# A Voxel-Based Approach for Imaging Voids in Three-Dimensional Point Clouds

by

Katie N. Salvaggio

B.S. Rochester Institute of Technology, 2010

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Chester F. Carlson Center for Imaging Science
College of Science
Rochester Institute of Technology

May 21, 2015

Signature of the Author _____

Accepted by _____
Coordinator, Ph.D. Degree Program Date

CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE

COLLEGE OF SCIENCE

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

---

Ph.D. DEGREE DISSERTATION

---

The Ph.D. Degree Dissertation of Katie N. Salvaggio
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Imaging Science

_____

Dr. Carl Salvaggio, Dissertation Advisor

_____

Dr. David W. Messinger

_____

Dr. Derek J. Walvoord

_____

Dr. Rajendra K. Raj

_____

Date

# A Voxel-Based Approach for Imaging Voids in Three-Dimensional Point Clouds

by

Katie N. Salvaggio

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Doctor of Philosophy Degree
at the Rochester Institute of Technology

## Abstract

Geographically accurate scene models have enormous potential beyond that of just simple visualizations in regard to automated scene generation. In recent years, thanks to ever increasing computational efficiencies, there has been significant growth in both the computer vision and photogrammetry communities pertaining to automatic scene reconstruction from multiple-view imagery. The result of these algorithms is a three-dimensional (3D) point cloud which can be used to derive a final model using surface reconstruction techniques. However, the fidelity of these point clouds has not been well studied, and voids often exist within the point cloud. Voids exist in texturally difficult areas, as well as areas where multiple views were not obtained during collection, constant occlusion existed due to collection angles or overlapping scene geometry, or in regions that failed to triangulate accurately. It may be possible to fill in small voids in the scene using surface reconstruction or hole-filling techniques, but this is not the case with larger more complex voids, and attempting to reconstruct them using only the knowledge of the incomplete point cloud is neither accurate nor aesthetically pleasing.

A method is presented for identifying voids in point clouds by using a voxel-based approach to partition the 3D space. By using collection geometry and information derived from the point cloud, it is possible to detect unsampled voxels such that voids can be identified. This analysis takes into account the location of the camera and the 3D points themselves to capitalize on the idea of free space, such that voxels that lie on the ray between the camera and point are devoid of obstruction, as a clear line of sight is a necessary requirement for reconstruction. Using

this approach, voxels are classified into three categories: occupied (contains points from the point cloud), free (rays from the camera to the point passed through the voxel), and unsampled (does not contain points and no rays passed through the area). Voids in the voxel space are manifested as unsampled voxels. A similar line-of-sight analysis can then be used to pinpoint locations at aircraft altitude at which the voids in the point clouds could theoretically be imaged. This work is based on the assumption that inclusion of more images of the void areas in the 3D reconstruction process will reduce the number of voids in the point cloud that were a result of lack of coverage. Voids resulting from texturally difficult areas will not benefit from more imagery in the reconstruction process, and thus are identified and removed prior to the determination of future potential imaging locations.

## Acknowledgements

An old African proverb states "It takes a village to raise a child." Well, I am here to attest that it also takes a village raise a grad student. I am lucky enough to have many people in my life who helped me get to this point, and I would like to take a moment to say thank you.

First and foremost, I must thank my thesis committee. Dr. Carl Salvaggio has been my advisor through both my undergraduate and graduate career, and I cannot thank him enough for his dedication, patience, and encouragement. Without him, I never would have made it to this point. I would like to offer my sincerest appreciation to Dr. Derek Walvoord, who started out as my contact point at Exelis and through this process has become a trusted advisor and friend. His passion and enthusiasm for the research is unparalleled and he served as a constant source of encouragement and inspiration. Thank you to Dr. David Messinger for serving on my committee and his endless support. His thought-provoking questions and helpful insights were always appreciated. And finally, thank you to Dr. Rajendra Raj, for not only introducing me to the world of object-oriented programming, but also for serving as the chair of my thesis committee.

A very special thanks to Dr. David Nilosek, who was my 3D partner in crime and braved the unknown world of C++ and computer vision with me. Despite the fact that he has graduated and moved on from his work at RIT, he still made himself available to help debug finicky 3D codes via late night chats. Thank you to Dr. Shea Hagstrom, whose voxel-based LIDAR processing for his own research inspired me to look at image-based 3D point clouds in a different way, and eventually led me down a voxel-based path of my own. Another very special thanks goes to Jason Faulring, who always made himself available to help, mostly by putting out fires as I continually broke different pieces of code that I was working on. Without his insight and technical expertise, the voxel processing would have never come to fruition. Many thanks go to Cindy Schultz, DIRS students such as myself would be lost without your help and guidance. Sue Chan has also offered support and guidance to help keep me on track, and I truly appreciate it.

I would like to thank all of the professors that I had through my course work at RIT, both within the Center for Imaging Science and outside it. Imaging Science is what drew me to RIT, combining my love of photography, mathematics, and science, but I feel truly blessed to have been a part of an institution with such a wealth of knowledge and talent. I am grateful to have had a chance to learn from all of you.

I am lucky enough to have shared office space with several people during my DIRS tenure, and for that experience I am grateful. Amanda, Dave, Josh and Monica were the first lucky

bunch, and we navigated the first year of graduate coursework together. Though most of our time was spent trying to figure out Fourier homeworks, we did manage to have some fun along the way (like that time we covered everything on Dave's desk with aluminum foil). Phil, Shaohui, and Mike were on deck next, and I could not have asked for people more willing to help think through complex research problems, or tackle me into a pit of mud. To the others not mentioned, our time was brief, but I truly enjoyed all of it.

I would also like to thank my family and friends. To my mother, thank you for always believing in me and supporting me in every way imaginable. To my father, thank you for believing in me, and offering to read everything I wrote, even though it was outside of your domain of expertise. To my sister, I am so proud of the person you have become and so grateful to have had the support of your friendship along the way. And to the rest of my family, I love you all, thank you for being there for me. A huge thank you goes to Oesa and her husband Sid, who graciously gave me an open invitation to stay with them whenever I needed to return to Rochester, after I moved to Pennsylvania to complete my research from afar. I cannot thank them enough for their hospitality, as I took full advantage of their offer and gained lifelong friends in the process.

I need to thank my dog Maggie. I was supposed to be the one rescuing her, but it was her who rescued me. Sitting with me in the office during those late nights writing, or finding a way to make me laugh through my frustrations, she truly has been an outstanding "research dog." Last but certainly not least, I would not have made it here without the unyielding support that Chris has provided me. He has always believed in me, even when I had doubts myself, and supported me anyway that he could, be it helping with code, editing, or simply putting food in front of me so that I remembered to eat. I am so fortunate to have him to rely on, and as a partner in life.

*To those who believed in me, and those who taught me along the way.*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Advancements in modern computing have expanded the domains of research into areas previously considered to be too computationally intensive. One area that has benefited from such advancements and seen rapid development in the past decade is the automated generation of three-dimensional (3D) models from imagery. The photogrammetry community has been using aerial imagery for decades to develop topographic maps using stereo techniques, and more recently the computer vision community has developed an interest in photo-tourism, using Structure from Motion (SfM) techniques.

Image-based modeling techniques are of particular interest in an array of fields due to their wide applicability and low cost. Autonomous navigation and guidance has benefited tremendously from global positioning systems (GPS), however that is not available in all conditions (e.g. indoors), making vision-based techniques an attractive alternative. Typical two-dimensional (2D) segmentation techniques, such as thresholding, pixel clustering, and region-growing, used for object segmentation and recognition could see significant improvements with the inclusion of depth information. Other potential areas for application include image mosaicking, photo organization, spatial and temporal tracking, and robotic hand-eye calibration, to name a few [1].

The objective of scene reconstruction methodologies is to automatically extract 3D structure from the imagery to obtain a complete model of a scene. SfM techniques usually result in a point cloud, containing individual 3D point locations for a model. In most cases, a point cloud will not suffice as a model as they are not directly usable in most 3D applications. The

point clouds themselves are converted to a surface model through a process referred to as surface reconstruction. The reconstruction of surfaces from points is not a straightforward problem, as point sampling and spacing is often non-uniform, positions (and normals if available) are noisy, and some regions are lacking data due to obscuration, accessibility limitations, and textural challenges. Traditional surface reconstruction techniques will often fail in complex regions where a point cloud is devoid of points, resulting either in a hole in the model or a poorly estimated surface that is neither accurate nor aesthetically pleasing.

Voids exist in a point cloud where multiple views of an area were not included in the input imagery, occlusion or heavy shadow blocked an area from clear view, texturally difficult areas which fail to generate features or result in poorly matched correspondences, or an insufficient baseline between images that resulted in poor triangulation. Developing methods to identify the voids in the point cloud as well as methods for filling in these voids are of significant interest. Fitting surfaces to the raw point cloud data or rasterizing the point cloud to identify surfaces are popular approaches in surface reconstruction techniques, but as stated previously these methods are not equipped to reconstruct heavily occluded areas that resulted in large voids in the point cloud containing complex intersecting geometries.

When reconstructing structure from imagery, the resulting point cloud can only represent part of the information. If a point is reconstructed from a particular image, it must be assumed that there was a clear line of sight between the camera and the point, thereby introducing the concept of free space. Point cloud representations are not equipped to represent the idea of free space, but this can be achieved using voxel-based representation. Representing the point cloud in a three-dimensional rasterized space will provide means to capitalize on the free space idea as a way to identify voids in the point cloud.

In areas where voids are a result of occlusion or lack of coverage, it is reasonable to assume that including images in the reconstruction where those areas were visible would result in an improved reconstruction. In addition to identifying the voids, it is of interest to identify locations from which the voids could be seen, because inclusion of such images in a reconstruction may in the voids in the initial point cloud. Additional imagery is not necessarily readily available to include in a reconstruction, particularly when specifying areas of interest. Therefore it may be useful to provide potential imaging locations for the purposes of flight planning for future image collections.

This work will focus on the development of algorithms to construct voxelized scene repre-

sentations from point cloud data, and to leverage the information available in the voxel space to identify voids and potential future image locations.

## 1.1 Research Goals

As was previously stated, reconstructions of three-dimensional (3D) point clouds from multi-view aerial imagery are readily obtainable, but the fidelity of these point clouds has not been well studied, and voids often exist within the point cloud. Voids in the point cloud are present in texturally difficult areas that failed to generate feature matches during the densification process of reconstruction, as well as areas where multiple views were not obtained during collection or a constant occlusion existed due to collection angles or overlapping scene. The goals of this work are to be able to identify voids in point clouds, identify the type of void, and subsequently identify suitable locations from which to image the void. Ideally, inclusion of these new images in the 3D reconstruction would reduce the voids in the point cloud that are a result of lack of coverage.

## 1.2 Objectives

1. Create a set of high frame rate, oblique synthetic image data with corresponding ground truth for every pixel. DIRSIG software will be used to produce synthetic imagery of a scene in addition to truth imagery with corresponding ground truth positions and normals from the model.

2. Generate 3D point clouds with high frame rate oblique imagery, both real and synthetic, using an open source Structure from Motion (SfM) workflow.

3. Develop tools and algorithms to construct voxel maps from the point cloud data. It is assumed that the point cloud data is a result of a specific workflow, and therefore has corresponding camera coordinates that were used to reconstruct each point, registered in the same coordinate system as the point cloud.

4. Identify voids in the point clouds using the voxel maps. By using collection geometry and information derived from the point cloud, it is possible to detect unsampled voxels such

that voids can be identified.

5. Identify voids that were likely the result of texturally difficult areas. When determining the best location to image the voids from to increase the fidelity of the 3D model, it is important to recognize that texturally difficult areas will likely not benefit from more coverage and therefore should have little impact in determining the best locations.

6. Generate maps that can be used to determine future aircraft imaging locations and pointing vectors. This is a six-dimensional problem that will be constrained using limitations of airborne platforms. The maps themselves can then be used to find specific locations and/or generate flight lines for future collections.

7. Evaluate the assumption that the predicted view would be beneficial to the 3D reconstruction by verifying that it encompasses void voxels.

