying interpolation weight parameters (min samples = 10, reconstruction depth = 10).

Table 5.22.: Number of vertices and faces for varying interpolation weights (reconstruction depth = 10, min samples = 1.5)

Interpolation Weight	1	2	4	8	16	32	64	128	256
# Vertices	1202488	1213238	1225570	1240264	1256398	1275351	1299997	1329035	1364598
# Faces	2404665	2426221	2450962	2480442	2512886	2550864	2600088	2658372	2729727

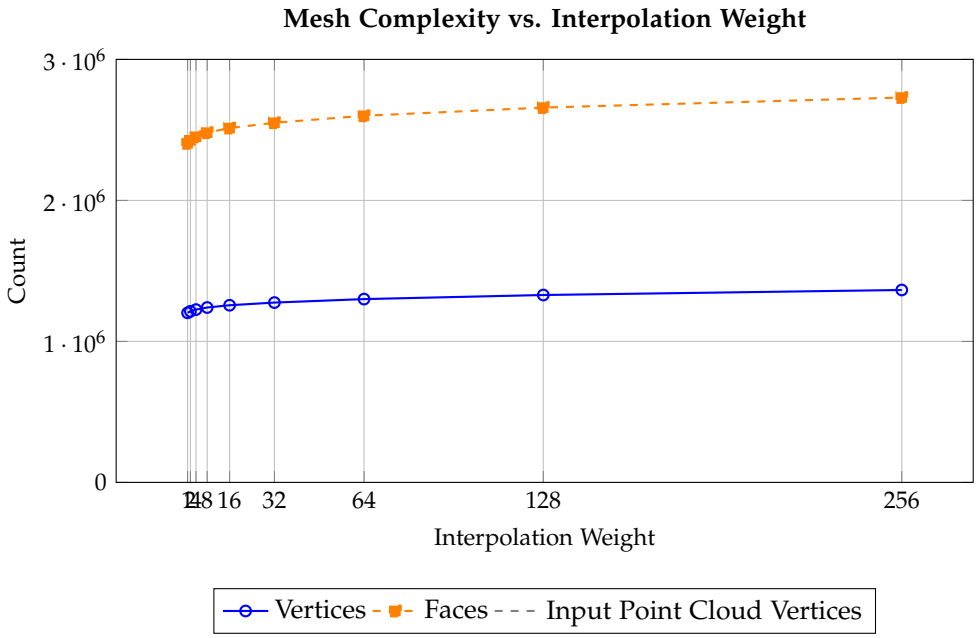


Figure 5.21.: Number of vertices and faces as a function of the interpolation weight (reconstruction depth = 10, min samples = 1.5). The dashed gray line shows the number of vertices in the original point cloud.

5.4.4. Final Poisson mesh reconstruction configuration

Based on this analysis, we chose the following Poisson reconstruction parameters for all point clouds: *reconstruction depth 10*, *minimum number of samples 50*, and *interpolation weight 1*. Depth 10 balances detail and resource usage. A minimum sample count of 50 effectively smooths noise while preserving geometry. Interpolation weight 1 avoids overfitting and artifacts.

The result of this surface reconstruction on the dense point cloud using the full images can be seen in Figure 5.22, which will be used as the ground truth. This configuration was applied to all 27 point clouds (9 thresholds x 3 methods), resulting in 27 meshes.

(a) Ground Truth View 1 (b) Ground Truth View 2

Figure 5.22.: Reference ground truth mesh views used for comparison with the reconstructed meshes.

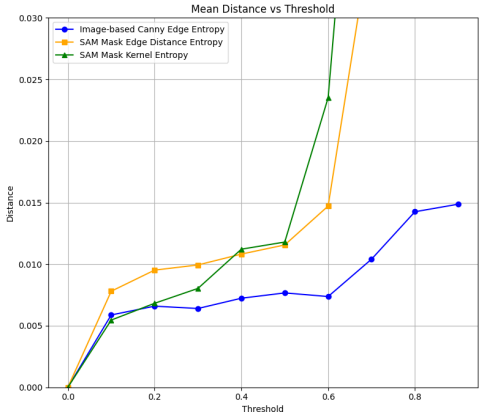
5.5. Mesh quality evaluation

We evaluated three importance-region strategies: SAM kernel entropy, SAM edge distance, and Canny edge distance, across threshold levels ranging from 0.1 to 0.9. Mesh quality was assessed using the mean, median, and Hausdorff distances to a ground truth mesh. To provide additional insight into memory and runtime performance, we also report the number of input point cloud vertices, resulting mesh vertices and faces, and the time required for dense matching.

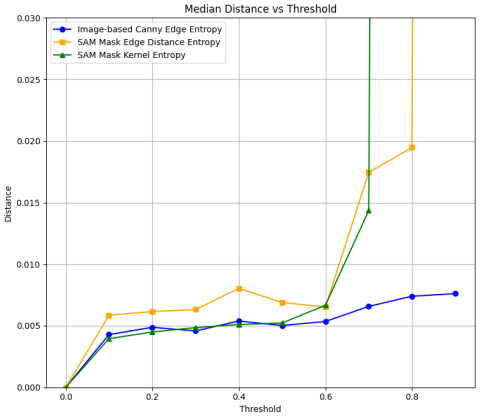
5.5.1. SAM kernel entropy

The reconstruction results for the SAM kernel entropy method are presented in Table 5.23, as well as visualized in Figure 5.23. For the mean reconstruction error, this method closely follows the performance of Canny edge distance up to a threshold of 0.2. Median distance remains comparable to Canny edge performance until threshold 0.5. However, beyond this point, both mean and median errors increase significantly. Hausdorff distance shows a similar trend: relatively stable until threshold 0.5, followed by a sharp rise.

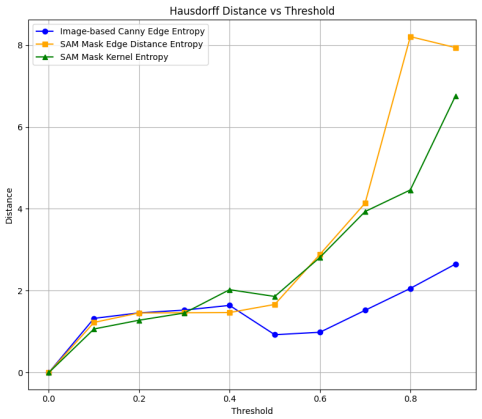
Visual inspection aligns with the observed trends. Figure 5.24 shows mesh reconstructions at increasing entropy thresholds. At threshold 0.4, the overall structure of the mesh is still preserved. However, from threshold 0.5 onward, larger portions of the input images are excluded, leading to visible holes and geometric distortions in the resulting mesh. Figure 5.24e illustrates an thresholded input image at 0.5, highlighting the removal of image regions at higher thresholds.



(a) Mean reconstruction error.



(b) Median reconstruction error.

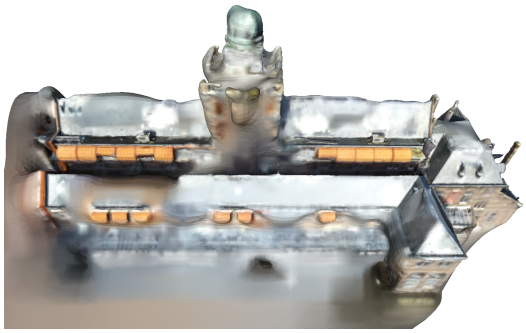


(c) Hausdorff reconstruction error.

Figure 5.23.: Mesh reconstruction accuracy across threshold values for all three methods.

(a) Threshold 0.4

(b) Threshold 0.5



(c) Threshold 0.6

(d) Threshold 0.7

(e) Threshold 0.5 input image

Figure 5.24.: Visual examples of mesh reconstruction at different SAM kernel entropy thresholds. The final image shows a corresponding input image at threshold 0.5, highlighting the loss of information leading to reconstruction artifacts.

Table 5.23.: Mesh quality metrics for SAM Mask Kernel Entropy

Threshold	Mean Dist.	Median Dist.	Hausdorff Dist.	# Vertices (input)	# Vertices	# Faces	Dense match. (time)
0.0	0.0	0.0	0.0	7,564,981	580,800	1,161,704	5h3m
0.1	0.005463	0.003953	1.065408	6,842,222	534,254	1,068,540	4h56m
0.2	0.006826	0.004495	1.279737	6,353,980	510,493	1,020,999	4h56m
0.3	0.008034	0.004849	1.455565	5,663,213	465,265	930,449	4h57m
0.4	0.011218	0.005102	2.025437	3,785,706	326,051	651,946	4h59m
0.5	0.011797	0.005225	1.859387	2,533,572	228,682	457,319	5h0m
0.6	0.023519	0.006664	2.814955	1,157,186	119,809	239,515	4h24m
0.7	0.064950	0.014365	3.935143	442,145	53,199	106,258	3h48m
0.8	0.804311	0.682607	4.461905	53,149	14,115	28,054	2h10m
0.9	2.079102	1.702971	6.759951	1,404	1,096	2,065	0h4m

5.5.2. SAM edge distance

SAM edge distance has the highest distance errors for mean distance up untill a threshold of 0.4, and for 0.6 for median distance. For the Hausdorff distance it is similar to SAM kernel entropy and only becomes higher than that at high thresholds 0.8, 0.9 when accuracy is already low.

SAM mask edge distance-based mesh reconstruction has some similar issues to the kernel entropy-based approach. In general, it finds the important parts of the image for mesh reconstruction pretty well. However, when it misses something, it can lead to noticeable errors in the mesh—see Figure 5.25.

(a) SAM edge distance based mesh reconstruction with threshold 0.6

(b) Example image for SAM edge distance based with threshold 0.6

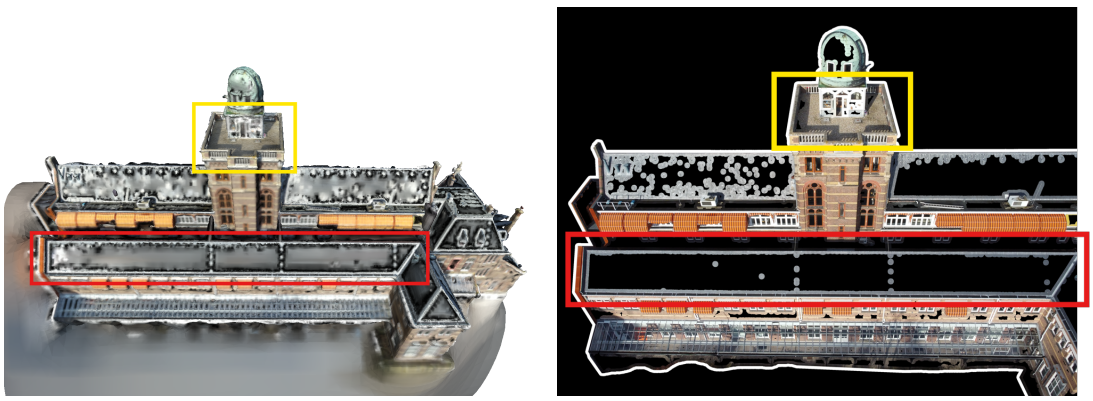
Figure 5.25.: SAM edge detection based mesh reconstruction and its artifacts

Table 5.24.: Mesh quality metrics for SAM edge distance

Threshold	Mean Dist.	Median Dist.	Hausdorff Dist.	# Vertices (input)	# Vertices	# Faces	Dense match. (time)
0.0	0.0	0.0	0.0	7,564,981	580,800	1,161,704	5h3m
0.1	0.007805	0.005860	1.225450	7,339,276	580,580	1,161,275	5h2m
0.2	0.009519	0.006152	1.456943	6,673,821	541,141	1,082,180	5h1m
0.3	0.009938	0.006321	1.461223	5,539,212	453,234	892,342	5h2m
0.4	0.010819	0.006541	1.466547	4,520,885	388,853	777,651	5h0m
0.5	0.011563	0.006887	1.664933	3,522,776	328,475	656,892	4h44m
0.6	0.014719	0.006529	2.887796	2,498,264	259,305	518,417	4h16m
0.7	0.036951	0.017453	4.140401	1,728,857	206,635	413,067	4h7m
0.8	0.058049	0.019476	8.208295	982,411	132,591	265,048	3h30m
0.9	1.369602	1.011981	7.940171	108,029	25,801	51,393	1h31m

5.5.3. Canny edge detection distance

Canny edge detection yields the lowest reconstruction errors overall when compared to the ground truth mesh (see Figure 5.23). Even at a high threshold of 0.8, it still preserves significant mesh detail without introducing major artifacts (see Figure 5.26a). The two primary artifacts observed are mesh-related and texture-related. In Figure 5.26b, the yellow circle highlights an area where many pixels are retained due to strong edge responses caused by surface texture, despite the fact that the region is geometrically flat. This illustrates a common characteristic of Canny edge detection: it can sometimes misclassify textured flat regions as edge-rich. Secondly, in the red square, the reconstruction is generally accurate, though the roof appears unnaturally dark. This is not due to geometric error, but rather a limitation in texture mapping, since that region is poorly visible in the original images, the mesh must infer appearance. Despite this, the underlying geometry aligns closely with the ground truth.