## 1.3 Scope and Limitations

Generation of 3D point clouds from imagery requires a high level of overlap between frames, in addition to image baselines sufficient for reconstruction. The computer vision community has posed the Structure from Motion problem in the framework of photo tourism, where the input imagery is mined from internet photo collections, and the assumption is that the cameras are uncalibrated with unknown locations. The resulting models are reconstructed in an arbitrary space. For the purposes of this research, the datasets will come from airborne platforms equipped with a Global Positioning System (GPS) and Inertial Navigation System (INS) so that a post-processing step can be applied to put the model in a fixed Earth-based coordinate system. In addition it will be assumed that properties of the sensor, such as the focal length and pixel pitch, are known. Use of the focal length and pixel pitch (or the focal pixels) is required input to the SfM workflow that is used at RIT such that the recovered scene geometry is a metric reconstruction, which differs from the real-world model by a similarity transform (rotation and translation). For the purposes of this work, it will be assumed that a metric reconstruction is recovered from the SfM process and that a similarity transform can be used to place the reconstruction in a fixed Earth-based coordinate system.

Voxel maps require large amounts of memory, depending on the resolution. It may be necessary to limit the resolution of a voxel space based on memory capabilities of the systems available, despite the fact that the voxel space may be capable of supporting smaller voxel sizes. In addition, the models will be limited to cubic voxels to simplify calculations. Similarly, while voxel maps will be created for the entire scene, subsequent steps may focus on a subset of the scene, particularly for visualization purposes.

This work is based on the assumption that inclusion of more imagery in a 3D reconstruction will result in a point cloud with fewer voids. While there is nothing fundamentally wrong with this assumption, it is important to note the limitations of the 3D workflow used at RIT to automatically generate point clouds from imagery. The 3D workflow leverages several well-known computer vision algorithms, including Bundler (an algorithm to perform a bundle adjustment) and CMVS (a Cluster-based Multi-View Stereo algorithm). Bundler performs a bundle adjustment, but in some cases may reject images in the optimization if the resulting error vectors become too large. CMVS clusters images in such a way that is intended to remove redundancy, but the behavior of this algorithm is not well known, and it frequently removes large portions of the input dataset. In addition, CMVS has a random component to it, resulting in the removal of different images each time the algorithm is run, despite using identical parameters and image datasets. Removal of Bundler and CMVS from the workflow is beyond the scope of this work.

Another objective of this work is to identify the void areas and subsequently identify voids that were a result of texturally difficult areas, where a difficult area is a region in which it is difficult to obtain accurate image-to-image correspondences. This can be a result of homogeneous regions, spatially repetitive textures, or other regions not well suited for feature detection and matching. These regions will not benefit from the inclusion of more imagery in the reconstruction and as such should be identified prior to the future image location prediction stage. It should be noted that it may be possible to identify such regions during the densification process if all pixels were tested for matches such that each pixel could be scored, and the score could then be used to quantify poorly matched regions. This is not easily achieved within the PMVS framework and as such is beyond the scope of this work.

The final stage of the voxel-based workflow is to generate a map that can be used to determine future image locations based on a cost function. For the purposes of this work, the cost function will be based on the number of unsampled voxels visible from a given location. Due to the nature of the cost function, the predicted image location is not guaranteed to tie into

the current reconstruction and as such the predicted view may be vastly different from previously included views. It will be left up to the end user to tie the predicted view into the current reconstruction.

Finally, the field of 3D reconstruction is rapidly developing and there are numerous methods available to generate point clouds from multi-view imagery. Innovations have been made since the development of the 3D workflow used here such that the workflow may no longer be state-of-the-art, but modifications and upgrades are beyond the scope of this research. Instead the focus will be on developing methods and techniques that address voids in a point cloud, which could be applied to point clouds generated by other methods, provided that the necessary camera reconstruction information is available.

## 1.4 Contributions to Knowledge

This research will provide methods to convert point cloud models to voxel spaces, identify voids within the voxel space, and subsequently identify optimal viewing angles at which the voids can be imaged.

# Chapter 2

# Background

The use of 3D models extends to a variety of applications including navigation, visualization and animation. While models can be generated using computer-aided design (CAD) software, it is difficult to achieve an accurate and realistic model of a complex scene or object. Thus there is an interest in the photogrammetry and remote sensing community, as well as the computer vision community, in 3D reconstruction of scene geometry from imagery with no a priori knowledge of the world. This chapter is intended to provide the background information necessary to understand the process of generating 3D point clouds from imagery, including post-processing reconstruction techniques used to create surface models. References to external sources are included throughout that explain processes and algorithms in greater detail.

## 2.1 Structure from Motion

One method of identifying objects within a scene and subsequently extracting information about their structure is through analysis of their motion in a series of images; this process is commonly referred to as Structure from Motion (SfM). SfM has its roots in photogrammetry, but recent advancements have come from the computer vision community. As a result of this, much of the SfM chain assumes little knowledge of the sensor or its position and the resulting structure is in a relative coordinate system. In general, there are three steps in the SfM pipeline that leads to automated 3D reconstruction: (1) generation of image-to-image correspondences, (2) estimation of relative geometry via bundle adjustment, and (3) dense recon-

struction.

### 2.1.1 Image-to-Image Correspondences

Identification of image-to-image correspondences is a crucial aspect of solving for 3D geometry and has been well studied. The correspondences allow estimation of fundamental matrices, which describe the relation between image stereo pairs and provide the necessary epipolar geometry for initial triangulation. A fundamental matrix is a homogeneous 3x3 matrix, described by seven degrees of freedom, that relates image correspondences between stereo pairs. A 3x3 homogeneous matrix has eight independent ratios and therefore eight degrees of freedom, however a fundamental matrix satisfies the constraint that the determinant of the matrixis zero, thereby removing one degree of freedom. Prespective changes in an image are described by an $XYZ$ rotation, $XYZ$ translation, and a scale factor, thereby achieving the seven degrees of freedom in the fundamental matrix. A strictly correlation-based approach to feature matching is not suitable for SfM applications because it can only describe a two-dimensional translation.

Image-to Image correspondences are generated as a result of a three step process: (1) detecting possible feature locations, (2) distinguishing individual features through use of a unique feature descriptor, and (3) matching feature descriptors across images to obtain the desired correspondences. The scale-invariant feature transform (SIFT) [2], currently one of the most popular feature detectors in the community, accomplishes all three tasks and will be the basis for discussion here. It should be noted that there are many other feature detectors available, many of which are cited in the forthcoming sections. Use of other feature detectors is becoming more common as there has been an increasing trend in the community to move away from SIFT.

#### Feature Detection

The objective of detection is to identify locations such that the detector will reliably find the same points of interest under varying viewing conditions. Detection of local interest points can be traced back to corner detectors and the work of Moravec [3] and Harris [4]. These algorithms are designed to detect image locations that have large gradients in all directions, and therefore are not limited to identifying only corners, however they are not scale-invariant. Objects in the world appear differently depending on the scale of the observation. If the objective is to de-

scribe and relate these objects, the notion of scale becomes important. The idea of automated scale selection was introduced by Lindeberg [5], based on the notion of scale space introduced by Witkin [6]. Mikolajczyk and Schmid showed that extrema in the scale-normalized Laplacian of Guassian produced the most repeatable features [7].

SIFT detects features using a cascaded filtering approach. Locations that are invariant to scale change of an image are determined by searching for features across all possible scales, where the scale space of an image is given by the convolution of the image and a variable scale Gaussian kernel. Stable feature locations are found using scale space extrema in a difference-of-Gaussian function convolved with the image. The difference-of-Gaussian, easily computed by subtracting adjacent image scales, is a close approximation to the scale-normalized Laplacian of Gaussian required for true scale invariance [2]. Local minima and maxima in the difference-of-Gaussian images identify possible candidate locations. A detailed model is fit at each candidate location, and candidates are selected based on measures of stability. For a more in-depth explanation of how the SIFT algorithm chooses candidate pixels, refer to Lowe [2].

Other feature detectors include SURF (Speeded Up Robust Features) [8], FAST (Features from Accelerated Segment Test) [9], and MSER (Maximally Stable Extremal Regions) [10], though this list is far from exhaustive.

#### Feature Descriptors

Once a stable set of feature locations has been found, the locations must be characterized with a descriptor that will be invariable under a variety of viewing conditions. A wide range of descriptors have been proposed, though it has been shown that distribution-based descriptors of the region of interest in the feature's local neighborhood outperform other methods [11]. The SIFT descriptor is distribution based and its features have been shown to be invariant to image rotation and scale in addition to being robust to a range of affine transforms, the addition of noise, as well as some change in illumination [2].

SIFT achieves rotation invariance by assigning a dominant orientation to a feature such that the descriptor is developed relative to this orientation. The scale of the feature is used to select a Gaussian smoothed image in which to perform computations so as to achieve scale invariance. The gradient magnitude and orientation are computed for each pixel and an orientation histogram is formed. The orientations are weighted by their gradient magnitude and

a Gaussian-weighted circular window, and the highest peak in the histogram defines the dominant orientation of the feature. For locations with multiple dominant peaks, multiple features are created.

To generate the feature descriptor, the image gradient magnitudes and orientations are sampled around the feature location. The orientations are rotated relative to the dominant orientation computed previously. Orientation histograms summarizing subregions around the point of interest are then computed in a manner similar to the one described previously. Typical SIFT features are 128 elements in length. The complexity of the SIFT descriptor can be varied by changing the number of orientations in the histogram and the size of the sampled regions. Though larger descriptors will theoretically result in better discrimination between features, they will also be more sensitive to occlusions and perspective changes. Finally, the feature descriptor is normalized to unit length in order to account for constant changes in brightness or contrast and the influence of large gradient magnitudes is reduced by thresholding.

Mikolajczyk and Schmid provided an extension to the SIFT descriptor by changing the location grid and reducing the size with principle components in the Gradient Location and Orientation Histogram (GLOH) descriptor [11]. The DAISY descriptor replaces the weighted sums of gradient norms by recursive convolutions with oriented derivatives to reduce computational requirements such that a descriptor can be efficiently generated for every pixel in an image [12]. DAISY is intended for dense wide-baseline matching and therefore does not employ a detection stage. Finally, the SIFT descriptor is invariant to four of six parameters of an affine transform, and ASIFT or Affine-SIFT was designed to extend SIFT such that full affine invariance is achieved [13].

#### Feature Matching

Once features and descriptors have been computed, the next step is to match feature descriptors across images such that the desired image-to-image correspondences are finally obtained. There are several methods that could be employed to match features. Simple metrics such as Euclidean distance, Mahalanobis distance, and spectral angle matching can be employed such that two features are considered matches if the distance or angle between them is less than some predetermined global threshold. These methods require an exhaustive search and would likely result in a high number of false matches due to the global thresholds applied. In addition

to such brute-force techniques, model-fitting algorithms such as random sample consensus (RANSAC)[14] can be employed.

The method of matching employed by SIFT is based on a Euclidean distance, but adds another level of complexity by comparing the distance of the closest neighbor to the distance of the second closest neighbor in order to reduce the number of incorrect matches. Due to the high dimensionality of the space, for false alarms it is likely that there are several other matches within a similar distance. Using this logic, the second match can be thought of as providing an estimate of the density of false matches while also identifying feature ambiguity [2]. If the closest neighbor and second closest neighbor are the same distance from the feature of interest, the ratio of the distance of the second closest neighbor to the distance of the first closest will be close to unity. A threshold can be implemented such that only feature pairs where this ratio is below the threshold are considered matching and those above the threshold are rejected as potential matching pairs. Lowe suggests rejecting all matches in which the distance ratio is greater than 0.8 to eliminate 90% of false matches while discarding less than 5% of correct matches [2].

### 2.1.2 Bundle Adjustment

Image-to-image correspondences can then be used to define the projective geometry between two scene views, known as epipolar geometry, through use of what is known as the fundamental matrix [15]. Though the fundamental matrix is dependent on intrinsic camera parameters and relative pose, it can be computed from corresponding scene points without this knowledge. A series of fundamental matrices provides the necessary epipolar geometry for cursory triangulation, resulting in a series of equations that relate the image coordinate system to the world coordinate system. Given these initial estimates, refinement of the camera projection matrices is critical to ensuring accurate relative orientation and consistent triangulation. This refinement is accomplished using a bundle adjustment.

Bundle adjustment refers to the large non-linear least squares problem that is solved in a feature-based SfM algorithm. The term is utilized both in the photogrammetry community, where it was conceived in the 1950s [16], and the computer vision community, where it is now regarded as the gold standard for performing 3D reconstructions from correspondences [15]. The objective of a bundle adjustment is to estimate the camera projection matrices and 3D

scene points to obtain an optimal reconstruction. This is achieved by minimizing the reprojection error between the observed and predicted points, expressed as a sum of squares of non-linear real-valued functions [17]. This minimization is achieved using non-linear least squares techniques, of which the Levenberg-Marquardt (LM) algorithm has been found to be most successful [17].