(a) Canny edge mesh at threshold 0.8. The red part shows that

(b) Example image for SAM-based edge distance with threshold 0.8

Figure 5.26.: Canny edge detection based mesh reconstruction and its artifacts

Table 5.25.: Mesh quality metrics for edge detection

Threshold	Mean Dist.	Median Dist.	Hausdorff Dist.	# Vertices (input)	# Vertices	# Faces	Dense match. (time)
0.0	0.0	0.0	0.0	7,564,981	580,800	1,161,704	5h3m
0.1	0.005878	0.004282	1.324103	7,425,792	556,027	1,112,052	5h5m
0.2	0.006593	0.004874	1.459109	7,248,932	559,047	1,118,061	5h2m
0.3	0.006403	0.004580	1.526472	7,070,308	553,533	1,107,082	5h3m
0.4	0.007237	0.005385	1.641304	6,743,709	532,761	1,065,523	5h3m
0.5	0.007668	0.005020	0.924883	6,306,428	509,423	1,018,794	5h2m
0.6	0.007374	0.005348	0.985115	5,996,500	499,133	998,067	4h55m
0.7	0.010405	0.006573	1.521736	5,041,322	464,917	929,439	4h48m
0.8	0.014257	0.007399	2.056343	4,484,251	446,582	893,005	4h38m
0.9	0.014872	0.007610	2.651838	3,326,978	356,556	712,934	4h25m

5.5.4. Results: memory efficiency and mesh quality trade-offs

This section quantitative and qualitative results of our memory efficiency evaluation using the three different thresholding methods: SAM kernel entropy, SAM edge distance, and Canny edge distance.

5.5.5. Memory usage overview

In this analysis, we focus on evaluating memory efficiency based on properties of the output meshes, specifically the number of vertices and faces, as well as the size of the input dense point clouds.

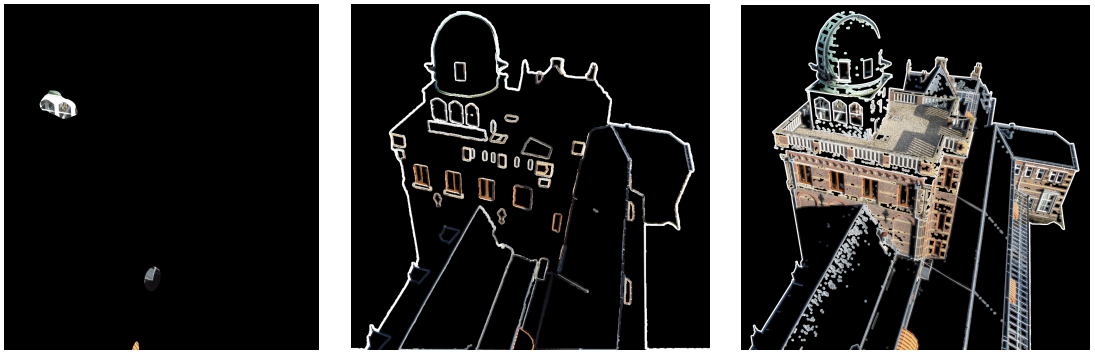
5.5.6. Memory reduction

Table 5.26.: Input point count and memory savings across entropy thresholds for all three methods. Percentage savings are relative to the baseline threshold (0.0).

Threshold	SAM Kernel Entropy		SAM Edge Distance Entropy		Canny Edge Entropy	
	Points	% Savings	Points	% Savings	Points	% Savings
0.0	7,564,981	0%	7,564,981	0%	7,564,981	0%
0.1	6,842,222	9.6%	7,339,276	3.0%	7,425,792	1.8%
0.2	6,353,980	16.0%	6,673,821	11.8%	7,248,932	4.2%
0.3	5,663,213	25.1%	5,539,212	26.8%	7,070,308	6.5%
0.4	3,785,706	50.0%	4,520,885	40.2%	6,743,709	10.9%
0.5	2,533,572	66.5%	3,522,776	53.4%	6,306,428	16.6%
0.6	1,157,186	84.7%	2,498,264	66.9%	5,996,500	20.7%
0.7	442,145	94.2%	1,728,857	77.1%	5,041,322	33.4%
0.8	53,149	99.3%	982,411	87.0%	4,484,251	40.7%
0.9	1,404	99.98%	108,029	98.6%	3,326,978	56.0%

We observe clear reductions in point cloud size as threshold values increase. As shown in Table 5.26, the SAM kernel entropy method achieves the most significant memory savings, reducing input point cloud size by up to 66.5% at a threshold of 0.5 and over 99% at threshold 0.9. In contrast, Canny edge detection retains more input data, resulting in only

a 16.6% reduction at threshold 0.5, and 56.0% at threshold 0.9. SAM edge distance reduces the input point cloud size with 53.4% at threshold 0.5, and similar to SAM kernel entropy to 98.6% at threshold 0.9. A part of this observation can be attributed to how much pixels are retained, which obviously the canny edge detection method retains a lot more pixels with the thresholding, see Figure 5.27.



(a) Example image for SAM-based kernel entropy with threshold 0.9

(b) Example image for SAM-based edge distance with threshold 0.9

(c) Example image for Canny edge distance with threshold 0.9

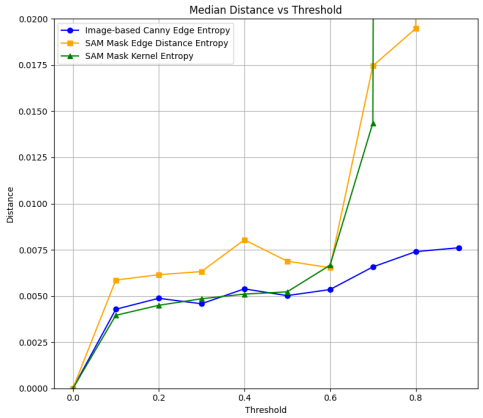
Figure 5.27.: Example images at threshold 0.9 for the three different thresholding methods. The differences in retained pixel regions highlight how aggresively each method filters image content at high thresholds. Canny edge detection preserves significanlty more structure, which helps explian its more stable reconstruction quality at high thresholds.

5.5.7. Quality-to-size tradeoff

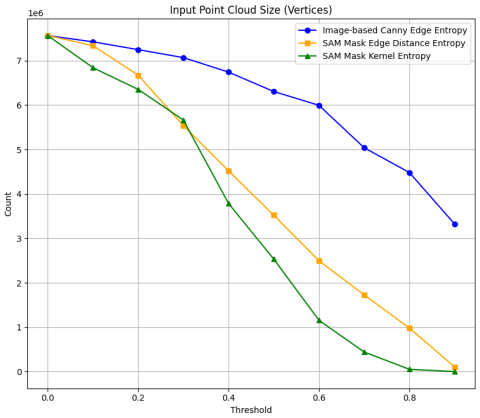
In Figure 5.28a, we see that, as previously discussed, Canny edge distance achieves the lowest reconstruction errors on average. However, as shown in Figures 5.28b and 5.28c, it also produces the largest input point cloud and output mesh sizes compared to the two SAM-based methods.

To better assess memory efficiency, we compute the ratio of reconstruction error to both input and output size, shown in Figure 5.29. This normalization allows for a more meaningful comparison across methods, accounting for both accuracy and data size. We observe that Canny edge detection consistently has the lowest quality-to-size ratio across nearly all thresholds, indicating the most efficient trade-off between geometric accuracy and memory usage. It performs similarly to the SAM kernel entropy method at low thresholds (0.1–0.2), but clearly outperforms both SAM-based methods beyond that point.

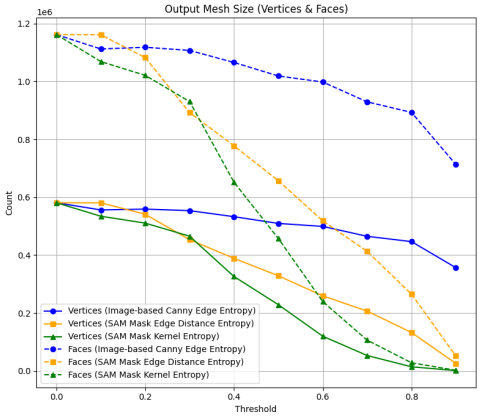
Between the two SAM-based approaches, the kernel entropy method shows better performance than the edge distance method up to a threshold of 0.5, after which it is slightly surpassed. However, since mesh quality degrades significantly beyond 0.5, the comparison is more relevant at lower thresholds, where the kernel entropy method performs best.



(a) Median reconstruction error.

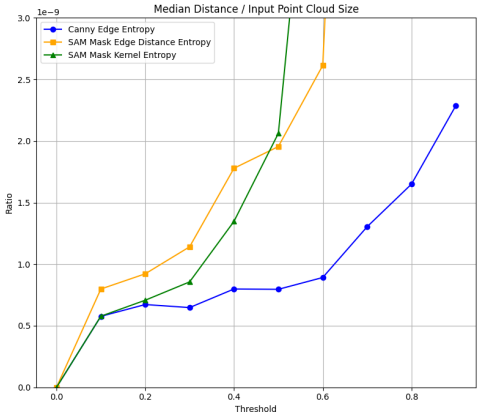


(b) Input point cloud size.

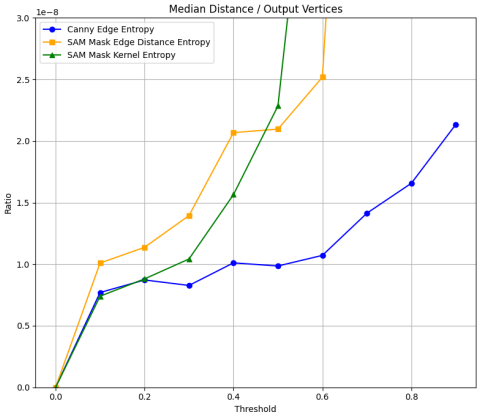


(c) Output mesh complexity.

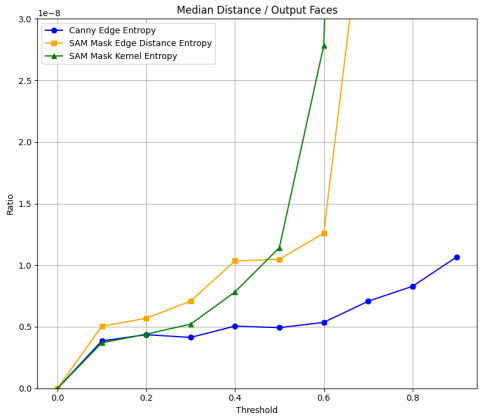
Figure 5.28.: Reconstruction metrics across threshold levels for all three entropy-based threshold methods. (a) Median distance to the ground truth mesh shows how geometric accuracy degrades with increasing threshold. (b) Input point cloud size illustrates how thresholding reduces fused point cloud size, particularly for SAM-based methods. (c) Final mesh complexity, shown by vertex and face counts, correlates with both input sparsity and filtering strategy.