The LM algorithm iteratively linearizes the function in the neighborhood of the current estimate to solve what are known as the normal equations. However, due to the nature of the bundle adjustment, this can become an extremely large minimization problem. Consider a reconstruction of $n$ 3D points over $m$ views. Given that each 3D point has three degrees of freedom, and each camera matrix has eleven degrees of freedom, this gives rise to a problem with $3n + 11m$ parameters. The size of the Jacobian and large matrix factorizations required by the LM algorithm become more complicated and costly as $m$ and $n$ grow. However there is a sparse block structure in the normal equations matrix due to the lack of interaction among parameters, and considerable computational benefits can be gained by taking advantage of this structure.

Bundler, a software package for iterative bundle adjustment written by Snavely [18], leverages `SBA`, a package for generic sparse bundle adjustment written by Lourakis [17], to obtain camera matrices and a sparse point cloud. Bundler works to find geometrically consistent matches between image pairs by computing a fundamental matrix from matching features using a RANSAC algorithm [14]. Features are organized into tracks by matching across multiple images. Camera parameters and 3D locations for each track are recovered using an incremental bundle adjustment approach to add one camera at a time. The initial pair is chosen such that it has a large number of matches and a large baseline. Large SfM problems are prone to getting stuck in bad local minima and the incremental approach helps to avoid this problem [18]. Relative position of the cameras is estimated from the images and requires no external information. Cameras are added one at a time, adding ones that have observed the most previously estimated tracks first. Extrinsic parameters are estimated using a direct linear transform and 3D points are estimated with triangulation [15]. `SBA` minimizes the objective function at every iteration, and the process is repeated until there are no cameras remaining.

Snavely's work successfully demonstrated the application of SfM techniques on real world photo collections from internet sources [19]. These photo collections offer challenges as they are taken from many different cameras with different resolution, levels of zoom, illumination,

time of day, weather conditions, etc. This differs from the parameters of an aerial reconstruction in which images are taken in sequence from the same camera source, often with position and orientation information readily available. Admittedly, it may be possible to leverage the additional information available with an aerial dataset, however, Bundler is widely used in the community and was implemented for convenience.

### 2.1.3 Dense Stereo Matching

The bundle adjustment process results in an estimation of the camera matrices in addition to a sparse point cloud. While the sparse point cloud does contain scene structure, a denser version is often desired for surface reconstruction to obtain a realistic model. Standard multi-view stereo (MVS) algorithms can be applied to achieve a dense reconstruction of the desired object or scene.

These methods generally require two different inputs, the images themselves and the camera pose, estimated precisely by a SfM algorithm such as the bundle adjustment described previously. MVS algorithms can be organized into four categories based on their underlying object models: voxel-based models, deformable polygonal mesh models, depth map models, and patch-based models. The following discussion will focus on the patch-based models [20, 21], but it should be noted that probabilistic voxel modeling [22] and semi-global image matching [23] have also shown promise and should be explored in future endeavors.

#### Cluster-based Multi-View Stereo (CMVS)

As the number of images to be used in a 3D reconstruction grows, it is no longer feasible to use all available images simultaneously to construct a model and it becomes necessary to use a cluster-based method to achieve scalability in dense reconstruction. View selection can be used to decompose a set of images into clusters, a MVS algorithm can then be used to reconstruct dense 3D points and the resulting solutions can be merged into a single model. The Cluster-based Multi-View Stereo (CMVS) algorithm [20], written by Furukawa, accomplishes this task.

There are three constraints that need to be satisfied to solve the overlapping view clustering problem: (1) redundant images must be excluded to ensure compactness, (2) a size constraint must be enforced to ensure that clusters are small enough for reconstruction, and (3)

reconstructions from the clusters must result in a minimum loss of detail in comparison to that which can be obtained using the full image set to ensure coverage [20]. It is important that there is some amount of overlap in the clusters because a strict partition of the imagery would result in undersampling surfaces near cluster boundaries. Redundant images in the set are excluded to reduce the noise that will result from an insufficient baseline between images, as well as to improve computational efficiency. The algorithm implicitly incorporates image quality because poor image quality will result in fewer SfM points and therefore low quality images are more costly to include given the imposed coverage constraint.

The algorithm begins by minimizing the number of SfM points by merging points with neighbors. Each image is then tested to see if the coverage constraint is satisfied without the image; images removed in this step are removed permanently to speed processing in other steps. Image clusters are then divided if they do not satisfy the size constraint and then images are added back to the clusters if the coverage constraint is not satisfied; this process is repeated until the size constraint is satisfied.

#### Patch-based Multi-View Stereo (PMVS)

Once the images have been divided into clusters using CMVS, the Patch-based Multi-View Stereo (PMVS) algorithm [21] is used on each cluster to generate a dense point cloud. Patch-based methods suffice for a point-based rendering such as a point cloud, but will require post processing to turn the point clouds into a mesh model if desired.

Images are associated with a regular grid of pixel cells and the objective of the algorithm is to reconstruct at least one patch in every image cell. In this case, a patch is a local tangent-plane approximation to a surface that is fully described by its center, unit normal vector, and the reference image in which the patch is visible [21]. The reconstruction process is achieved by initial feature matching, followed by a repetitive expansion and filtering operation designed to increase the density of the model and remove erroneous patches.

Initial features are detected in the images using both difference of Gaussians and a Harris corner detector. For each detected feature, a set of features is collected within other images that lie within two pixels of the corresponding epipolar lines and a triangulation is performed to obtain the associated 3D points. These features are considered potential patch centers. It is assumed a patch is visible in an image if the angle between the camera axis and the patch

normal is less than a specified threshold angle. If a specified number of images exist with low photometric discrepancy, then the patch generation is deemed a success. Once a patch has been reconstructed and stored, all the features in the cell are removed and not used again.

Once the initial patches are created, then an expansion takes place where the objective is to reconstruct at least one patch in every image cell. A set of neighboring image cells are identified for a given patch and new patch candidates are generated. The new patches are filtered and a visibility constraint is enforced to eliminate erroneous patches and incorrect matches. The color and normals of matching patches are compared and discrepancies are indicative of incorrect matches. The expansion and filtering processes are iterated three times to make patches dense and remove outliers. Certain parameters and thresholds are loosened after each iteration. Once the expansion and filtering is complete, the result is a dense 3D point cloud. It is important to note that the particular implementations of the bundle adjustment and multiple-view stereo algorithms used require no input beyond initial imagery, and therefore the scene is reconstructed up to a projective ambiguity with respect to the world coordinate system.

### 2.1.4 Geoaccurate Transformation

The models generated through a typical SfM process cannot be used directly to extract precise geographic measurements without additional information. Many SfM applications use imagery available on the internet that has been tagged with geographic location data. With imagery captured from an airborne platform equipped with a global positioning system (GPS) and inertial navigation system (INS), position and orientation information is available within the image metadata. The position and orientation information from GPS/INS systems, though not without noise, is often much more reliable than that associated with most internet photo collections.

While it may seem that triangulation from existing metadata and the physical sensor model will yield the desired results, this approach will likely fail due to large triangulation errors as a result of uncertainties in the data, even with an initial bundle adjustment [24]. Similarly, using existing metadata to initialize a bundle adjustment such that the resulting solution is in the desired coordinate system will likely result in a poor model due to the mixing of image and sensor based geometries [24]. Instead, geolocation can be performed as a post-processing step, such that the previously discussed algorithms can still be leveraged to obtain dense point

clouds.

In many practical SfM processes, including Bundler [25], intrinsic camera parameters (i.e. focal length, pixel pitch, and sensor size) are used such that the resulting model is a metric reconstruction, where metric properties such as the angles between lines and ratios of lengths are preserved [15]. A metric reconstruction and an absolute reconstruction in the world-coordinate system differ by a rotation, translation, and uniform scale factor and thus can be described by a similarity transform with seven degrees of freedom [26].

In general, there are two assumptions made to obtain a geoaccurate model: (1) both the SfM reconstruction and the real world can be considered metric reconstructions, and (2) the difference between the two can be modeled by a similarity transform. It should be noted that a fully metric reconstruction is not always obtained from the SfM approach [18]. In such cases, the arbitrary coordinate system of the reconstruction and the desired fixed earth-based coordinate system will differ by more than a similarity transform, and therefore an affine or projective transformation may be necessary to put the SfM reconstruction in the desired space.

The similarity transform can be calculated using a set of known corresponding points between the two coordinate systems by first computing the scale, then the rotation, and finally the translation. This method is sensitive to noise in the correspondences and could benefit from a model fitting method that is robust to outliers such as RANSAC [14].

Traditional photogrammetry approaches would call for ground control points to compute the similarity transform, but in the absence of ground control points other methods to estimate the mapping must be employed. The simplest registration method uses the estimated camera centers from the SfM process as the set of points from the metric coordinate system, and the corresponding GPS located camera centers as the points from the world coordinate system. This is the method most commonly used in the computer vision community, but it is sensitive to noise in the GPS locations. Additional information can be used to further refine the transform, such as the digital surface models used by Wendel *et al* [27] or the Google Street View imagery and Google Earth models employed by Wang *et al* [28]. The high-fidelity estimates of the position and orientation provided by GPS/INS systems on airborne platforms as well as complete knowledge of the ground-to-image function allows for a simpler approach to georegistration, presented by Walvoord *et al* [24].

For the purposes of this work, the GPS camera center approach will be used. In future endeavors, this could easily be substituted for a more complex method.

### 2.1.5 Accuracy and Completeness

Automated scene reconstruction from imagery is an inherently under-determined problem. While the state of the art in 3D reconstruction is rapidly improving, there is a noticeable lack of quality metrics available for quantitative comparisons of point clouds. The Middlebury benchmark dataset and corresponding qualitative evaluation was the first of its kind for multi-view stereo algorithms [29]. Calibrated image datasets were collected in addition to 3D ground truth models. Various reconstruction methods were compared to the ground truth models and evaluated for accuracy, how close the reconstruction was to the truth, and completeness, how much of the truth was modeled by the reconstruction. However, the study was limited in scope and explicitly stated it did not consider SfM methods [29].

Often point density or completeness metrics are used as a method of comparison, or even simply the number of points in a model. However, there are no industry standards, and various definitions for such metrics exist due to the fact that the definitions of accuracy and completeness are application dependent. As such, there is still a heavy reliance on visual inspection of the point clouds. In addition, while a variety of techniques to perform georegistration exist, researchers have begun to question the accuracy of these methods.

Many have taken to using some type of ground truth as a means for accuracy assessment of the georegistered model. Neitzel *et al.* [30] evaluated the point density and completeness of resulting SfM models, in addition to evaluating the accuracy of georegistration using error residuals between survey points and the corresponding points in the SfM georegistered model. Crandall *et al.* [31] used geotags from consumer GPS (e.g. iPhone 3G) to seed their bundle adjustment, and compared the results with highly accurate camera pose information collected as ground truth. Hudzietz *et al.* [32] compared known path distances to the corresponding measured path distances in the generated models. Generating ground truth using 3D range scanners is another popular method for accuracy assessment. The Middlebury dataset [29] used such a method to generate the ground truth models. Koutsoudis *et al.* [33] also took this approach, using a single structure as a test case to evaluate readily available commercial SfM-based software packages.

While using ground truth is a valid approach, sources of error within the SfM chain itself cannot be isolated. Nilosek *et al.* [34] presented a technique to assess the accuracy of SfM models using a synthetic dataset such that sources of error could be separated. It was con-

cluded that there are two major sources of error in the geo-accurate SfM process: the georegistration process itself (noise associated with camera pose information) and the SfM process (inaccurately matched features resulting in error in triangulation).

With regards to completeness, the quality of the dataset available largely determines the extent to which 3D information will be able to be extracted. Multiple views of each part of a scene must be present in the dataset in order to have a chance at reconstruction. This type of overlap has been well studied in the photogrammetry community, particularly with aerial photography and the construction of digital elevation maps. With aerial photography, overlap can be built into flight lines and/or the data acquisition rate, though this rate is aircraft and sensor dependent. Typical near-nadir aerial photography is flown with 60 percent forward lap and 30 percent side lap. While intuitively it may seem that reconstructions would increasingly improve with more overlap and more photographs, this is not the case. The base-to-height ratio of the aerial collection has a significant impact on the elevation accuracy that can be achieved with a model.

### 2.1.6 Datasets

The computer vision community has more of a focus on photo-tourism and as such has employed methods for mining internet photo collections, such as those from Google and Flicker [18]. One of the goals of this area of study is to allow for virtual tourism of the world's most interesting landmarks; an example of this work is the "Building Rome in a Day" project [35]. Small scale datasets are also available (e.g. the Middlebury multi-view datasets [29]).