(a) Median Distance / Input Points



(b) Median Distance / Output Vertices



(c) Median Distance / Output Faces

Figure 5.29.: Normalized mesh quality (median distance) per memory-related factor. Lower is better.

5.5.8. Visual comparison

To better understand how thresholding reduces mesh and file size, we include a visual inspection of the meshes behind the numbers in Figure 5.30. This comparison shows the difference between the ground truth mesh and the mesh reconstructed using thresholded images at 0.5 from the SAM-edge distance method.

Even at the overview level, Figures 5.30c and 5.30d clearly show that the face and vertex density in the mesh corresponds to the input image coverage in Figures 5.30a and 5.30b. Areas with fewer input pixels result in visibly lower mesh density. This becomes even more apparent when zooming into a planar roof section (Figures 5.30e and 5.30f): edge detail is largely preserved, while flat regions are simplified significantly.

5.6. Results: runtime efficiency

This section shows the runtime efficiency results through the proposed methods. We compare the dense matching runtimes of reconstructions using thresholded images against a baseline that performs dense matching on the full image set.

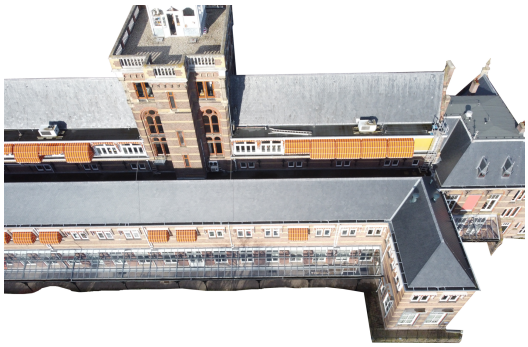
The dense matching runtime results in Table 5.27 and Figure 5.31 show that performance improvements are minimal for thresholds below 0.5. Only the SAM Kernel Entropy method yields a modest 2.3% gain at threshold 0.1, with negligible benefits from the other methods. At threshold 0.5, SAM Edge Distance begins to show moderate improvement (6.3%), but it is not until threshold 0.6 and above that substantial runtime reductions emerge, reaching 24.8% for SAM Kernel Entropy and 18.5% for SAM Edge Distance at threshold 0.7. These improvements closely follow the reduction in the number of registered images during the SfM stage, which directly influences dense matching load.

The Canny edge detection method demonstrates slower runtime improvements, achieving gains of up to 12.5%, likely due to its more conservative thresholding behavior. However, it maintains superior mesh quality at higher thresholds (see Figure 5.31a), underscoring a trade-off between runtime efficiency and geometric fidelity. To more objectively evaluate this trade-off, we compute a quality-to-runtime ratio across thresholds in the following section, as was done for the memory efficiency results.

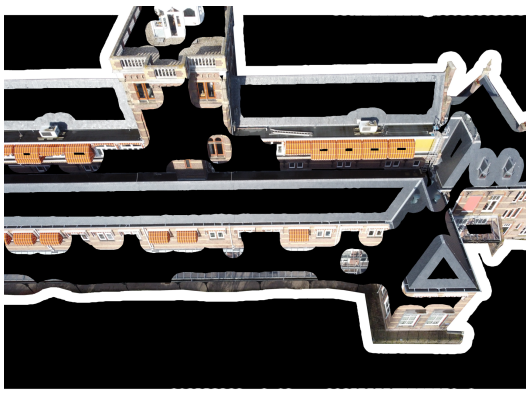
5.6.1. Quality-to-runtime ratio analysis

In this section, we normalize mesh quality by runtime to compute a quality-to-runtime ratio, enabling comparison across methods and threshold levels. The results are shown in Figure 5.32.

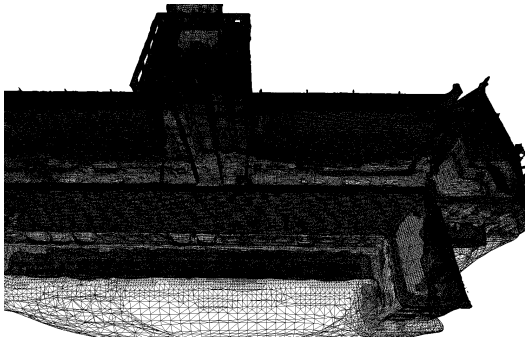
These results reflect trends similar to those observed in the memory efficiency analysis. Both Canny Edge Entropy and SAM Kernel Entropy achieve the lowest quality-to-runtime ratios, indicating more efficient reconstructions in terms of quality per unit time, up to a threshold of 0.5. Beyond this point, the performance of SAM Kernel Entropy deteriorates rapidly, while Canny exhibits a more gradual decline. In contrast, SAM Edge Distance consistently yields higher (i.e., less favorable) ratios across thresholds, only converging with SAM Kernel Entropy after threshold 0.6, where both methods experience significant drops in efficiency.



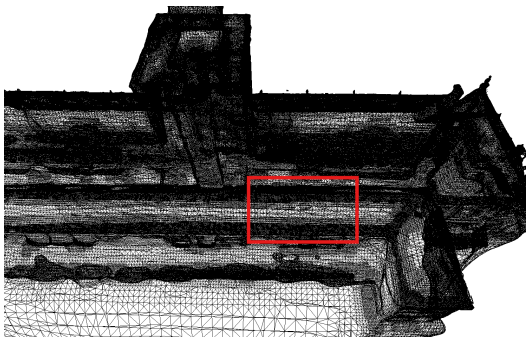
(a) Example input without thresholding (used for ground truth mesh). 580,801 vertices, 1,161,704 faces



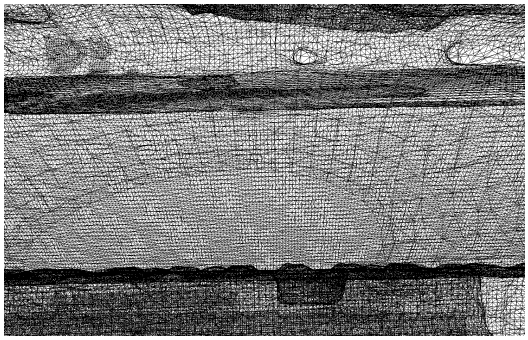
(b) Example image threshold 0.5 SAM-edge distance. 328,475 vertices, 656,892 faces.



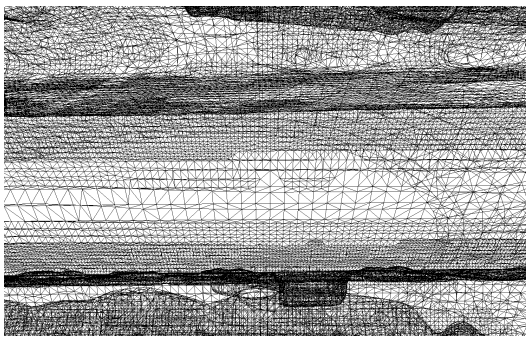
(c) Ground truth mesh



(d) Mesh SAM-edge distance threshold 0.5



(e) Zoomed in part (red) on ground truth mesh

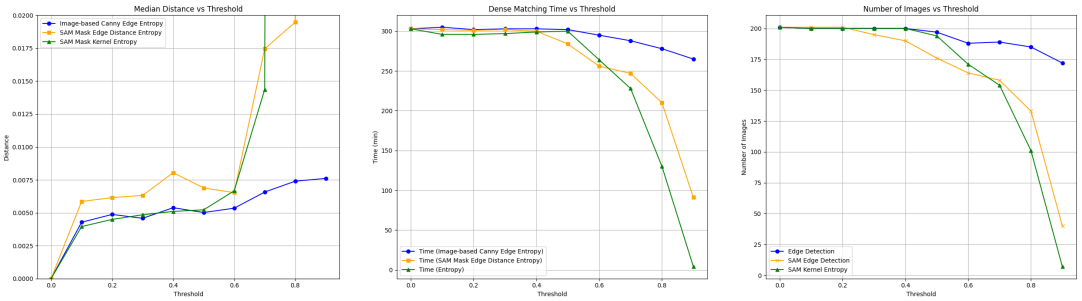


(f) Zoomed in part (red) on mesh threshold 0.5

Figure 5.30.: Visual comparison of reconstruction quality using SAM edge distance thresholding at 0.5.

Table 5.27.: Dense matching runtime and relative savings across entropy thresholds for all three methods. Percentage savings are relative to the baseline threshold (0.0).

Threshold	**SAM Kernel Entropy**		**SAM Edge Distance Entropy**		**Canny Edge Entropy**	
	Time (min)	% Savings	Time (min)	% Savings	Time (min)	% Savings
0.0	303	0%	303	0%	303	0%
0.1	296	2.3%	302	0.3%	303	0.0%
0.2	296	2.3%	301	0.7%	302	0.3%
0.3	297	2.0%	302	0.3%	303	0.0%
0.4	299	1.3%	300	1.0%	303	0.0%
0.5	300	1.0%	284	6.3%	302	0.3%
0.6	264	12.9%	256	15.5%	295	2.6%
0.7	228	24.8%	247	18.5%	288	5.0%
0.8	130	57.1%	210	30.7%	270	10.9%
0.9	4	98.7%	91	70.0%	265	12.5%



(a) Median reconstruction error across thresholds.

(b) Dense matching runtime across thresholds.

(c) Number of registered images across thresholds

Figure 5.31.: Runtime and accuracy metrics for all three entropy-based filtering methods across threshold levels. (a) Median error provides insight into reconstructed mesh quality. (b) Runtime trends indicate performance improvements with increasing thresholds. (c) The number of registered images reflects how many images were used in the reconstruction during the SfM stage.

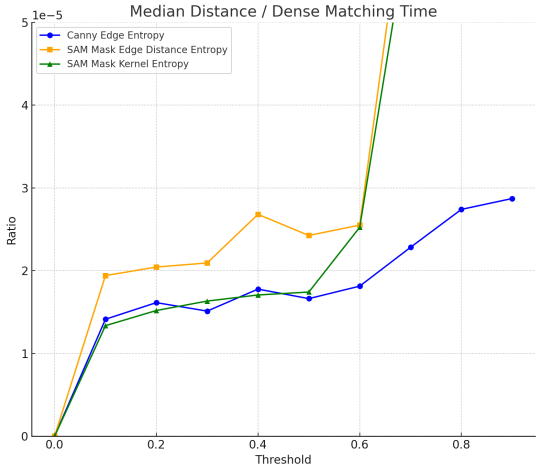
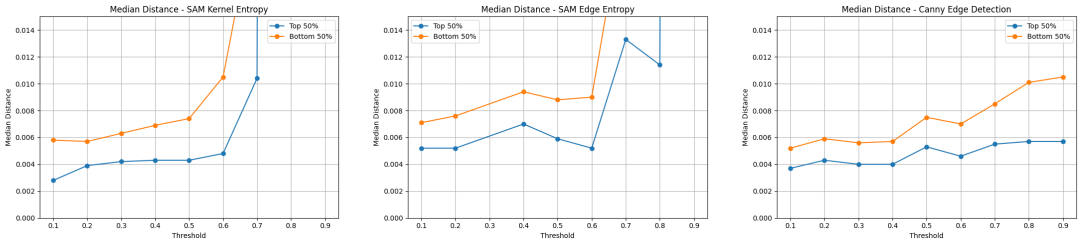


Figure 5.32.: Normalized reconstruction error relative to dense matching time. Lower values indicate more efficient reconstructions in terms of quality per unit time.

5.6.2. Low vs. high density points mesh quality

We split mesh points into top and bottom 50% based on local reconstruction density. Figure 3.10 shows representative examples. We then evaluated how well each entropy strategy performs in sparse vs. dense regions.

According to our hypothesis that importance-based thresholding preserves geometrically informative regions, we expect lower reconstruction error in the top 50% vertex density regions compared to the bottom 50%. Figure 5.33 shows this exactly: across all entropy strategies and thresholds, the median distance to the ground truth is consistently lower for high-density (top 50%) mesh regions. on the other hand, the bottom 50% regions, which correspond to the areas discarded by the thresholding, exhibit higher error, although within acceptable range.



(a) SAM Kernel Entropy (b) SAM Edge Entropy (c) Canny Edge Detection

Figure 5.33.: Median distance comparison for top and bottom 50% mesh density across different entropy strategies.

Finally, if we compare the bottom 50% subsets across all entropy measures (Figure 5.34), we observe that both the SAM kernel and Canny edge methods achieve the lowest reconstruction errors up to a threshold of 0.5. This suggests that these two methods are more effective at discarding low-importance regions without significantly degrading mesh quality

in sparse areas. Beyond this point, however, the error for the SAM kernel increases sharply, indicating that too much useful detail may be discarded at higher thresholds. In contrast, the Canny-based approach maintains a more stable performance, suggesting greater robustness to aggressive thresholding. What is more interesting is that when we normalize this with the input point cloud size in Figure 5.35a, this is the first metric in the whole research where the Canny edge distance is clearly outperformed by the two SAM based methods, especially on lower thresholds.

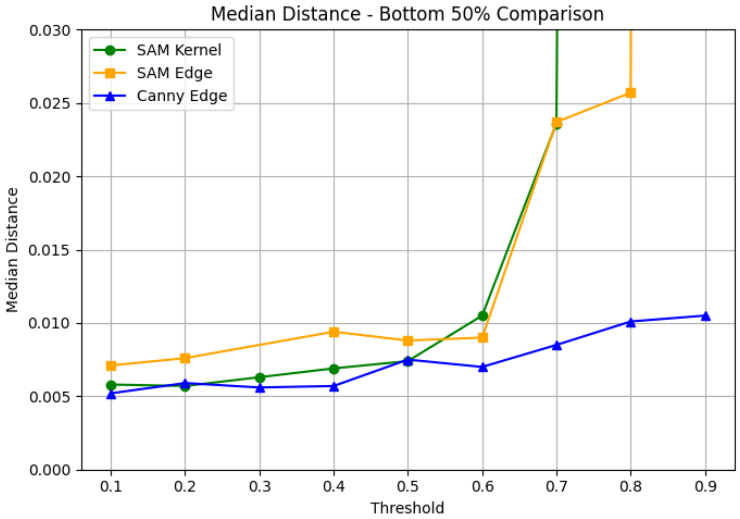


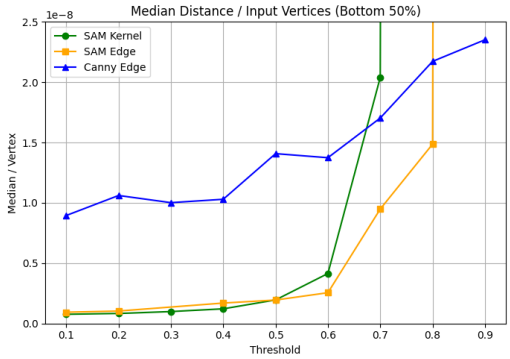
Figure 5.34.: Median distance in bottom 50% density regions across all entropy measures. Canny and SAM kernel perform best up to a threshold of 0.5, with Canny showing more robustness at higher thresholds.

Table 5.28.: Distance metrics for SAM mask kernel entropy

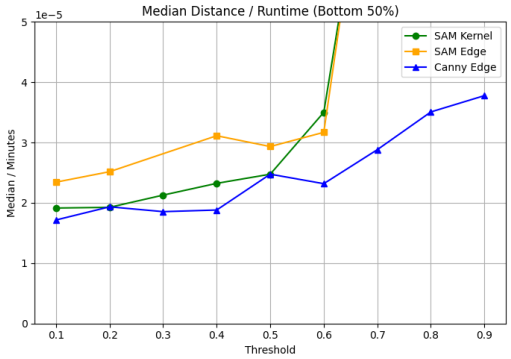
Threshold	Mean	Mean Bottom	Mean Top	Median	Median Bottom	Median Top	Hausdorff	Hausdorff Bottom	Hausdorff Top
0.1	0.0054	0.0064	0.0029	0.0044	0.0058	0.0028	1.0654	1.3544	0.0768
0.2	0.0068	0.0094	0.0040	0.0045	0.0057	0.0039	1.7797	1.5351	0.0190
0.3	0.0080	0.0116	0.0047	0.0048	0.0063	0.0042	1.4556	1.6546	0.0447
0.4	0.0152	0.0198	0.0048	0.0049	0.0069	0.0043	2.1254	2.2613	0.0523
0.5	0.0118	0.0187	0.0049	0.0052	0.0074	0.0043	1.8594	1.9021	0.0539
0.6	0.0235	0.0431	0.0059	0.0067	0.0105	0.0048	2.8150	3.0046	0.0560
0.7	0.0649	0.1277	0.0116	0.0144	0.0236	0.0104	3.9351	4.1836	0.0631
0.8	0.8043	0.9620	0.6515	0.6826	0.7341	0.6731	4.4619	4.1379	1.4781
0.9	2.0791	2.7417	1.9587	1.7030	2.1459	1.8859	6.7600	6.8468	4.5773

Table 5.29.: Distance metrics for SAM edge distance entropy

Threshold	Mean	Mean Bottom	Mean Top	Median	Median Bottom	Median Top	Hausdorff	Hausdorff Bottom	Hausdorff Top
0.1	0.0078	0.0099	0.0058	0.0059	0.0071	0.0052	1.2255	1.4129	0.0276
0.2	0.0095	0.0121	0.0068	0.0062	0.0076	0.0052	1.4569	1.4569	0.0707
0.4	0.0108	0.0145	0.0075	0.0080	0.0094	0.0070	1.4665	1.9761	0.0253
0.5	0.0116	0.0172	0.0062	0.0069	0.0088	0.0059	1.6649	1.8767	0.0324
0.6	0.0147	0.0236	0.0058	0.0065	0.0090	0.0052	2.8878	2.5247	0.0374
0.7	0.0370	0.0561	0.0180	0.0175	0.0237	0.0133	4.1404	4.0636	0.1184
0.8	0.0580	0.1023	0.0147	0.0155	0.0257	0.0114	8.2083	7.5961	0.1078
0.9	1.3696	1.8178	0.8940	1.0120	1.5476	0.6445	7.9402	7.9402	3.5129



(a) Median distance divided by input point cloud size for the bottom 50% of mesh vertices.



(b) Median distance divided by dense matching runtime for the bottom 50% of mesh vertices.

Figure 5.35.: Efficiency analysis of mesh quality: normalizing the median distance error by (a) input point cloud size and (b) runtime, across entropy-based and edge-based thresholding methods.

Table 5.30.: Distance metrics for image-based canny edge detection

Threshold	Mean	Mean Bottom	Mean Top	Median	Median Bottom	Median Top	Hausdorff	Hausdorff Bottom	Hausdorff Top
0.1	0.0059	0.0079	0.0038	0.0043	0.0052	0.0037	1.3241	1.2759	0.0159
0.2	0.0066	0.0090	0.0044	0.0049	0.0059	0.0043	1.4591	1.5225	0.0224
0.3	0.0064	0.0087	0.0041	0.0046	0.0056	0.0040	1.5265	1.4516	0.0183
0.4	0.0064	0.0087	0.0041	0.0046	0.0057	0.0040	1.6671	1.4229	0.0336
0.5	0.0077	0.0100	0.0054	0.0060	0.0075	0.0053	0.9249	1.3412	0.0403
0.6	0.0074	0.0097	0.0050	0.0053	0.0070	0.0046	0.9851	0.6777	0.0561
0.7	0.0104	0.0137	0.0069	0.0066	0.0085	0.0055	1.5217	1.7268	0.0631
0.8	0.0143	0.0208	0.0080	0.0074	0.0101	0.0057	2.0563	2.5397	0.0495
0.9	0.0149	0.0221	0.0075	0.0076	0.0105	0.0057	2.6518	2.5519	0.0473

6. Discussion

6.1. Segmentation using SAM in oblique aerial imagery

This section evaluates the effectiveness and limitations of applying SAM for segmenting buildings in oblique aerial imager, addressing the subquestion *"How can SAM be applied to accurately and efficiently segment individual buildings in oblique aerial imagery?"*

We constructed a pipeline that used the orientation data from the oblique images in combination with BAG objects to segment buildings with SAM. The results showed that it was possible to segment buildings within an image, generating datasets with building masks linked to specific BAG objects. The box prompt mode of SAM proved effective in many cases. However, we observed that the SAM confidence score could be overly optimistic, sometimes assigning high confidence to objects not part of the building.

The goal of this stage was to support the rest of our research by generating building masks from oblique aerial images, and using those for 3D reconstruction. We applied our approach to several buildings and attempted reconstruction by generating sparse point clouds. However, as shown in Section 5.3, none of the aerial oblique datasets produced usable reconstructions.

There are possible improvements to this pipeline. We experimented with prompting using foreground and background points around the building (see Figure 3.4c). To automate this, the building footprint needs to be reprojected into the oblique image spaces, and it has to align very precisely. If it doesn't, background points can distort the segmentation. Because of this, automation wasn't an option for now, and we only tested it manually. But when we did, it gave very good results.

However, even when we manually segmented two buildings, resulting in near-perfect masks, reconstruction still failed. This suggests that the issue lies in the information loss from using segmented masks instead of full images. If we want to extract buildings from oblique imagery, we may need to first reconstruct from the full image set, and only then perform segmentation on the resulting 3D point cloud, using real-world coordinates. Figure 6.1a shows the dense point cloud generated from all full oblique images containing BAG object 0344100000157740. As shown in Figure 6.8b, this results in a much more complete point cloud at the building level. However, this approach diverges from the original research goal of reconstructing buildings using only the most relevant input data, as it involves using full images.

This leads into the question of scalability. Will this approach also work for other datasets, and is it adaptable to different scenarios? In principle, any oblique image can be segmented with SAM if one provides the appropriate prompt coordinates. However, to automate segmentation at scale across hundreds of images, we need metadata similar to the BAG: world coordinates of the target objects and image orientation data for reprojection. If those are available, the pipeline is scalable.

(a) Dense matched point cloud for all images containing BAG object 0344100000157740. The building position is shown in red.

(b) Extracted points for BAG object 0344100000157740.

Figure 6.1.: Performing dense reconstruction on the full oblique images and then extracting the target building results in a more complete point cloud than reconstructing directly from segmented masks.

That said, there are challenges depending on the dataset. We tested on the city of Utrecht, where the dataset contains mostly low-rise buildings, and only one remotely tall building (see Figure 3.5a). In a city like Manhattan, reprojection of building footprints would be far less reliable. Occlusion by other buildings is a major issue, and a reprojected footprint could easily end up on the side of an entirely different structure.

Although there is room to improve the prompting strategy, for this dataset it wouldn't make a major difference, since even perfect masks didn't lead to usable reconstructions. Instead, future work could explore reconstructing from full oblique images first, and only segmenting buildings afterward in 3D space. In this case, SAM wouldn't be involved anymore. Another possible direction is research into occlusion detection, particularly to ensure that the building being prompted is not partially or fully occluded by high-rise structures. This could support more accurate projection in dense urban areas.

In summary, while SAM can segment buildings from oblique images with reasonable accuracy using box prompts, the lack of reliable reconstruction from these masks highlights limitations in using segmented oblique imagery directly for 3D modeling.

6.2. Using SAM to identify geometrically important image regions

We also explored whether SAM could assist in identifying the image regions that matter most for producing high-quality 3D meshes, addressing the question: *"How can SAM be used*

to identify image regions that are most important for achieving high-quality 3D mesh reconstruction?" The zero-shot segmentation mode from SAM sparked our interest to see if its mask predictions could highlight areas of geometric importance. Our assumption was that regions with many segments likely correspond to parts of the image with more geometric detail of the building. We tried to capture this by applying a kernel in a convolution over the segmented image. We also assumed that if SAM would segment objects of the building, such as windows, doors, or parts of the roof, areas near the edges of those masks would represent important features for the geometric construction of the mesh.