While internet photo mining and ground-based systems present interesting problems, they are not the only type of imagery used in 3D reconstruction. Researchers have also been taking advantage of aerial photography for the purposes of constructing digital elevation maps and large scale 3D models. RIT released an aerial dataset to the community that was flown with 90 percent overlap for the purposes of evaluating image collection requirements for 3D reconstruction [36].

For the purposes of this research, the focus will be on high frame-rate airborne imagery with significant overlap. Detailed information about the specific datasets used is located in Appendix A.

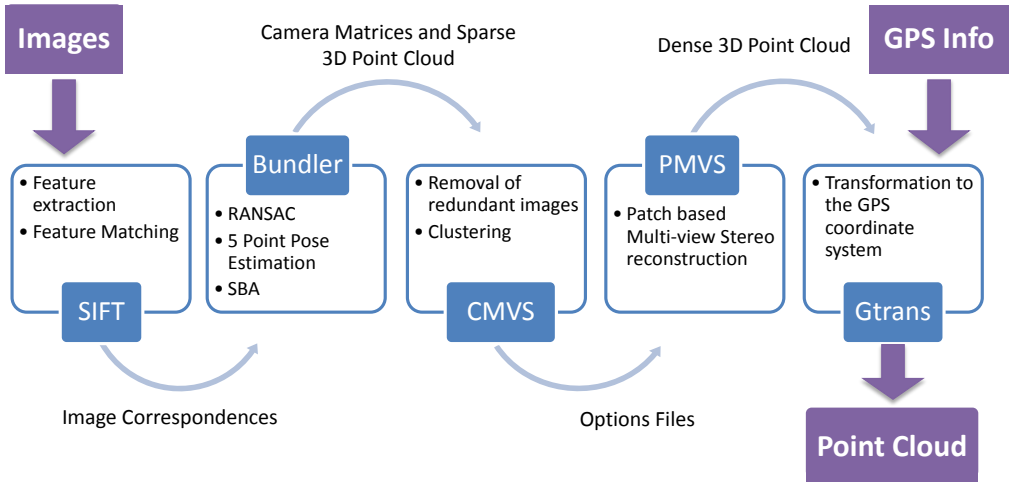


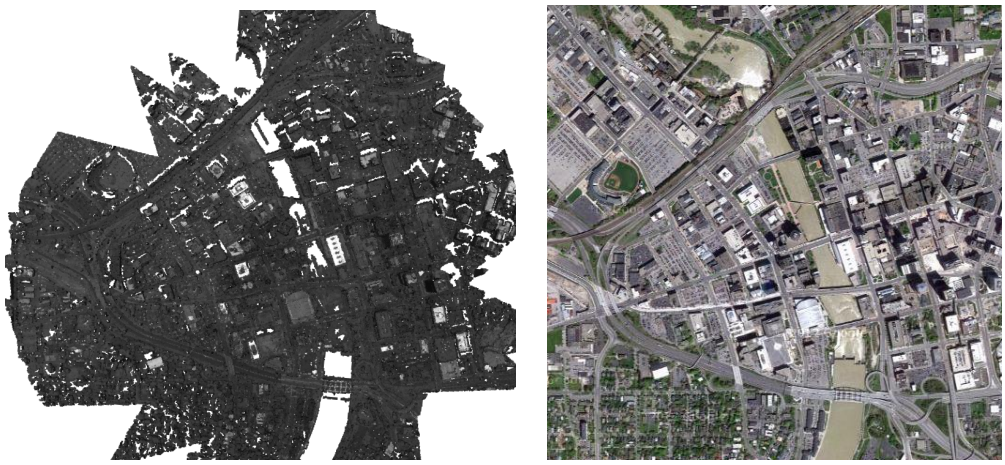

*Figure 2.1: A block diagram of the 3D workflow used at RIT to develop dense point clouds from multiple-view imagery. This workflow is a combination of several well known, open-source software packages from the computer vision community, with an additional post-processing step to place the point cloud in a geoaccurate coordinate system.*

### 2.1.7 Software

At this point, it seems appropriate to discuss specific software packages available for 3D reconstruction. Due to the heightened level of interest in the field, a collection of open-source software has emerged from the community that can be put together to form a workflow to extract 3D structure from multi-view imagery.

The workflow used to generate dense 3D point clouds at RIT is a combination of several well known software packages. Image correspondence information is generated using a GPU-accelerated version of SIFT, written by Changchang Wu [37]. Sparse bundle adjustment is then performed by Bundler, written by Noah Snavely [25]. The optimized camera information is passed through CMVS, to remove redundant images and perform clustering, and then PMVS to generate a dense 3D point cloud; both CMVS [20] and PMVS [21] are written by Yasutaka Furukawa. Finally, the GPS/INS information is used to calculate a similarity transform that places the point cloud in a fixed earth-based coordinate system; this code utilizes the simple camera center approach discussed in Section 2.1.4 and was written in house by David Nilosek [38]. The workflow is shown as an end-to-end process in Figure 2.1.

Though typically used in the computer vision community on close range photo collections,



(a) Reconstruction (b) Rochester, NY [39]

*Figure 2.2: Full scene reconstruction of downtown Rochester, NY using 48 WAMI images, and an aerial image from Google maps for visual comparison.*

this workflow was used initially at RIT to test its modeling capabilities using nadir-looking imagery from airborne platforms. From there, the scope was expanded to verify the workflow performance on high frame-rate oblique imagery from airborne platforms. Images from the Exelis Wide Area Motion Imagery (WAMI) system were used to generate a point cloud, shown here to demonstrate the SfM capabilities of the workflow. A detailed description of the WAMI sensor and imagery is available in Appendix A. The imagery used to generate this point cloud covered approximately 4 minutes of flight time over 477 images, completing just over one full rotation in a circular pattern with a fixed stare point around downtown Rochester, NY. The high frame rate achieves a significant overlap and the image frame is large; to reduce the computational load, every tenth frame was selected in the subset of 477 images, giving a separation of 5 seconds of flight time between images and a total of 48 images to be used for the reconstruction. The 5 seconds in flight time results in an estimated 90% overlap between the imagery. Due to the circular flight pattern, there is no side-lap to estimate, but there is an additional rotational component between consecutive images.

The full scene reconstruction, containing over two million points, is shown in Figure 2.2 along with an aerial image from Google Maps of the same region for comparison purposes. The Genesee river is easily discernible, defined by a lack of points, and the highways that loop

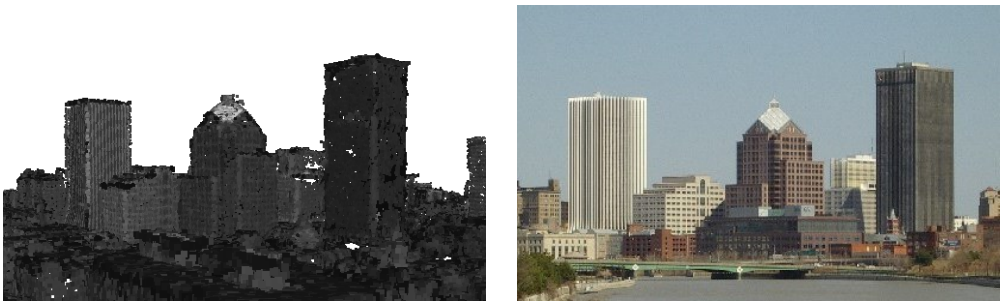


*Figure 2.3: Reconstruction of the Rochester, NY skyline as compared to an image of the city skyline [40].*

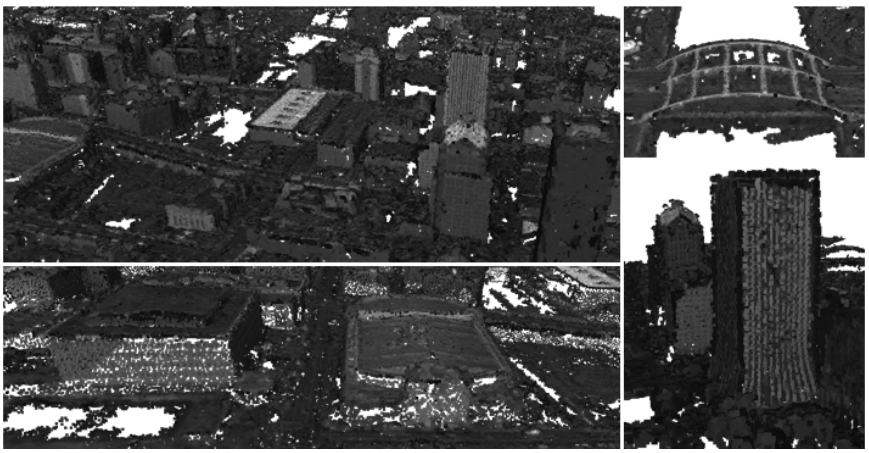


*Figure 2.4: Top Left: City view. Top Right: Fredrick Douglass bridge. Bottom Left: Monroe County Civic Center and Blue Cross Arena. Bottom Right: Chase building.*

around the city are also perceptible. Knowledge of the city may reveal other details at this high level, such as the Blue Cross Arena, the Riverside Convention Center, and the Fredrick Douglass Bridge that crosses the Genesee river on the inner loop (I-490).

A more detailed view of the reconstructed city skyline is shown in Figure 2.3, along with a comparison image for reference. The major difference in this reconstruction using oblique imagery, as opposed to previous reconstructions with only nadir imagery, is that the sides of buildings in the scene are reconstructed because they were visible in the imagery when previously they were not. More detailed views of the point cloud are shown in Figure 2.4.

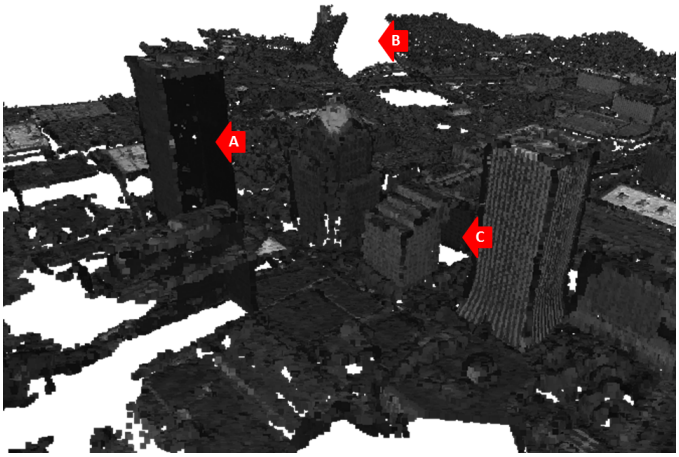



*Figure 2.5: View of the reconstruction that clearly depicts some of the voids in the point cloud. Particular areas of interest are labeled for further discussion.*

## 2.2 Voids in Point Clouds

The results of the SfM workflow shown in Section 2.1.7 are promising, though not without imperfection. Voids in the point cloud, areas that lack points and appear to be missing information, are readily apparent by visual inspection. Certain voids, such as small holes in surfaces, may be easy to fill using a few basic assumptions and known surface reconstruction and/or hole filling algorithms, discussed in Sections 2.3 and 2.4 respectively. The problem is with larger voids, particularly those present in the large point clouds generated from aerial imagery, as they can contain multiple surfaces and are not as easily filled.

Before discussing methods for filling the voids, it is important to develop an understanding for why they exist. To illustrate this, a view of a point cloud showcasing some of the voids is shown in Figure 2.5. Note that this is the same reconstruction that was shown in Section 2.1.7. There are small holes throughout the model, as well as the larger, more obvious, holes. A human observer is also able to discern areas where building walls are missing (two such walls are denoted by the arrows labeled A and C). The river, noted by the arrow labeled B, is distinguished by a distinct lack of points.

There are many potential reasons for the voids in the point cloud. The most obvious reason

is that multiple views of an area were not included in the input imagery, or a constant obscuration was present. Voids can also be a result of texturally difficult areas that resulted in poor image correspondences, an insufficient baseline between images that resulted in poor triangulation, or mismatched image correspondences that resulted in large triangulation errors.

### 2.2.1 Lack of Coverage

First consider the problem of lack of coverage. As stated previously, this is one of the most obvious reasons for a void in the point cloud because an area that was not present in the input imagery cannot possibly be reconstructed. A similar case can be made for constant obscuration resulting from overlapping geometries. There needs to be a clear line of sight between the camera and the area of interest, and the area must be visible in a minimum of three views in order to have a chance at reconstruction.