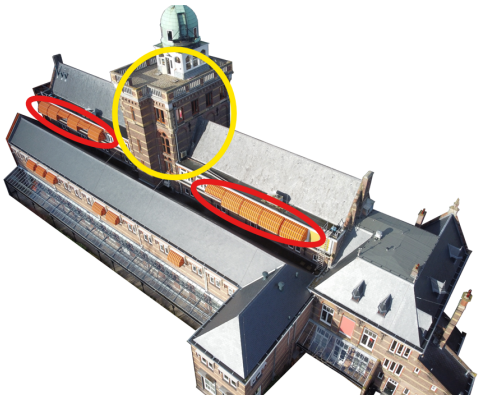
Our translation from those assumptions to a way to pick those elements took place by making the sliding kernel approach using the entropy formula on the segmented masks, and a decaying edge distance function from the mask edges (see Section 3.6). We saw that they were able to generate qualitatively good meshes at low thresholds of those importance maps, and thereby reduce memory usage. While this was promising, it was outperformed by the image-based Canny edge detection method on nearly all memory efficiency metrics. Even after accounting for the fact that the Canny-based edge distance retained more of its pixels, the kernel approach only matched its performance at a few threshold values. Still, at some low thresholds, especially in the kernel-based method using SAM, it performed about as efficiently as the Canny edge distance method. This suggests that SAM does hold some potential for identifying image regions that are most important for achieving mesh quality in reconstruction.

The main reason for this difference is that SAM is less consistent and less evenly distributed over the image compared to the Canny edge distance method. If we take a look at Figure 6.2, we can see that in Figure 6.2a it captures quite a lot of detail. However, in the red circles in Figure 6.2b, there are around 10 sunscreens, of which only two are segmented. Additionally, it segments the area in the yellow circle as a whole, while it clearly features some important edges for meshes. Thus, although it is quite good in segmenting parts of the image, overall mesh quality, as we show in the results, suffers from the missing parts. We can see the difference in importance maps for SAM kernel entropy in Figure 6.2c and Canny edge distance in Figure 6.2d. The Canny edge distance importance map shows a much denser detection of important areas, whereas the SAM-based map is more sparse. This behaviour is visible throughout the datasets. Even after tuning SAM's parameters to improve segmentation quality and mask coverage, this uneven and sometimes sparse segmentation remained a limiting factor. We explored several configurations of the `SamAutomaticMaskGenerator` to strike a balance between segmentation detail, runtime, and the number of generated masks (see Section 5.2).

In the end, we only tested these two SAM-based methods and the Canny edge detection method as an alternative for comparison. Of course, these two methods may be either suboptimal or actually quite effective, more variants should be explored. This could also mean exploring different ways to incorporate SAM, or maybe combining the options we presented using weighted averages for the importance maps.

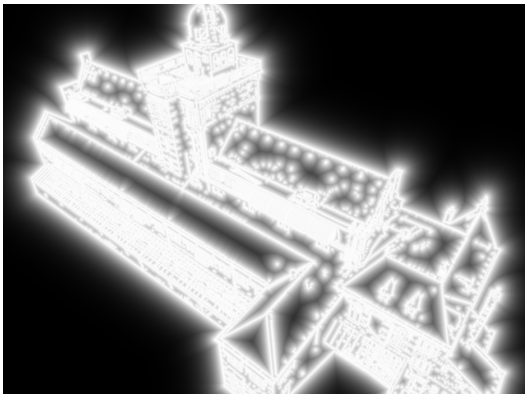
The main limitation here is: before we know how well an importance map operates, we have to do the dense matching and mesh reconstruction, which can take a lot of time. Then we need to do that for all kinds of different methods and threshold values. This bottleneck in the pipeline limits the swift exploration of new methods, or for example, training a neural network. But this is exactly where future work can build: this project lays the foundation for a workflow that connects mesh quality to regions in the images. It must now be worked on further by optimizing the framework for easier and quicker testing, so that you can

(a) SAM-based segmentation output used to guide entropy filtering.



(b) Corresponding mesh reconstruction highlighting filtered high-entropy regions.

(c) Importance map derived from SAM mask contours and edge distance.



(d) Importance map based on Canny edge detection and distance transform.

Figure 6.2.: Comparison of segmentation-driven and edge-driven entropy estimation methods. Top row: SAM-generated segmentation and its influence on mesh detail. Bottom row: Image-space entropy maps used to guide selective reconstruction, based on edge and segmentation mask distance metrics.

analyze those results and see a direction to go in, or perform a lot more tests with different variants.

Another limitation of our results we should discuss is that these are results for only this building and image dataset. We can conclude that, for example, the SAM kernel entropy-based approach with threshold 0.3 works very well here, and that we save up to 2.0% in time and up to 25.1% in memory, while still maintaining good mesh quality. However, it is hard to extrapolate that to other datasets because it is so specific to this building and the way the images are shot. If we perform this on another building, as we saw with a lot of the aerial imagery in Section 5.3, this might not even make a good sparse reconstruction. Or for another dataset, we might employ threshold 0.5 or 0.6 and have the same mesh quality while achieving much higher savings. It remains dataset-specific.

6.3. Assessing mesh quality of selected image regions

To address the question, *"How can the effectiveness of image region importance-driven resource allocation be evaluated in terms of mesh quality and computational efficiency?"* we linked the selected image regions directly to the resulting mesh. Specifically, we generated meshes from the selected image areas and compared them to the ground truth using distance metrics. By applying different threshold levels, we obtained datasets with varying sizes of pixel regions, where higher thresholds corresponded to fewer but more important pixels according to the entropy method. Since the ground truth mesh represents the target quality, it served as a natural reference for assessing our improvements.

We explored Poisson surface reconstruction to create a visually detailed mesh that didn't take too much runtime while remaining smooth and not overfitting to the points. This gave us a good ground truth to compare other meshes to.

We then used the mean, median, and Hausdorff distances to capture the distances between the points of the evaluated mesh and the nearest face of the ground truth. In the end, we discussed most of our results in the results section based on the median distance. Because the Poisson surface reconstruction tries to create a watertight mesh around the points, it almost always produced a kind of rounded finish at the bottom of the mesh, as shown in Figure 6.3. This influences the mean and Hausdorff distance the most. For the mean, it introduces outliers because points on that part of the mesh that are evaluated against the closest face of the ground truth will inevitably show greater distances, thus skewing the mean. For the Hausdorff distance, it is even worse: since it finds the maximum of all minimum distances, this value will almost always lie in the part of the mesh that contains the bottom artifact.

This part of the pipeline currently relies only on median distances, which could be improved in the future. For example, instead of computing distances globally, one could introduce region masks that exclude known artifacts, such as the underside of the mesh. Another option would be to compute an importance value for the points being evaluated and then restrict the evaluation to the top 75%, for instance. This would allow the evaluation to focus only on meaningful geometry and reduce the influence of non-informative regions.

In this light we also explored the low vs high density points, with the hypothesis that high density points belonged to the parts that are retained in the images, and the low density points the parts that are removed by thresholding. These results give a more detailed view

Figure 6.3.: Mesh from the SAM kernel entropy method with threshold 0.3, showing the underside of the mesh clearly as an artifact

of how thresholding affects different parts of the mesh. The breakdown into high and low density regions helps to isolate where the reconstruction actually degrades and whether that matters. The fact that the bottom 50 percent of points, from sparser and less informative areas, still show good accuracy up to a certain threshold suggests that the method works as intended. It filters out mostly redundant regions without harming quality too much. Especially when normalized for point cloud size, the SAM-based methods show real potential and even outperform the Canny edge method at lower thresholds.

This normalization is necessary because, in our pipeline, the three methods are thresholded over a range from 0.1 to 0.9. However, effectiveness cannot be judged solely by distances to the ground truth. This is due to the different nature of the importance maps, where each method thresholds pixels more or less aggressively. For example, Canny edge detection retains more pixel data, which explains its higher stability at higher thresholds. To account for this, we post-processed and normalized results by dividing by input/output point cloud mesh size or runtime.

To make comparisons more balanced, future work could improve this by equalizing the importance maps during thresholding. Instead of applying fixed value thresholds, a fixed percentage of the normalized entropy map could be retained, such as selecting pixels within a range from 10 to 90 percent. This approach would ensure each threshold level corresponds to a consistent portion of selected pixels across methods, making comparisons fairer and easier to interpret.

6.4. Impact of selective image masking on 3D mesh reconstruction quality and computational efficiency

In this section, we look at the question: *"How does selective masking of images based on importance threshold affect the quality and efficiency of 3D mesh reconstruction across different building dataset?"* The goal was to explore whether focusing only on the most relevant image regions could reduce the computational cost of reconstruction, without compromising the quality of the resulting meshes. While the initial plan was to evaluate this across multiple datasets, in

the end we focused on the Geodelta drone dataset, since it was (next to the bouwpub dataset) the only one where the results were consistent enough to draw meaningful conclusions.

As discussed earlier, particularly at lower thresholds, good quality meshes can still be obtained despite removing less important image regions through thresholding. Although we initially expected SAM to perform very well in predicting regions important for mesh reconstruction, it was outscored on many metrics by the simpler Canny edge detection distance strategy. Mesh quality remained generally stable up to a threshold of 0.4 for both SAM-based methods, but it dropped off more sharply for these methods compared to the Canny edge distance method. These results were influenced by the number of retained pixels in the input images, which was higher for the Canny edge method. Even after normalizing for input point cloud size and runtime, Canny edge had more efficient ratios except at thresholds below 0.2. The SAM methods only showed higher quality when focusing on the sparse parts of the mesh relative to the input point cloud size.

For the efficiency evaluation part, we look into file size reduction and runtime reduction. In the results, we see that we can achieve decent reductions in file size and slight reductions in runtime. Even when looking at the low-thresholded images in the 0.1 to 0.2 range, we observe significant size reductions—up to 16.0% with the SAM kernel entropy method, and 11.8% for the SAM edge distance method. For dense matching, we see runtime reductions of up to 2.3% at the 0.2 thresholds across all methods. Even these small reductions, when applied to good mesh results, can lead to considerable savings if extrapolated to neighborhood or city-scale datasets.

However, it is important to note that the full processing pipeline generates additional intermediate data, such as segmentation masks, entropy maps, and thresholded image subsets for the entire dataset. An example of this data flow (for threshold = 0.4) is shown in Figure 6.4. While this intermediate data can be safely discarded, either before mesh generation (such as SAM-based building segmentation and entropy maps) or immediately after (such as the thresholded image dataset), it is still something to consider when evaluating the overall efficiency of the pipeline.

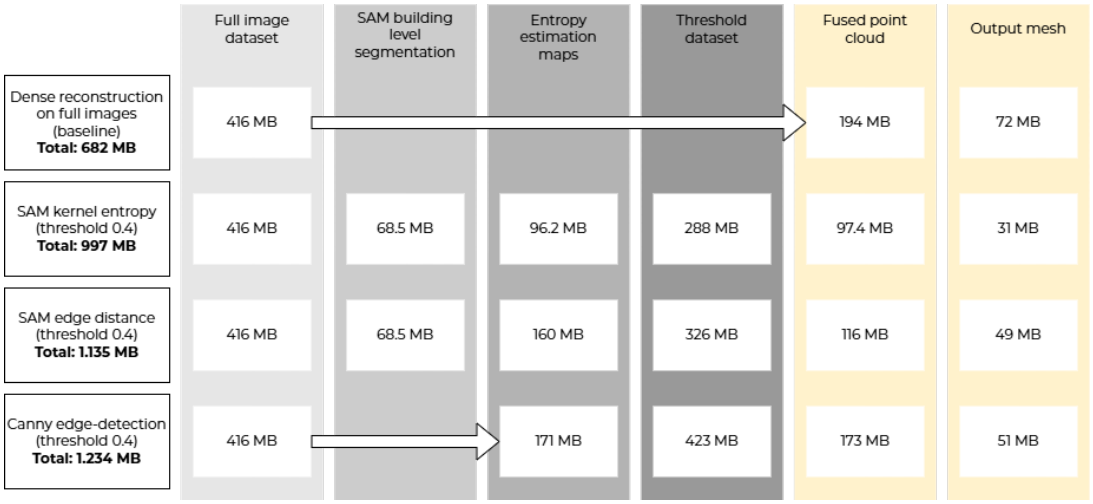


Figure 6.4.: Overview of the memory usage pipeline during mesh reconstruction. While the full pipeline consumes more memory than dense reconstruction alone (682 MB), SAM Kernel Entropy uses 997 MB, SAM Edge Distance 1135 MB, and Canny Edge Detection 1234 MB. However, all intermediate data can be discarded directly after use. Therefore, our analysis focuses only on the datasets highlighted in yellow.

For the runtime efficiency part, we compared the dense matching time of our thresholded image datasets across all three methods against the original dense matching time.

We expected to see more runtime improvements by removing parts of the images. However, the dense matching time remained roughly the same, around five hours, compared to the ground truth dense matching time for all three methods. Only when the threshold reached levels that no longer produced quality meshes (0.6 for SAM kernel entropy, 0.5 for SAM edge distance, and 0.7 for Canny edge distance) did the runtime start to decrease noticeably.

As with the memory efficiency results, we focused in the results section only on this isolated dense matching step. Looking at the complete runtime, shown in Figure 6.5 for an example with threshold 0.4, we see that to fully understand the efficiency gains, the full context must be considered. While dense matching is a major component, additional overhead comes from upstream processes like SAM-based segmentation, importance map generation, and image thresholding. Unlike the full memory pipeline, these steps cannot be discarded and are required before dense matching. Therefore, overall processing time might not actually improve despite some runtime savings during dense matching itself.

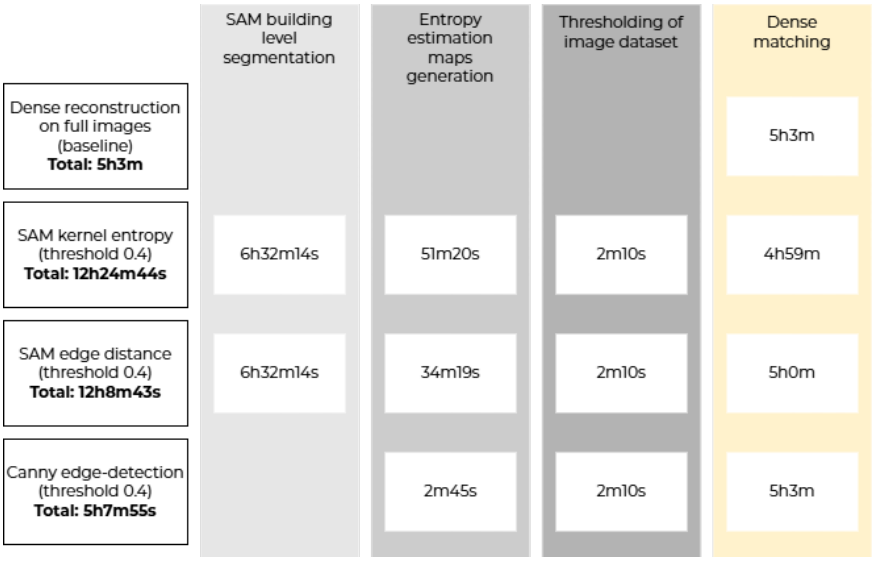


Figure 6.5.: Overview of the runtime pipeline during mesh reconstruction, illustrated for threshold 0.4.

For efficiency, we should therefore look into methods that can not only produce accurate importance maps, but also do so quickly. SAM clearly takes a significant amount of time in the full processing pipeline, while we see that Canny edge detection requires only a fraction of that time and still produces good results.

We can conclude that, in terms of memory efficiency, excluding the intermediate data generated during processing allows for noticeable savings when using the SAM importance maps. However, regarding runtime efficiency, this selective masking approach does not provide improvements; in fact, it tends to increase overall processing time due to the overhead from upstream steps like segmentation and importance map generation. Therefore, while SAM-based methods show promise for reducing memory usage, optimizing runtime remains a challenge and may benefit from faster or more lightweight importance estimation techniques.

6.5. Scalability to full nadir and oblique datasets

To test whether our importance region-based thresholding approach could be applied directly to full oblique and nadir datasets, we moved beyond the controlled setting of the drone images over the Geodelta office. Figure 6.6 shows that, for building datasets, applying a Canny edge distance threshold of 0.7 still results in a reasonable mesh. The structure is preserverd, and only some texture detail is lost in areas where the pixels are discarded.

Figures 6.7 and 6.8 show that applying the same approach to a full oblique or nadir dataset does not produce the desired results. This is mainly because of how the important parts in the image are determined. On the building-level scale, it is much easier to distinguish parts that are geometrically important than at the city-level scale, where the distinction of important parts plays out on a much smaller scale in the image. With the current approach,

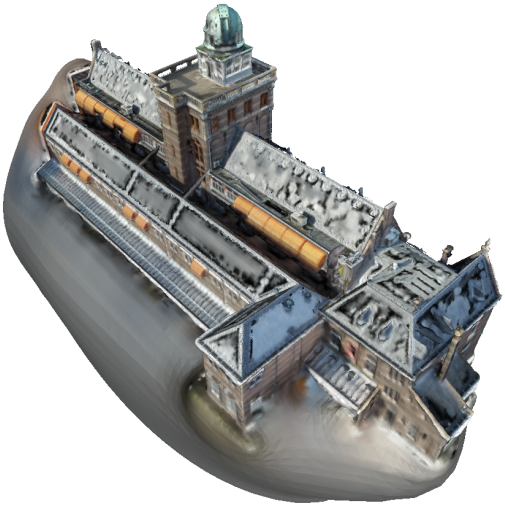
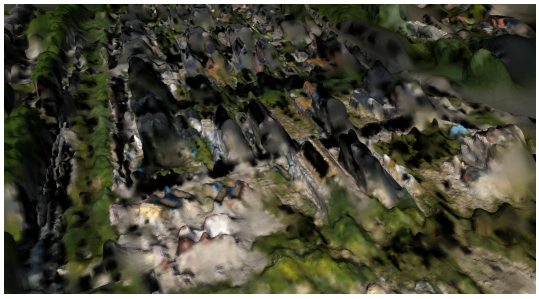


Figure 6.6.: Example of the Geodelta office mesh with Canny edge distance threshold of 0.7, still exhibiting a good mesh. Only the texture is affected in regions where the pixels are discarded.

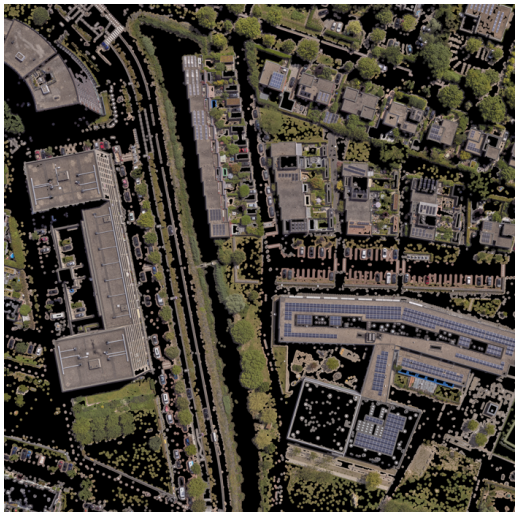
we often lose entire sections of roofs or even full buildings. Our importance region detection is simply not fine-grained enough to handle full-scene inputs, especially not at this high of a threshold. The idea of removing unimportant parts of the image for reconstruction still holds, but the way of steering the importance base maps for full-scene views is something that should be researched further.



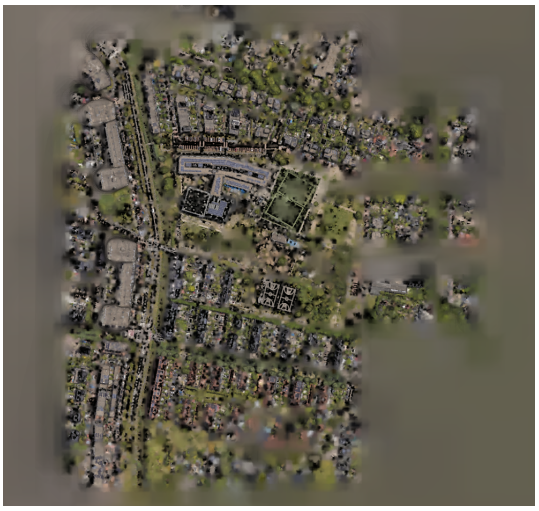
(a) Mesh from oblique dataset with 0.7 thresholding

(b) Mesh from oblique dataset without thresholding

Figure 6.7.: Exaple of our approach on oblique dataset.



(a) Example thresholded nadir input image.



(b) Mesh from nadir dataset with Canny edge distance threshold 0.7

Figure 6.8.: Example of our approach on nadir dataset.

6.6. Conclusion

This section will conclude our research and answer our main research question:

How can SAM-based building segmentation and importance estimation in oblique aerial imagery be used to improve the efficiency of 3D mesh reconstruction by selectively focusing on important image regions for reconstruction quality?

At the start of this research, we planned to enhance sparse reconstruction by adding geometrically important points derived from point cloud analysis. However, practical challenges, such as requiring a dense point cloud beforehand and the risk of reinforcing errors, led us to shift the methodology toward an image-based approach. This new strategy leverages SAM and other importance estimation methods to selectively mask image regions, streamlining dense reconstruction without relying on sparse point cloud augmentation.

SAM shows potential in identifying important image regions for mesh reconstruction and segmenting buildings from oblique aerial imagery. Our results indicate that it can reduce output size while maintaining mesh quality, particularly in terms of the ratio of sparse points to input size, where it outperformed the Canny edge-based method.

However, SAM-based approaches fall short in optimizing the full runtime pipeline. Runtime improvements were minimal, especially once the additional segmentation time was considered. Gains in dense matching speed were only noticeable when images were aggressively thresholded, at the cost of reconstruction quality. As a preprocessing step for building-level reconstruction, the method also failed to preserve sufficient image context for successful SfM matching, highlighting the importance of full-scene information in oblique imagery.

This research contributes a pipeline that integrates BAG data, oblique image orientation, and SAM to generate building masks across an entire AOI. It also introduces a method to evaluate image region importance by comparing resulting meshes to ground truth.

Nonetheless, several limitations remain. The approach struggles in scenes with heavy occlusion, such as dense urban areas. Importance estimation methods like SAM are also relatively slow, and faster alternatives (e.g., Canny edge detection) may offer more practical trade-offs. Additionally, our evaluation was based on a single dataset (Geodelta drone office), limiting the generalizability of the configurations (importance-based methods, thresholds).

Future work should focus on:

- Improve importance estimation techniques to enhance mesh quality while keeping the process fast (unlike SAM, which can take up to 6 hours).
- Enhancing evaluation metrics to focus only on relevant reconstruction regions.
- Adapting the pipeline for more complex or large-scale datasets, including nadir and full oblique scenes.
- Try performing sparse reconstruction on the full images, followed by dense reconstruction using the selectively masked image regions.