The void labeled A in Figure 2.5 is an example of lack of coverage. In the particular sequence of images used, that building rotated out of the frame multiple times, such that only three sides of the building were ever visible. The void labeled C in Figure 2.5, at first glance may appear to be the result of constant obscuration, as the building is surrounded by taller structures. The original input images from the reconstruction, cropped to feature the building of interest, are shown in Figure 2.6. The images clearly capture all sides of the building, albeit at a steep angle in some cases, but there are multiple views of each side of the building. However, due to the algorithms used in the workflow, particularly the CMVS algorithm, not all 48 initial images were used in the final dense reconstruction performed by the PMVS algorithm. Those image highlighted in green in Figure 2.6 were used, and those highlighted in red were flagged as redundant images and removed from the set. When considering only those views in green, there are not enough views of the particular portion of the building that is missing for it to have been reconstructed.

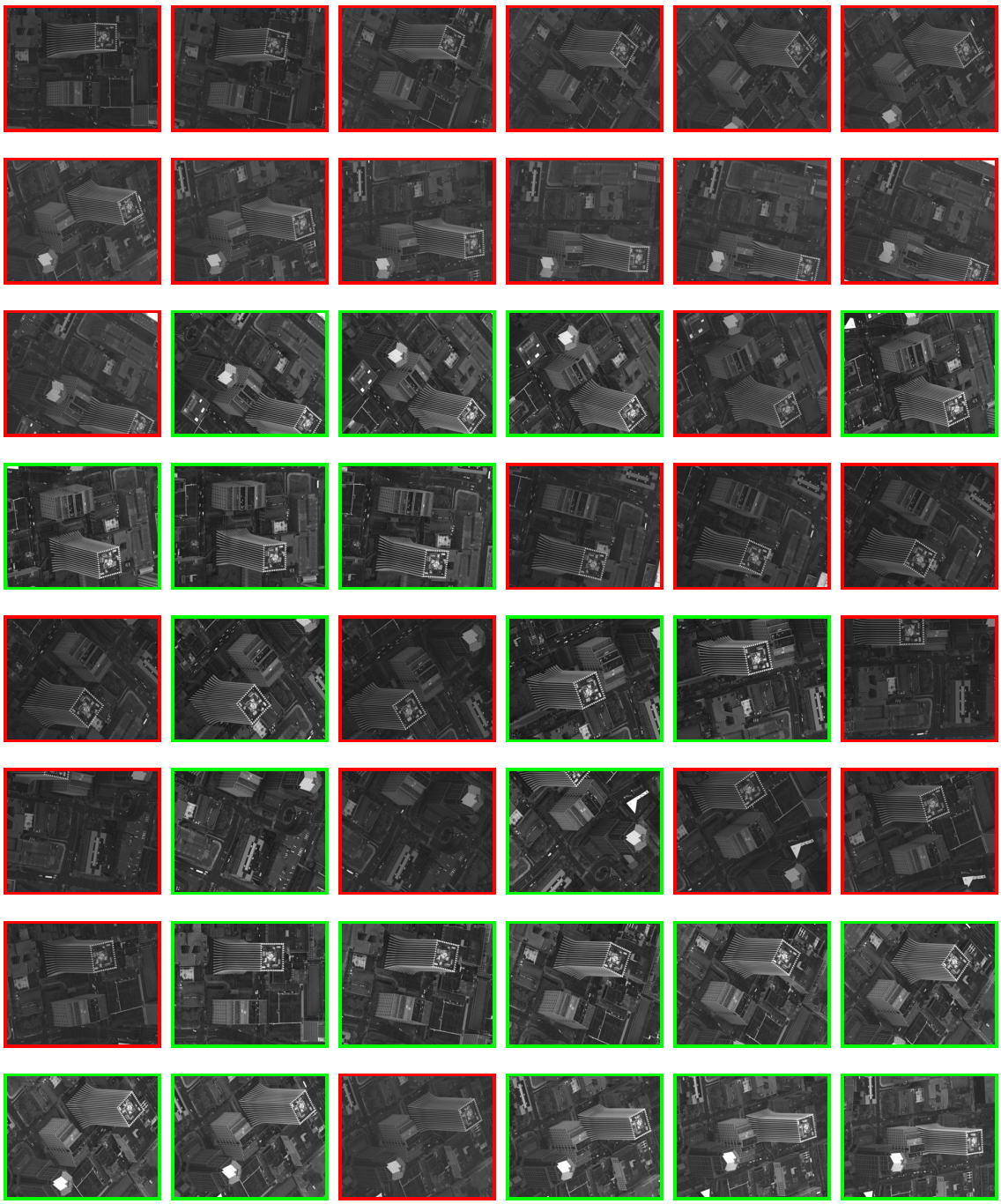


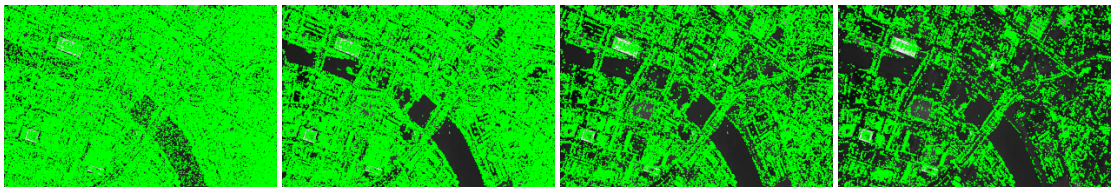
*Figure 2.6: The set of 48 input images used in the reconstruction in Section 2.1.7, cropped to feature buildings of interest in the downtown region. Images highlighted in green were used in the final dense reconstruction by PMVS, and image highlighted in red were removed by the CMVS algorithm and not included in the final reconstruction.*

### 2.2.2 Texturally Difficult Regions

Now consider the case of texturally difficult regions in the imagery. A texturally difficult region can be a homogeneous region that fails to generate features, but it can also be a region of repetitive texture that results in non-unique features that are not easily matched. Feature detection and matching is a crucial component of reconstruction, and it fails in two ways on such surfaces. First and foremost, these surfaces may not be particularly interesting with respect to texture and/or gray-level gradients and therefore may fail to generate features. Consider the river, noted by the absence of points in the reconstruction, labeled B in Figure 2.5. Though present in every input image, the river is a large homogeneous region with few points of interest. The SIFT algorithm, discussed in Section 2.1.1, was run on one of the WAMI images used for reconstruction. Peaks of the difference-of-Gaussian scale space were filtered using different thresholds to remove those that were too small. The feature locations were displayed on the image and the results are shown in Figure 2.7. Note that there are fewer features in the river than anywhere else in the image, even before a threshold is applied (indicated by the 0.0 threshold image). As the threshold increases, the feature locations in the river are the first to disappear. In some sense, this is indicative of the stability of the features themselves, as more stable features will have larger peaks.

The stability of a feature leads to the second reason for failure, and that is in the matching. Features need to be stable in the sense that they will be generated across multiple views such that there is at least a chance of finding a match. Not only does a feature need to be stable, it also needs to be unique. Feature descriptors with multiple potential matches are discarded. Because texturally flat surfaces are not particularly interesting, it is possible that the features generated across the surface will be too similar to one another such that there are no stable and unique matches. The same idea can be applied to regions with spatially repeating patterns, again because there is not a way to distinguish one area from another. Regions that may be texturally repetitive could include geometric repetitions, such as architectural design on structures, but can also include more organic repetitions, such as regions of trees, grass, etc.

Saponaro *et al.*[42] implemented an approach to fill in holes in SfM point clouds that resulted from textureless regions using Shape from Shading (SfS) techniques. SfS determines the shape of an object, up to scale, from a single image and performs better than SfM in textureless regions. Regions that lack texture often still contain shading information that varies smoothly,



(a) 0.0 (b) 0.5 (c) 1.0 (d) 1.5

*Figure 2.7: Feature detection performed on a WAMI image, using the indicated threshold to filter the peaks that were too small; feature locations are labeled in green. Images generated using VLFeat [41].*

which is what SfS techniques use to determine object shapes. The proposed approach uses a SfS based method, known as gradient constrained interpolation, to fill in holes in SfM point clouds. The work used standard SfM algorithms, and successfully filled in holes in textureless regions, focusing on icescapes in particular, which can be difficult to reconstruct using SfM alone.

## 2.3 Surface Reconstruction

In general, the objective of 3D reconstruction is a complete model of the object and/or scene. Point cloud models, including those resulting from SfM methods discussed previously, are not directly usable in most 3D applications, rather the models are made of geometric primitives. These geometric primitives are extracted from the point cloud automatically to obtain a surface model through a process commonly referred to as surface reconstruction. Reconstruction of surfaces from points is not a straightforward problem. Point sampling and spacing is non-uniform, positions (and normals if available) are noisy, and some regions are lacking data due to obscuration and accessibility limitations, regardless of application. Given these issues, surface reconstruction algorithms are designed to infer the topology of the unknown surface, while accurately fitting but not over-fitting or under-fitting the noisy data.

The most popular surface reconstruction techniques can be classified as geometric and implicit methods. Geometric surface reconstruction methods are proximity-based methods that employ mechanisms such as Delaunay triangulation (e.g. alpha shapes [43]) and region growing (e.g. ball pivoting [44]). These techniques tend to leave holes in undersampled regions, particularly ones dependent on balls and shapes as it can be difficult to find a radius that will

bridge holes without bridging the fine surface details of the model. Implicit methods are by far the more popular methods. These methods generally produce hole-free models, but have difficulty with surfaces that have boundaries, as they will often interpolate gaps or extend the surface across boundaries. Poisson surface reconstruction is an example of an implicit polynomial fitting algorithm [45]. There are also volumetric methods that are often used as part of reconstruction and hole-filling algorithms (e.g. marching cubes [46] and dual contouring [47]). In general, these algorithms cannot handle a set of unorganized points directly, but require that the data is converted to a volumetric representation on a regularly spaced grid.

It is important to address the idea of sampling density in reconstruction. Given adequate sampling density, a topologically consistent surface can be constructed, as proven by Amenta *et al* [48] and others. Intuitively, the density demands will vary based on the complexity of a surface, as smooth surfaces can be reconstructed with fewer samples. In the point cloud data of interest here, the sampling density varies greatly across the dataset. Texturally rich and distinctive areas result in a dense sampling of points, whereas texturally difficult areas result in few points due to the nature of the feature detection and matching algorithms employed in the dense stages of reconstruction. In addition, the holes in the point cloud present an additional challenge for reconstruction.

Surface reconstruction techniques are often application specific, designed to address a particular type of data. Some algorithms focus on ordered datasets, such as those that would come from a range scanner, and other focus on unorganized datasets, such as those that may result from SfM techniques. In addition, some algorithms simply require the points as input, while others require additional information such as surface normals. As the surface reconstruction problem grows and expands, it becomes more apparent that reconstructing the entirety of a surface (particularly considering large point clouds over large surface areas) may not be the best approach. Assuming that individual buildings can be extracted from a larger point cloud, Zhou *et al* [49] present a method just for reconstructing buildings. It is not difficult to surmise that multiple algorithms may be necessary to reconstruct a single scene, as some algorithms may perform well with manmade structures, and others with more organic materials.

Due to the constant evolution and rapid expansion of the field, this is not an exhaustive review of the field of surface reconstruction. Rather, the following is a review of common reconstruction algorithms, designed to provide a sampling of the different approaches. More specifically, it is designed to illustrate why it may not be feasible to apply these algorithms to

the types of point clouds being addressed here, particularly because of the non-uniform sampling and large gaps in the data.

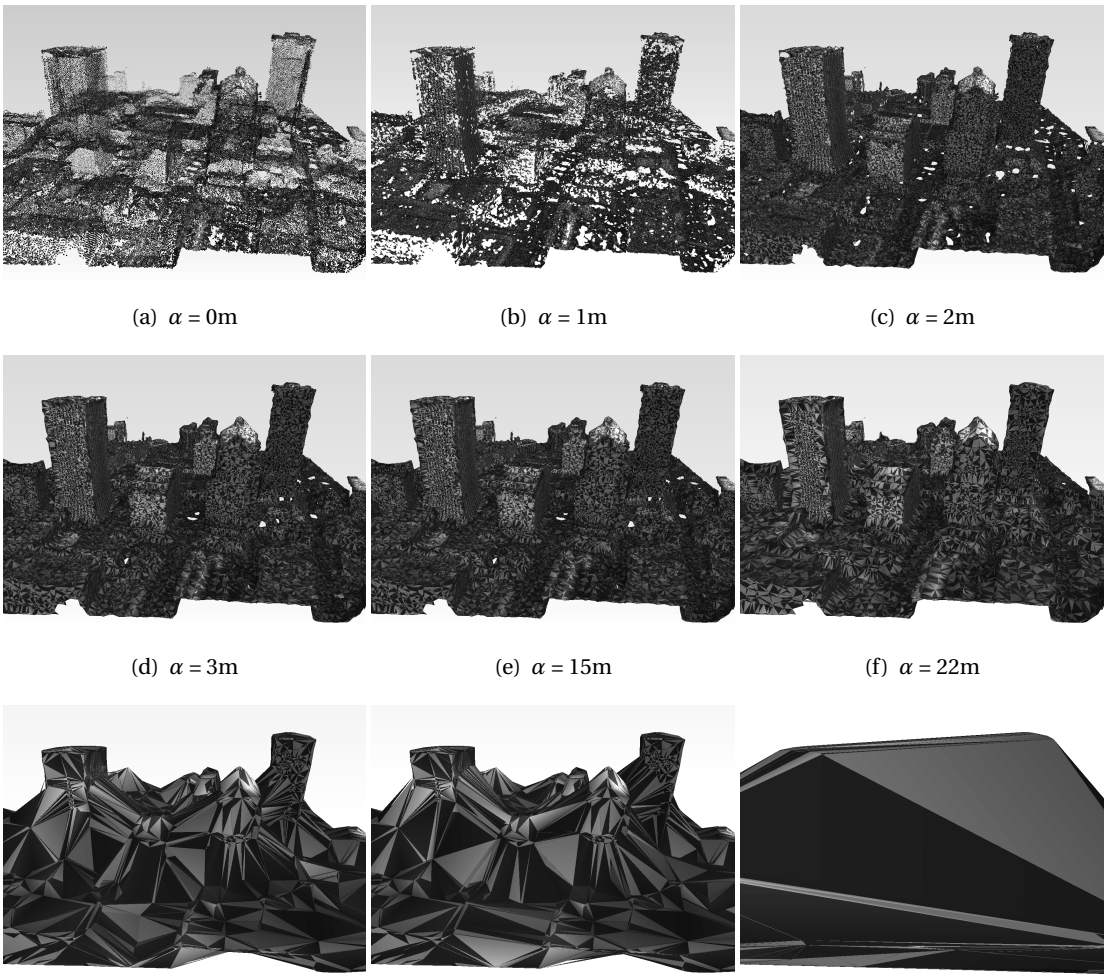
### 2.3.1 Alpha Shapes

Alpha shapes are a generalization of the convex hull of a point set. In mathematics, the convex hull of a point set is given by the smallest convex set that contains the points, where a set is considered to be convex if every line segment that joins two points in the set, is also within the set. Every convex hull is an alpha shape, but not every alpha shape is a convex hull.

Let $S \in \mathbb{R}^3$ be a finite set of points, and $\alpha \in \mathbb{R}$ where $0 \leq \alpha \leq \infty$. When $\alpha = \infty$, the $\alpha$-shape and convex hull are the same, but as $\alpha$ decreases, the $\alpha$-shape will develop cavities, tunnels, and holes as the shape molds itself to the point set. Consider the $\alpha$-shape where $\alpha = \infty$; intuitively pieces of this shape will disappear as $\alpha$ decreases, such that spheres with a radius $\alpha$ can occupy portions of the space without encompassing any of the points in $S$. The resulting object is an $\alpha$-hull, and the $\alpha$-shape is obtained by substituting straight edges for the circular ones and triangles for the spherical caps [43]. The resulting $\alpha$-shape is not guaranteed to be convex nor is it guaranteed to be connected as the shape itself can contain components such as single points, edges, and patches of triangles. The family of $\alpha$-shapes can be represented implicitly by the Delaunay triangulation of the point set $S$ [43].

While a convex hull may be able to adequately describe a simplistic rectangular building, it is by definition unable to describe a building with any concavity. For this reason, convex hulls and $\alpha$-shapes where $\alpha = \infty$ are inadequate for the types of surface reconstructions considered here. Similarly, when $\alpha < \infty$, the alpha shape reconstruction is by definition, likely to contain holes. While a hole-free surface would be the ideal model, there is a trade-off to consider between the presence of holes in the reconstruction and a smooth model that, while hole-free, may be grossly inaccurate.

Meshlab [50] was used to compute the alpha shape for the point cloud that was shown in Section 2.1.7 and the results are shown in Figure 2.8 for various values of $\alpha$. This reconstruction is less than ideal, as there are not only several holes in the model, but the surfaces themselves appear rough and jagged due to the triangulation. A smaller $\alpha$ value could reduce some of the jaggedness of the model, but the number of holes will increase, and ultimately this will not achieve a complete reconstruction.



(a) $\alpha = 0$m (b) $\alpha = 1$m (c) $\alpha = 2$m

(d) $\alpha = 3$m (e) $\alpha = 15$m (f) $\alpha = 22$m

(g) $\alpha = 27$m (h) $\alpha = 30$m (i) $\alpha = \infty$m

*Figure 2.8: Nine different α-shapes for increasing values of α are shown. The points, α=0, are shown in (a), and the convex hull, $\alpha = \infty$, is shown in (i). The size of the triangles and connectivity increases as α increases. Smaller values of α result in more holes in the model, but large values of α connect components that should not necessarily be connected, removing some of the finer details.*

While $\alpha$-shapes may be the best-known approach, other surface reconstruction algorithms have been based on Delaunay triangulation. The crust algorithm, presented by Amenta *et al* [48], effectively automatically computes local values of $\alpha$, which allows sampling density of the input model to vary, overcoming the major drawback of $\alpha$-shapes. Yau *et al* [51] extended the crust algorithm to include the concept of inner and outer poles. Methods based on Delaunay triangulation usually result in the reconstructed surface model passing through the original sample points. Whether this is beneficial or desirable is dependent on the reliability and noise levels of the input point cloud, as well as the application.

### 2.3.2 Ball-Pivoting

The ball-pivoting algorithm (BPA) is a conceptually simple method for surface reconstruction based on region growing. Region growing methods start with a seed, often a triangle in the case of surface reconstruction, then consider new points to join to the existing boundary; this process is repeated until all points have been considered. BPA assumes that points are distributed over the entire surface with adequate sampling frequency for the application, and that an estimate of the surface normal is available for each point [44].

With an initial seed triangle, a ball with a radius $p$ is placed in contact with the initial three sample points and then, keeping in contact with two of the points, the ball is "pivoted" until it reaches another point [44]. A 2D illustration of BPA is shown in Figure 2.9, including the effect of low sampling density and high curvature on the resulting model. This process is repeated as the ball pivots around the edge of the mesh boundary, and the triplets of points that the ball contacts form the triangles in the mesh model. Surface normals are used to distinguish between internal and external regions and ensure consistency in the reconstruction. The use of these surface normals is particularly important in the presence of imperfect data, where sampling is not uniform, a very common occurrence across multiple modalities.

Because of the nature of the ball pivoting, knowledge of the sampling density and feature sizes are necessary to choose an appropriate radius for BPA. In areas with a higher sampling density, the ball will still pivot on top of the surface, though choosing a sampling spacing too large can result in a loss of smaller feature details. Areas with lower sampling densities are problematic because the ball with "fall through" and this will result in holes in the model. To address the holes, BPA can be run iteratively with multiple passes that use increasing ball radii,

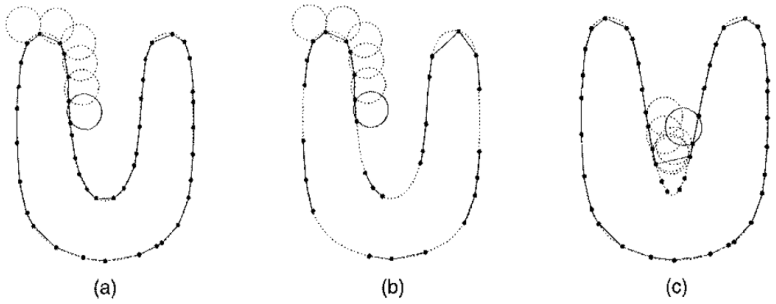
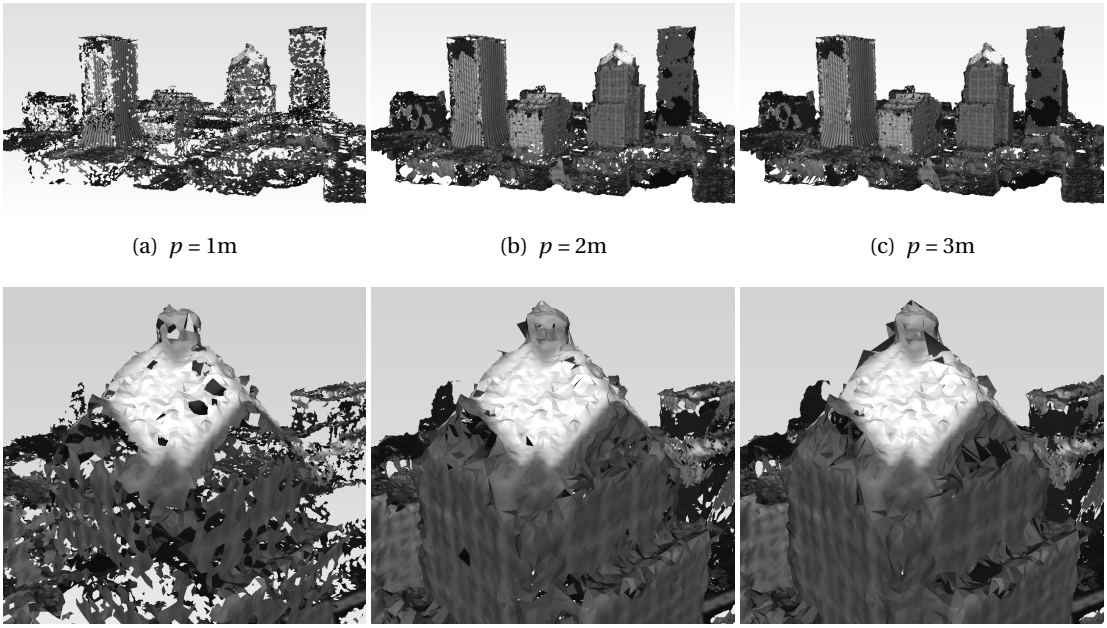



*Figure 2.9: Ball Pivoting Algorithm illustrated in 2D, where a circle with radius p pivots from point to point, connecting them with edges. (a) Adequate sampling density results in a fully closed surface. (b) Low sampling density results in holes, due to some edges not being created. (c) Curvature in the surface larger than* $1/p$ *results in missed features, due to sampling points not being reached by the pivoting ball. Figure from Bernardini [44].*

however there is no specification to be able to only fill certain holes.

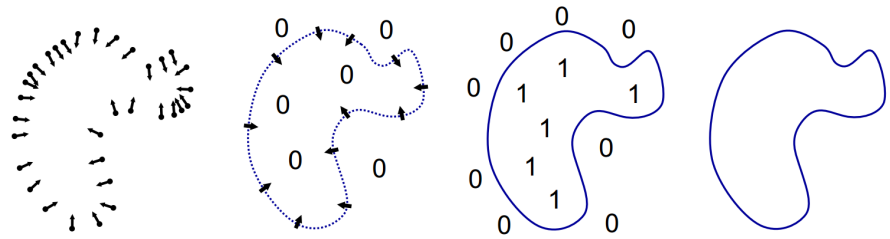
Again, Meshlab [50] was used to compute a BPA surface reconstruction of the point cloud. An initial radius of $p$=1m was used, followed by $p$=2m, and $p$=3m. A subset of the point cloud was used to speed up computation time, though this should have no impact on the reconstruction. The resulting surface is much smoother than that obtained with $\alpha$-shapes, though still exhibits holes, as to be expected due to the varying sampling densities. As can be seen, holes in the initial reconstruction can be filled by increasing the ball radii. However even with multiple passes, there are still holes in the model and a complete surface reconstruction is not obtained.

Region growing methods such as BPA can also be referred to as incremental surface construction or advancing-front surface construction algorithms, and there are many others of this type. Boissonnat [52] defined an edge-based algorithm that started with an edge and iteratively attached triangles at boundary edges of the emerging surface. Gopi *et al.* [53] also use a progressive triangulation technique to generate a mesh from a point set with no assumptions about underlying geometry.