A. SamAutomaticMaskGenerator parameters

Parameter	Description
`model`	The SAM model instance used for mask generation. Required.
`points_per_side`	Number of sampling points per image side. Total = N^2. If `None`, `point_grids` must be provided. Default: 32.
`points_per_batch`	Number of point prompts evaluated in parallel. Affects speed and memory. Default: 64.
`pred_iou_thresh`	Filters masks with low predicted IoU scores (quality threshold). Range: [0,1]. Default: 0.88.
`stability_score_thresh`	Filters unstable masks (those sensitive to binarization). Range: [0,1]. Default: 0.95.
`stability_score_offset`	Offset used when computing the stability score. Default: 1.0.
`box_nms_thresh`	IoU threshold for suppressing duplicate masks (Non-Max Suppression). Default: 0.7.
`crop_n_layers`	Enables recursive cropping. Sets number of refinement layers. Default: 0 (no cropping).
`crop_nms_thresh`	NMS threshold for duplicate filtering across crops. Default: 0.7.
`crop_overlap_ratio`	Amount of overlap between crop regions. Default: $512/1500 \approx 0.341$.
`crop_n_points_downscale_factor`	Reduces point density in each crop layer: $N_i = N/\text{factor}^i$. Default: 1.
`point_grids`	Optional list of custom point grids (normalized [0,1]). Exclusive with `points_per_side`. Default: None.
`min_mask_region_area`	Filters small or disconnected mask regions. Requires OpenCV. Default: 0 (no filtering).
`output_mode`	Format of returned masks: `"binary_mask"`, `"uncompressed_rle"`, or `"coco_rle"`. Default: `"binary_mask"`.

B. Point cloud indices computation

We briefly describe these features here, following the methodology of Shi et al. [2022].

B.1. Curvature Entropy Feature

Curvature alone does not always reflect the geometric importance of a region. To improve upon this, we followed the entropy-based curvature metric proposed in Xing and Hui [2013], building on the PCA-based curvature estimation from [Hoppe et al., 1992].

A local covariance matrix is constructed for a point's neighborhood:

$$Q = \frac{1}{m} \begin{bmatrix} \mathbf{P}_j^1 - \mathbf{O}_j \\ \vdots \\ \mathbf{P}_j^m - \mathbf{O}_j \end{bmatrix} \begin{bmatrix} \mathbf{P}_j^1 - \mathbf{O}_j \\ \vdots \\ \mathbf{P}_j^m - \mathbf{O}_j \end{bmatrix}^T \tag{B.1}$$

The computed curvature values for the neighborhood $K = \{K_{i_0}, K_{i_1}, \ldots, K_{i_n}\}$ are then used to calculate entropy:

$$H_i = -\sum_{j=0}^{n} p_j \log p_j, \quad \text{where } p_j = \frac{K_j}{\sum_{i=0}^{n} K_i} \tag{B.2}$$

This value is then normalized as:

$$C_i = \frac{H_i}{H_{i,\max}} \tag{B.3}$$

B.2. Edge Feature

The edge index quantifies how much a point deviates from the centroid of its local neighborhood:

$$\text{Edge}_j = \left| R_j - \frac{1}{m} \sum_{i=1}^{m} R_i \right| \tag{B.4}$$

This identifies points located near edges or acting as outliers in the point cloud distribution.

B.3. Density Feature

The density index measures the average distance from a point to its neighboring points:

$$\text{Density}_j = \frac{1}{m} \sum_{i=1}^{m} \left| R_j - R_i \right| \tag{B.5}$$

A kd-tree is used to efficiently find neighboring points, as detailed in Section **??**.

C. Structure from motion results

C.1. 1_detached_house (SAM) - 28 input images

Table C.1.: SfM results for 1_detached_house with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.086	0.005	0.287	12	418.333	1,193	4.20788	0.656676
0.2	0.088	0.004	0.113	7	187.143	384	3.41146	0.536607
0.3	0.089	0.005	0.265	6	179.833	340	3.17353	0.531796
0.4	0.084	0.004	0.030	4	122.25	194	2.52062	0.491187
0.5	0.086	0.003	0.003	2	60	60	2.0000	0.279012
0.6	0.080	0.002	0.009	4	112.5	168	2.67857	0.4359
0.7	0.084	0.001	0.004	2	29	29	2.0000	1.11085
0.8	0.082	0.002	0.000	0	0	0	0	0
0.9	0.081	0.001	0.000	0	0	0	0	0

Table C.2.: SfM results for 1_detached_house with SAM edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.096	0.005	0.525	14	470.357	1,401	4.70021	0.679753
0.2	0.088	0.006	0.361	4	448.75	670	2.6791	0.473758
0.3	0.086	0.006	0.511	11	164.818	332	5.46084	0.71258
0.4	0.088	0.005	0.157	7	1.14286	4	2.0000	0.000017
0.5	0.085	0.005	0.202	9	365.444	965	3.40829	0.664438
0.6	0.085	0.005	0.030	4	67.75	99	2.73737	0.548408
0.7	0.088	0.005	0.024	4	120	123	2.93848	0.738290
0.8	0.093	0.006	0.025	4	170	222	3.06306	0.802794
0.9	0.083	0.004	0.008	4	140.5	205	2.74146	0.839855

Table C.3.: SfM results for 1_detached_house with edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.088	0.006	0.311	17	490.123	1,359	4.71421	0.653123
0.1	0.087	0.005	0.486	11	331.545	694	5.25504	0.674564
0.2	0.099	0.006	0.717	12	549	1,487	4.4304	0.792753
0.3	0.092	0.005	0.351	5	368	626	2.9393	0.468019
0.4	0.086	0.006	0.470	17	468.412	1,895	4.20211	0.705464
0.5	0.090	0.006	0.518	11	322.273	697	5.08608	0.693039
0.6	0.089	0.006	0.505	4	389.25	576	2.70313	0.481089
0.7	0.086	0.007	0.375	12	253.083	818	3.71271	0.564703
0.8	0.089	0.008	0.084	5	316.4	440	3.59545	0.617071
0.9	0.086	0.006	0.019	3	292.667	386	2.27461	0.501573

C.2. 1_detached_house (manual segmentation) - 45 input images

Table C.4.: SfM results for 1_detached_house (manual) with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.110	0.037	3.673	35	1576.06	7,253	7.6054	0.911192
0.1	0.055	0.012	1.801	22	681.682	2,837	5.28622	1.02639
0.2	0.052	0.010	1.565	19	498.368	2,042	4.63712	1.10220
0.3	0.052	0.013	0.139	4	1.5	3	2	0.000028
0.4	0.050	0.016	0.089	4	279.75	490	2.28367	0.460264
0.5	0.046	0.009	0.019	3	0	0	0	0
0.6	0.044	0.008	0.006	2	176	176	2	0.600909
0.7	0.042	0.005	0.004	2	52	52	2	0.890843
0.8	0.036	0.003	0.001	2	25	25	2	0.575599
0.9	0.039	0.000	0.000	—	—	—	—	—

C.3. 2_tall_building (SAM) - 29 input images

Table C.5.: SfM results for 2_tall_building with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.309	0.064	2.402	15	1,921	4,241	3.12012	0.829121
0.1	0.289	0.061	1.989	5	2,282	3,785	3.01453	0.735958
0.2	0.240	0.062	1.390	6	1,196.17	2,591	2.76997	0.643824
0.3	0.226	0.062	0.466	3	1,002	1,426	2.10799	0.616121
0.4	0.253	0.052	0.091	3	11.33	17	2	0.111202
0.5	0.193	0.038	0.110	3	216.333	298	2.17785	0.783954
0.6	0.171	0.025	0.084	3	19	28	2.03571	0.406427
0.7	0.159	0.013	0.021	2	64	64	2	0.67827
0.8	0.157	0.006	0.005	2	57	57	2	0.62476
0.9	0.157	0.003	0.003	2	24	24	2	1.87786

Table C.6.: SfM results for 2_tall_building with SAM edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.309	0.064	2.402	15	1,921	4,241	3.12012	0.829121
0.1	0.244	0.055	2.075	19	567.842	2,942	3.66723	0.768535
0.2	0.256	0.059	0.882	3	639.667	868	2.21083	0.695119
0.3	0.218	0.064	0.681	5	1,495	2,705	2.7634	0.750072
0.4	0.219	0.070	0.591	6	967.833	2,108	2.75474	0.756143
0.5	0.203	0.072	0.402	7	692.286	1,820	2.66264	0.767264
0.6	0.188	0.070	0.390	7	661.571	1,703	2.71932	0.795041
0.7	0.221	0.076	0.217	3	235.333	337	2.09496	0.89953
0.8	0.223	0.075	0.207	2	155	155	2	1.21997
0.9	0.169	0.071	0.177	2	206	206	2	1.06122

Table C.7.: SfM results for 2_tall_building with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.309	0.064	2.402	15	1,921	4,241	3.12012	0.829121
0.1	0.304	0.106	2.886	10	1,716.4	5,219	3.28875	0.782024
0.2	0.291	0.104	3.444	19	1,721.21	9,782	3.34318	0.702166
0.3	0.264	0.102	6.101	14	1,663.21	7,760	3.00064	0.90227
0.4	0.279	0.098	1.967	13	1,249.69	5,316	3.05606	0.842655
0.5	0.305	0.100	3.205	10	69	267	2.58427	1.10686
0.6	0.296	0.102	2.430	11	200.636	791	2.79014	0.923647
0.7	0.291	0.095	3.539	12	550	2,222	2.9703	0.85611
0.8	0.290	0.088	2.332	6	4.83333	14	2.07143	0.000013
0.9	0.295	0.087	1.402	2	4	4	2	0.954711

C.4. 3_apartment_block (SAM) - 39 input images

Table C.8.: SfM results for 3_apartment_block with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.990	0.202	9.274	10	642.7	2,817	2.28151	0.922359
0.1	1.266	0.205	4.720	11	622.091	3,296	2.07615	0.938021
0.2	1.055	0.192	4.964	10	1,213.8	4,639	2.61651	1.152
0.3	0.860	0.180	4.136	15	713	3,422	3.12537	0.681197
0.4	0.833	0.149	3.441	14	997.429	3,900	3.58051	0.855569
0.5	0.670	0.118	2.312	13	1,150.92	3,765	3.97397	0.791224
0.6	0.552	0.066	1.730	11	933.909	2,761	3.72075	0.729669
0.7	0.446	0.025	0.336	9	585.778	1,416	3.72316	0.72576
0.8	0.324	0.009	0.126	6	292.5	602	2.91528	0.74585
0.9	0.326	0.007	0.019	2	61	61	2	0.409424

Table C.9.: SfM results for 3_apartment_block with SAM edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.990	0.202	9.274	10	642.7	2,817	2.28151	0.922359
0.1	1.530	0.195	12.240	3	1,733.33	1,916	2.71399	0.749354
0.2	2.563	0.196	12.565	11	736.545	3,428	2.36348	1.14071
0.3	1.336	0.236	2.761	4	2,601.25	4,180	2.48923	0.575091
0.4	3.084	0.192	5.222	4	1,293.25	2,453	2.10885	0.439248
0.5	1.779	0.175	7.205	10	1,100	3,482	3.1591	1.07109
0.6	1.745	0.166	2.783	16	505.313	2,509	3.2224	0.953957
0.7	0.792	0.164	1.358	10	752.5	3,128	2.40569	0.624312
0.8	0.638	0.142	1.046	5	348.6	713	2.71844	0.787391
0.9	0.526	0.110	1.225	2	68	68	2	1.86575

Table C.10.: SfM results for 3_apartment_block with edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0.990	0.202	9.274	10	642.7	2,817	2.28151	0.922359
0.1	1.158	0.213	8.317	12	1793.6	4,141	2.72894	0.861123
0.2	1.326	0.219	7.843	13	1957.2	5,067	2.87561	0.884502
0.3	0.822	0.201	7.323	13	2229.08	10,082	2.87423	0.866354
0.4	0.798	0.188	6.761	11	1724.3	7,981	2.81627	0.844139
0.5	0.746	0.179	5.598	10	1581.4	6,941	2.79283	0.839871
0.6	0.730	0.211	3.171	10	1150.0	5,000	2.75000	0.820000
0.7	0.688	0.183	2.269	9	897.1	3,826	2.64287	0.794208
0.8	0.657	0.148	1.044	6	468.4	1,839	2.53742	0.718529
0.9	0.620	0.120	0.524	2	70.0	70	2.00000	0.680000

C.5. 3_apartment_block (manual segmentation) - 70 input images

Table C.11.: SfM results for 3_apartment_block (manual) with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	4.340	0.730	32.351	18	127	658	3.47416	1.25129
0.1	4.691	0.738	21.536	16	1,001.8	4,819	3.32642	0.905801
0.2	3.745	0.729	16.476	14	1,275	4,107	4.34624	0.772198
0.3	3.123	0.672	14.707	15	249.533	747	5.01071	0.971711
0.4	3.720	0.675	20.814	19	1,929.84	12,690	2.88944	0.700181
0.5	2.044	0.600	12.718	22	59.2727	651	2.00307	0.965215
0.6	0.973	0.438	11.844	16	990.625	3,619	4.37966	0.876918
0.7	0.895	0.213	2.311	11	507.727	1,867	2.99143	0.597973
0.8	0.656	0.058	0.648	11	292.364	1,143	2.81365	0.908988
0.9	0.477	0.021	0.023	2	51	51	2	0.516273

Table C.12.: SfM results for 3_apartment_block (manual) with SAM edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	4.340	0.730	32.351	18	127	658	3.47416	1.25129
0.1	3.248	0.794	14.383	17	669.824	4,961	2.2953	1.1907
0.2	3.730	0.771	21.676	14	1,657.07	9,160	2.53264	0.660949
0.3	4.543	0.762	19.627	13	1,661.23	7,972	2.70898	0.77843
0.4	4.077	0.757	8.994	22	551.227	2,978	4.0722	1.15726
0.5	4.267	0.783	13.800	18	64.3333	451	2.56763	1.02932
0.6	4.617	0.767	8.500	7	63.8571	223	2.00448	1.2863
0.7	3.583	0.717	8.383	15	1005.13	5,214	2.89164	0.854771
0.8	3.245	0.612	6.173	10	742.4	3,062	2.70652	0.928545
0.9	2.893	0.465	3.720	6	354.1	1,497	2.12789	0.990324

Table C.13.: SfM results for 3_apartment_block (manual) with edge-detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	4.340	0.730	32.351	18	127	658	3.47416	1.25129
0.1	5.357	0.746	12.567	19	265.368	1,347	3.74313	1.2072
0.2	3.175	0.750	17.359	19	914	3,737	4.64704	1.16424
0.3	4.206	0.751	19.708	15	560.733	1,622	5.18557	1.04486
0.4	3.216	0.731	8.444	16	1,579.81	10,780	2.34481	0.785153
0.5	4.157	0.737	25.021	13	496.846	2,832	2.28072	0.892665
0.6	3.169	0.710	15.976	14	1,830	11,514	2.22512	0.635354
0.7	3.445	0.691	13.190	14	1600.5	9,025	2.48277	0.725661
0.8	3.691	0.655	10.101	10	1,055.2	4,071	2.59199	0.782113
0.9	3.570	0.560	10.981	14	469.5	2,925	2.24718	1.17148

C.6. 4_bouwpub (manual segmentation) - 53 input images

Table C.14.: SfM results for 4_bouwpub (manual) with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m20s	0m26s	1m36s	53	2,992.75	40,023	3.96312	1.09889
0.2	0m17s	0m26s	1m32s	53	2,811.66	38,224	3.89855	1.10369
0.3	0m17s	0m24s	1m27s	53	2,464.81	34,500	3.78652	1.09756
0.4	0m16s	0m21s	1m19s	53	2,124.09	30,801	3.65498	1.08972
0.5	0m15s	0m19s	1m6s	53	1,757.58	27,019	3.44765	1.08829
0.6	0m14s	0m14s	1m5s	50	1,421.2	21,573	3.29393	1.10230
0.7	0m13s	0m9s	0m36s	46	1,060.2	15,914	3.06453	1.09169
0.8	0m11s	0m6s	1m27s	28	898.464	8,773	2.86755	1.07772
0.9	0m10s	0m3s	0m16s	22	480.545	4,330	2.44157	1.07777

Table C.15.: SfM results for 4_bouwpub (manual) with SAM edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m18s	0m25s	1m51s	53	2,965.55	39,463	3.98282	1.10395
0.2	0m18s	0m24s	1m53s	53	2,745.38	37,116	3.92092	1.10973
0.3	0m17s	0m23s	1m52s	53	2,467.55	34,088	3.83654	1.11244
0.4	0m17s	0m22s	1m51s	53	2,180.53	31,059	3.72092	1.10269
0.5	0m15s	0m20s	1m25s	53	1,865.47	27,563	3.58706	1.11416
0.6	0m15s	0m16s	1m12s	52	1,505.38	22,637	3.45806	1.08461
0.7	0m14s	0m11s	1m41s	22	1,479.91	9,266	3.51371	1.09432
0.8	0m12s	0m6s	0m36s	32	917.344	9,415	3.11790	1.11087
0.9	0m9s	0m4s	0m24s	18	288.111	2,124	2.44162	1.14011

Table C.16.: SfM results for 4_bouwpub (manual) with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	0m20s	0m26s	1m32s	53	3,068.72	40,725	3.99366	1.10242
0.1	0m20s	0m22s	1m16s	53	2,223.13	31,398	3.75266	1.08451
0.2	0m17s	0m18s	1m11s	53	1,838.26	26,503	3.67611	1.08779
0.3	0m16s	0m13s	0m55s	51	1,456.43	20,563	3.61222	1.03688
0.4	0m13s	0m8s	0m41s	43	1,081.12	12,885	3.60792	0.974324
0.5	0m9s	0m4s	1m1s	33	704.121	6,653	3.49256	0.858625
0.6	0m8s	0m2s	0m16s	22	414.591	2,570	3.54903	0.758927
0.7	0m7s	0m1s	0m1s	2	67.000	67	2.0000	0.294403
0.8	0m6s	0m1s	0m1s	3	2.000	2	3.0000	0.000600
0.9	0m6s	0m1s	0m1s	2	39.000	39.000	2.0000	0.205755

C.7. 5_geodelta_drone (manual segmentation) - 200 input images

These are the results of extraction, matching and reconstruction for a dataset of 200 drone photos of the monumental Geodelta office.

Table C.17.: SfM results for 5_Geodelta_drone with edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	4m39s	6m38s	12m25s	200	5,507.8	216,269	5.09347	0.982634
0.2	6m55s	6m43s	13m33s	200	5,500.65	217,849	5.04997	0.988194
0.3	6m55s	6m43s	11m56s	200	5,352.19	218,915	4.88974	1.00084
0.4	4m36s	6m41s	11m49s	200	5,263.3	223,275	4.71463	1.01127
0.5	6m54s	6m42s	11m3s	197	5,197.9	226,189	4.52713	1.02545
0.6	5m27s	6m57s	13m45s	188	5,076.54	217,971	4.37851	1.04974
0.7	3m36s	7m11s	11m43s	189	4,867.19	218,592	4.20829	1.05362
0.8	4m8s	6m42s	10m59s	185	4,473.37	202,852	4.07969	1.0636
0.9	4m11s	6m26s	13m39s	172	3,847.78	169,543	3.90355	1.0684

Table C.18.: SfM results for 5_Geodelta_drone with SAM edge detection

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	3m28s	6m41s	10m56s	201	5,216.94	215,316	4.87007	0.98635
0.2	3m28s	4m42s	12m24s	201	4,820.04	212,044	4.56900	0.99577
0.3	3m28s	6m41s	16m32s	195	4,499.99	203,484	4.31237	1.00809
0.4	4m46s	6m16s	10m55s	190	4,163.51	193,712	4.08373	1.01948
0.5	3m34s	6m17s	14m37s	176	4,048.31	187,623	3.79752	1.03436
0.6	3m27s	6m11s	12m34s	164	3,838.39	175,753	3.58171	1.05837
0.7	2m43s	5m25s	9m34s	158	3,386.99	162,049	3.30237	1.07905
0.8	2m58s	4m33s	14m0s	133	2,679.07	117,663	3.02828	1.12399
0.9	2m34s	4m21s	8m57s	40	750.975	12,856	2.33657	1.19275

Table C.19.: SfM results for 5_Geodelta_drone with SAM kernel entropy

Threshold	Extr. Time	Match. Time	Reconstr. Time	# Images	Mean observations/image	# Points	Mean track length	Mean reproj. error
0.0	5m15s	6m18s	13m34s	201	5,595.12	212,625	5.28922	0.98005
0.1	3m9s	6m6s	11m47s	200	5,163.67	212,185	4.86714	0.980572
0.2	3m2s	5m47s	9m55s	200	4,841.91	205,527	4.7117	0.981102
0.3	2m45s	5m36s	9m16s	200	4,416.39	195,804	4.51103	0.980386
0.4	2m16s	5m22s	9m31s	200	3,879.08	183,980	4.21685	0.983975
0.5	2m3s	4m23s	7m10s	194	3,122.97	153,196	3.95478	0.977657
0.6	1m39s	2m29s	5m2s	171	2,133.85	96,284	3.78972	0.983343
0.7	1m19s	0m48s	3m10s	154	952.896	41,018	3.5776	0.951435
0.8	1m6s	0m13s	2m47s	101	274.010	8,383	3.30132	0.892823
0.9	1m2s	0m5s	0m3s	7	119.000	282	2.9539	0.64008

Bibliography

Adams, R. and Bischof, L. (1994). Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3d city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889.

Bloomenthal, J., Bajaj, C., Blinn, J., Cani-Gascuel, M.-P., Rockwood, A., Wyvill, B., and Wyvill, G. (1997). *Introduction to Implicit Surfaces*. Morgan Kaufmann, San Francisco, CA.

Bradski, G. (2000). OpenCV: Open Source Computer Vision Library. https://opencv.org/.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

Fraser, C. S. and Cronk, S. (2009). A review of digital camera calibration methods for close-range photogrammetry. *The Photogrammetric Record*, 24(124):274–293.

Guillard, B., Remelli, E., Lukoianov, A., Richter, S. R., Bagautdinov, T., Baque, P., and Fua, P. (2022). Deepmesh: Differentiable iso-surface extraction.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson Surface Reconstruction. In Sheffer, A. and Polthier, K., editors, *Symposium on Geometry Processing*. The Eurographics Association.

Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. (2019). Panoptic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9404–9413.

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., Girshick, R., and Radosavovic, I. (2023). Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14784–14794.

Liu, J., Gao, J., Ji, S., Zeng, C., Zhang, S., and Gong, J. (2023). Deep learning based multi-view stereo matching and 3D scene reconstruction from oblique aerial images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 204:42–60.

Liu, J. and Ji, S. (2020). A novel recurrent encoder-decoder structure for large-scale multi-view stereo reconstruction from an open aerial dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6049–6058.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169.

Nex, F. and Remondino, F. (2014). Uav for 3d mapping applications: A review. *Applied Geomatics*, 6:1–15.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66.

Remondino, F. and El-Hakim, S. (2006). Image-based 3d modelling: A review. *The Photogrammetric Record*, 21(115):269–291.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423.

Shi, Z., Xu, W., and Meng, H. (2022). A point cloud simplification algorithm based on weighted feature indexes for 3d scanning sensors. *Sensors*, 22(19).

Stucker, C., Ke, B., Yue, Y., Huang, S., Armeni, I., and Schindler, K. (2022). Implicity: City modeling from satellite images with deep implicit occupancy fields. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2–2022:193–201.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. *PeerJ*, 2:e453.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272.

Xing, L. and Hui, K.-C. (2013). Entropy-based mesh simplification. *Computer-Aided Design and Applications*, 7:911–918.

Yao, Y., Luo, Z., Li, S., Fang, T., and Quan, L. (2018). Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783.

Yu, D., Ji, S., Liu, J., and Wei, S. (2021). Automatic 3d building reconstruction from multi-view aerial images with deep learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 171:155–170.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.