(a) $p = 1\text{m}$ (b) $p = 2\text{m}$ (c) $p = 3\text{m}$

*Figure 2.10: BPA reconstruction of a point cloud. The algorithm was initially run with p=1m (a), then p=2m (b), and finally p=3m (c). On the first pass 262,526 faces were created, 84,449 faces were added on the second pass, and 18,610 faces were added on the third pass. As can be seen, the initial pass left a lot of holes and gaps in the surfaces. Subsequent passes were able to fill in some, but not all, of these holes. As a result, the final model still contains holes.*



(a) Oriented Points (b) Indicator Gradient (c) Indicator Function (d) Surface

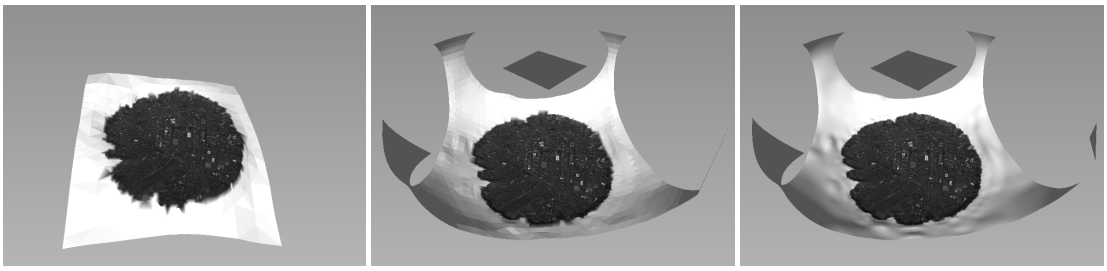
*Figure 2.11: A 2D, intuitive illustration of Poisson surface reconstruction. Given a set of points with surface normals (a), the surface normal can be assumed to be a sample of the indicator gradient (b), which can be used to derive the indicator function (c). The indicator function is defined such that it is unity inside the surface, and zero outside the surface and can be used to derive the final surface model (d). Figure from Kazhdan, et al [45].*

### 2.3.3 Poisson Surface Reconstruction

Poisson surface reconstruction uses an implicit function framework, expressing the surface reconstruction as a solution to a Poisson equation [45]. The solution is independent of view direction and considers all points at once, and therefore does not need to resort to partitioning or blending. The resulting model is resilient to noise and is composed of smooth surfaces. The objective of the algorithm is to reconstruct a watertight, triangulated model of the surface.

For a Poisson reconstruction the input data is a set of points, each associated with an inward-facing surface normal, assumed to lie on or near the surface of the unknown model. A 3D indicator function, which is a binary function defined to be unity inside the model and zero outside the model, is computed and then used to extract the appropriate isosurface as the reconstructed surface. The key insight of the Poisson algorithm is that the gradient of the indicator function is zero almost everywhere except at the surface, where it is equal to the inward surface normal; thus it can be assumed that the points are a sampling of the gradient of the indicator function. An intuitive illustration of Poisson surface reconstruction is shown in 2D in Figure 2.11.

In order to handle the demands of reconstructing large datasets, a parallel implementation of the Poisson surface reconstruction algorithm was implemented, designed to run on multiprocessor systems with distributed memory [54]. The parallel algorithm demonstrated both



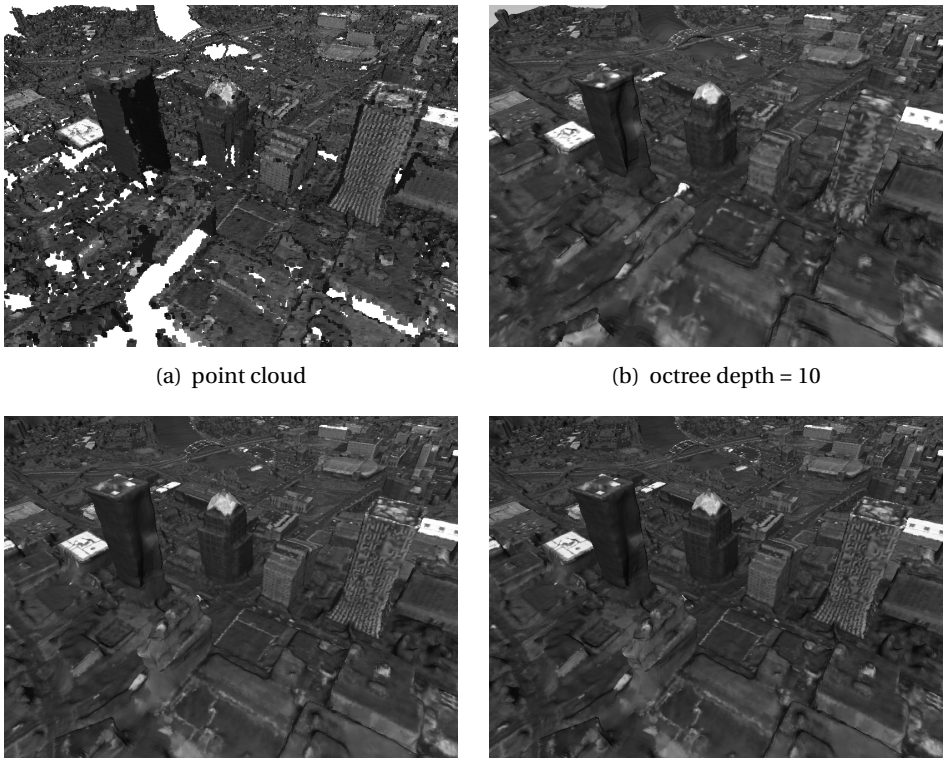
(a) octree depth = 10 (b) octree depth = 12 (c) octree depth = 14

*Figure 2.12: Full scene Poisson surface reconstruction of a point cloud at different octree depths. Note the additional surfaces, not part of the original point cloud, that have been added as part of the reconstruction, shown in white.*

scalability and an equivalence to the serial implementation as far as reconstruction accuracy. Another group of researchers expanded on the Poisson idea to improve efficiency, using a linear interpolation and quicker searching method to obtain the final isosurface [55].

An implementation of the Poisson surface reconstruction algorithm is available in Meshlab [50]. The point cloud shown in Section 2.1.7 was reconstructed using the algorithm at different octree depths. The octree depth in the Poisson reconstruction is related to the number of functions that are used to define a surface, thus finer details can be reconstructed with a higher octree depth, but there is a trade-off as the higher depth requires more memory and computation time. A full view of the reconstructions at octree depths of 10, 12, and 14 is shown in Figure 2.12. Octree depths above 14 could not be computed due to insufficient memory. Notice that there is an additional surface present in each of the point cloud reconstructions, extending beyond the original points. This is likely due to the fact that the reconstruction algorithm is designed to reconstruct watertight models, something that will never be achieved with this kind of data, and its attempt to do so has introduced these extraneous surfaces. At depths of 12 and 14, the surface curves upward, seemingly to approximate a sphere, and additional surfaces are added far above the point cloud itself.

From a distant perspective, the models created from the surface reconstruction may look promising. However a closer examination of the reconstruction will begin to reveal flaws, as shown in Figure 2.13. The original point cloud was shown for comparison purposes, alongside the three different Poisson reconstructions. Examination of the point cloud reveals that the tall



(a) point cloud (b) octree depth = 10

(c) octree depth = 12 (d) octree depth = 14

*Figure 2.13: A close up view of the point cloud and corresponding Poisson surface reconstructions at different octree depths. This view in particular was chosen because of the obvious holes in the point cloud model, highlighting that a simple surface reconstruction cannot necessarily achieve quality results, depending on the hole present.*

building on the left is missing one side entirely, and the shorter building in foreground is missing several sides. Poisson reconstructions with higher octree depths are able to capture finer details in the surfaces because higher resolution functions are used to fit the indicator function at higher depths, and improvement can be noted as the octree depth is increased in the model. Despite the improvement with increased octree depth, both buildings exhibit significant deformations in the areas that are missing information because it is trying to reconstruct a portion of the scene with no information about the surface that should be there and the resulting approximation is poor.

### 2.3.4 Moving Least Squares

Moving Least Squares (MLS) is another method for constructing continuous surfaces out of a set of unorganized points, using an extension of well-known least squares methods. In this case, the 'moving' refers to the weights that are applied to individual points to calculate their contribution to the surface solution at a given location. The weights are biased toward the region around which the surface is being calculated. Unlike the often jagged results from triangulation based methods, MLS provides smooth surfaces. One important thing to note is that MLS surfaces are not guaranteed to pass through the original points, but rather the surfaces pass near the points. Like many other algorithms, MLS methods have trouble in areas that are undersampled, and it is sensitive to outliers.

MLS surfaces were originally introduced by Levin [56], and later adopted by Alexa *et al.* [57] for surface reconstruction from point set surfaces. Kolluri [58] improved upon traditional MLS methods by including surface normals. The algebraic point set surfaces (APSS) introduced by Guennebaud *et al.* [59] are based on moving least squares, but aim to fit a higher order algebraic sphere rather than a plane. In cases where plane fitting fails, such as undersampling and sheet separation, spheres have improved stability. Öztireli *et al.* [60] combined implicit MLS methods with statistics in order to improve robustness against outliers and retain sharp features in the point cloud without over smoothing. MLS also serves as the basis for some hole-filling algorithms, as discussed in Section 2.4.

### 2.3.5 Marching Cubes

Marching cubes is a popular computer graphics algorithm for extracting a polygonal mesh from a 3D scalar field, and is particularly applicable to voxel spaces. The algorithm was originally developed for medical data, such that quality 3D visualizations could be obtained from computed tomography, magnetic resonance, and single photon emission computed tomography data [46]. The marching cube algorithm is designed to work with data that is presented in a grid, and therefore is not well suited for point cloud reconstruction. However, it is frequently used as a component of volumetric algorithms that require the scan-conversion, also known as voxelization, of a point cloud.

The algorithm proceeds through space taking eight neighboring locations into account at a time to determine the polygon(s) necessary to represent the surface that passes through the

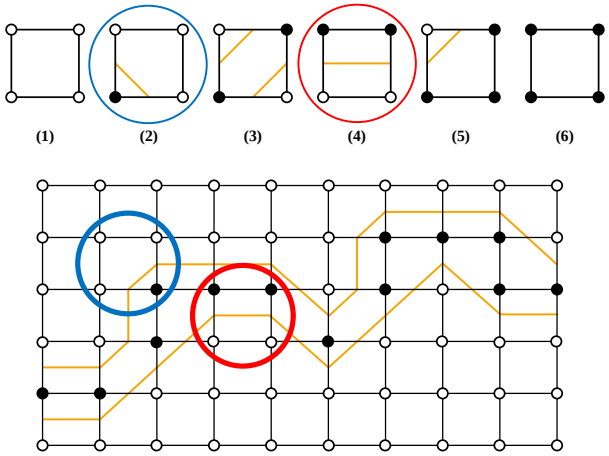



*Figure 2.14: Illustration of the marching cubes algorithm in 2D. The 6 2D primitives are shown at the top, and an example scalar field is shown below, with the corresponding surface. Two primitives have been highlighted and their location in the example is indicated.*

resulting cube. It is based on a threshold value for the scalar field, and points on the cube are assigned a value of unity if they exceed or equal the value of the surface threshold and are inside the surface, and those below receive a zero value and are outside the surface. The surface intersects the cube if the cube contains vertices that are both inside and outside the surface. Due to the eight corners of the cube and two possible states, there are 256 ways for the surface to intersect the cube. Symmetry of the cube allows for a reduction of this number. The original marching cubes algorithm [46] reduced to 15 different primitives (including the case where there was no surface intersection and all vertices were the same). The original method was shown to cause ambiguities and result in meshes with holes, and was improved upon to obtain topologically consistent models; an extended version presented 33 primitives [61]. An illustration of marching cubes in 2D is shown in Figure 2.14, with the 6 2D primitives and an example scalar field with the resulting surface.

### 2.3.6 Dual Contouring

In dual contouring methods, such as Surface Nets [62], vertices are free to move inside the cube, and not constrained to lie on the edges of the grid as in cube-based methods, such as marching cubes. Dual contouring is another method designed to work with data that is partitioned into

a grid. More specifically, it requires Hermite data which includes the intersection points of the surface with the grid, in addition to surface normals. Vertices are generated for each cube exhibiting a sign change, positioned at the minimum of a quadratic error function such that they are consistent with the normals [47].

Zhou *et al* [49] presented a dual-contouring method to create 2.5D building models from aerial LIDAR (Light Detection and Ranging) point cloud data. In this case, the 2.5D is based on the assumption that a building is composed of complex roof components that are connected with vertical walls. A disadvantage to this 2.5D approach is that it cannot handle overlapping geometries, such as overhangs on a building. Similar to marching cubes, dual contouring algorithms cannot be applied directly to a point cloud for surface reconstruction without some sort of voxelization process.

## 2.4 Hole Filling

As was seen in the previous section discussing surface reconstruction, many reconstruction methods will result in surface models that contain holes (*e.g.* alpha shapes and ball pivoting). Surface reconstruction is often used in conjunction with hole filling algorithms to address this problem. Holes and undesirable effects can be introduced in models for a variety of reasons. Depending on the algorithm used, surface reconstruction can leave holes in the resulting mesh model and therefore hole filling is an important part of object and scene reconstructions. Much of the work in developing hole filling algorithms has been derived from 3D scanning and range finding applications. Occlusions, missing views, and accessibility limitations are causes of holes and missing data in 3D scanning applications, in addition to problems with low reflectance, and constraints on scanner placement. While the datasets themselves may be large and contain millions of points, typically the scans are of figurines, statues, or single rooms involving a few cubic meters, and nothing on a city-wide scale. While a review of hole filling algorithms is certainly appropriate, it is important to note that most are simply not designed to handle the problems present in large-scale 3D point clouds derived from aerial imagery, nor would they be successful at doing so accurately. City-wide reconstructions may contain millions of points, but the point density is often low in comparison to data obtained from close range 3D scanners. The low point density and the variation in point density throughout a scene make the prospect of hole filling difficult. In addition, many hole filling algorithms are based

on the premise of a water-tight model. While this may be achievable for a singular building, it is generally not possible in such large-scale models that contain numerous objects.

Holes can be filled by addressing them in the surface reconstruction itself, as discussed in Section 2.3, or hole filling algorithms can be applied as a post-processing step, making them compatible with multiple kinds of reconstruction algorithms. In general, a hole filling algorithm should identify holes, and find appropriate parametrizations to reconstruct missing parts using all available information, resulting in plausible geometry with non-self-intersecting surfaces. Simple holes can often be filled with discs or smooth patches that meet the boundary conditions of the hole, and this technique works well in cases where holes are small in comparison to the geometric variation of the surface. However, the problem becomes more difficult when the holes grow in size and/or have multiple boundary components because the geometric variation of the surface becomes increasingly complex. In this case, there are multiple possible topologies that could fill the area. The possibility of a complex hole that includes twists and folds increases as the size of the hole grows, and this seemingly rare convoluted geometry occurs frequently in the presence of joints and crevices. The potential holes in point clouds derived from aerial imagery are almost guaranteed to be complex.

Due to the popularity of 3D scanning, the extent of the research in the area of hole filling is vast. Though there are many ways of grouping the algorithms, two major model repair groupings will be considered here: mesh-based algorithms and volumetric algorithms.

#### Mesh-Based Model Repair

Mesh-based repair algorithms operate directly on the polygons of the model to repair any geometric and topological errors that may exist. Typically only regions with defects are addressed, and the resulting surface models are not guaranteed to be closed and potentially contain self-intersecting polygons.

Liepa [63] presented an algorithm to fill holes in oriented, connected manifold triangular meshes. Holes are identified by finding boundary edges, where a boundary edge is adjacent to a single triangle in the mesh. Once a hole is identified, an initial triangulation of the hole is determined, using a minimum area triangulation accompanied by a weighting function that takes into account the angle between the normals of the triangles in order to avoid creating sharp folds and non-manifold edges. A smoothing process is then applied in order to fill the

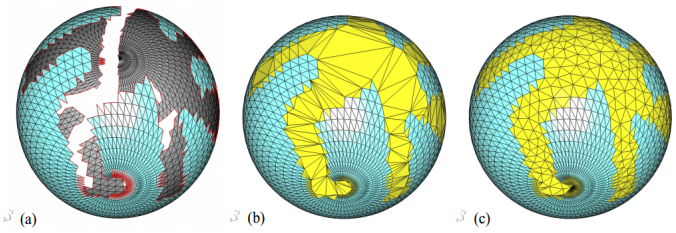



*Figure 2.15: An example of a complex hole, filled by the algorithm presented by Liepa et al [63]. (a) A complex hole that almost circumnavigates the sphere. (b) Initial triangulation. (c) Refined mesh with density approximating surrounding mesh. Figure from Liepa et al [63].*

hole with a mesh that will approximate the density of the surrounding mesh. An example of a complex hole and the resulting filled mesh is shown in Figure 2.15. Due to the assumption of having a connected manifold, this algorithm is limited and cannot handle large gaps between surfaces or holes with islands without some modification.

Wang *et al* [64] present a hole filling technique that can be performed after meshing, and thus any surface reconstruction technique that results in a triangular mesh can be used. Boundary edges belonging to a single triangle in the triangulated mesh are used as indicators of holes in the model. Once a hole boundary is identified, a reference plane for the hole vicinity is computed from the points on the boundary edge. Moving least squares is then used to interpolate the hole, and reconstructed patches will blend smoothly into the original mesh while still preserving the original boundary points. An illustration of the hole filling process is shown in Figure 2.16. Processing is limited to the vicinity of the holes, and the sampling frequency in the vicinity is recreated. This process is repeated until there are no holes left in the model. The algorithm is likely to fail if the hole presents twists or folds that do not define a one-to-one mapping function when projected onto the reference plane, a situation that is increasingly likely as the size of the hole grows.

#### Volumetric Model Repair

A volumetric approach to model repair will impose a volumetric grid on the surface, and the output surface is reconstructed from the grid. The key to this voxelization-type approach is the determination of whether a grid point lies inside or outside the model, often referred to as a sign function. It is important to note that these methods do not necessarily preserve the

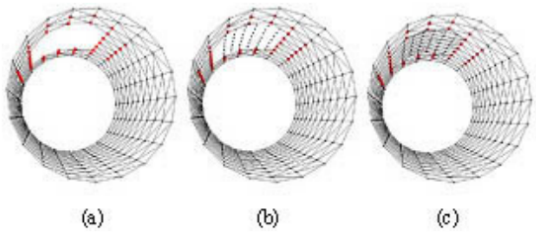



*Figure 2.16: A cylinder with a hole is shown in (a) and the vicinity of the hole is identified with red. New points are added in (b) and the resulting reconstructed mesh is shown in (c). Figure from Wang et al. [64].*

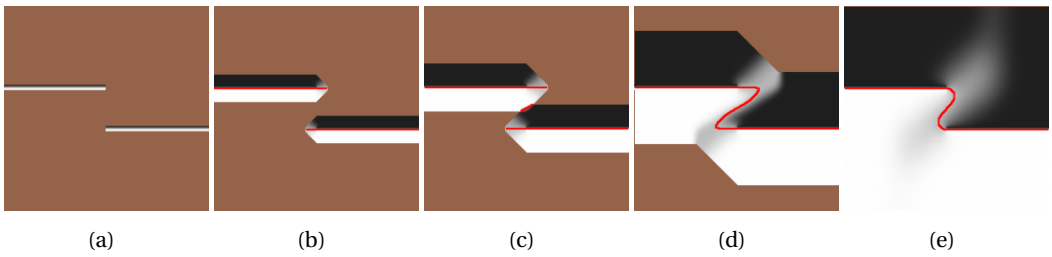
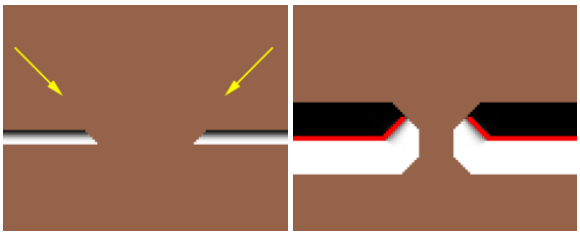



*Figure 2.17: Illustration of Davis' volumetric diffusion algorithm used to fill holes. The source term is shown in (a), where the grayscale values represent the signed distance function with black being outside the model and white inside. The diffusion process begins to extend the surfaces in (b), where the red curve denotes the zero set. The surfaces begin to interact in (c), the hole closes in (d), and the final converged shape is shown in (e). Figures from Davis et al [65].*

original points of the model.

Davis *et al* [65] present a method for building water-tight models from surfaces with holes that are topologically complex such that they cannot be filled by triangulation. The surface model is converted to a voxel-based representation, where voxel values are defined near the surface in a relatively narrow band; values are positive inside the surface, negative outside the surface, and zero on the surface. The objective of the algorithm is to diffuse the values out from the surface into undefined areas in order to complete the surface model. An illustration of this is shown in Figure 2.17.

This algorithm was designed for range scanning data, and was implemented in such a way as to take advantage of line-of-sight information given by the range scanner to indicate where surfaces are, in addition to where surfaces are not. Constraints are built-in to control how far a surface can extend into a known empty region and thus there is a trade-off between the



(a) (b)

*Figure 2.18: Due to the tendency of the diffusion process to propagate perpendicular to boundaries, the hole filling method presented by Davis et al. has trouble filling holes correctly for oblique scans. Diffusion of the source term in (a) results in boundaries that are incorrectly angled upwards as shown in (b). Figures from Davis et al [65].*

smoothness of the model, and how far it will protrude into an empty region. Due to the diffusion process and its tendency to propagate perpendicularly to boundaries, this approach has difficulties with the angled boundaries introduced by oblique scans, resulting in reconstructed surfaces that are incorrectly angled upward. An illustration of this is shown in Figure 2.18.

Verdera et al. [66] used a volumetric approach similar to that presented by Davis *et al* [65]. The major difference between the two approaches is the system of partial differential equations that is used to smoothly continue the surface of interest.

Ju [67] also presents a method for building water-tight models, using an octree grid that is more space efficient than uniform volumetric grids. The model repair is implemented as a three step process: (1) the input model is embedded in a uniformly-spaced grid and intersection edges are identified, (2) signs are generated that are consistent with the intersection edges, and (3) a closed surface is reconstructed on the grid by contouring. The intersection edges are edges of voxels that intersect polygons in the model; exact intersection points can be recorded to obtain a better surface reconstruction later. Signs are generated such that each intersection edge exhibits a sign change, thereby giving a consistent sign configuration. Contouring algorithms are then used to reconstruct the surface that separates grid points with opposite signs; if the intersection points were stored, an algorithm such as marching cubes (Section 2.3.5) can be used. An illustration that depicts the model at various steps in the process is shown in Figure 2.19. Areas with complex holes, highly curved shapes, or multiple boundaries may not result in optimal looking repaired regions due to the simplistic nature of this approach.

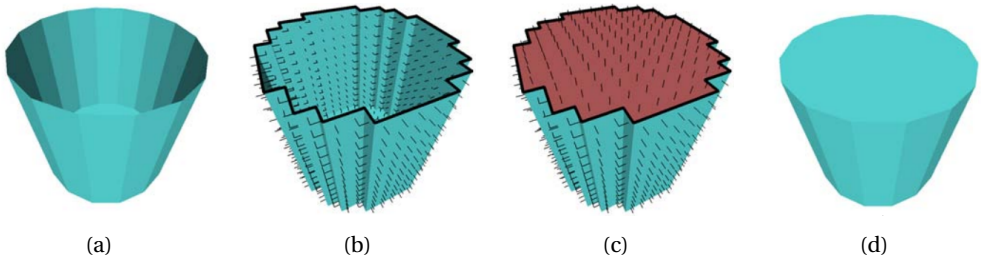



*Figure 2.19: Illustration of Ju's polygonal model repair algorithm used to fill holes. The input model is shown in (a), the edges intersected and corresponding dual surface is shown in (b), the patched dual surface is shown in (c), and the repaired model is shown in (d). Figures from Ju [67].*

Sharf *et al* [68] introduce a context-based method to complete surfaces, inspired by the recent advances in texture synthesis and image completion methods. Figure 2.20 shows the difference between smooth filling and context-based filling methods. In this approach, characteristics of the surface are analyzed and holes are filled iteratively by copying patches from other regions of the surface. More specifically, the point cloud data is divided into cells, where surface cells intersect the current surface approximation; a surface cell is valid if there are a sufficient number of points for surface representation in the cell, and invalid if it contains less points. The objective is to fill the invalid cells by copying content of valid surface cells that match the surface approximation around the empty cell. This process is repeated until cells contain enough points to be accepted as a final surface representation, and can be further refined by subdividing cells to achieve a higher level of detail. In this instance, the relationship between the sampling density and the detail frequency of the model is important, because cells must be sufficiently small to be able capture the high frequency content of the model. Due to the reliance on a local surface approximation to provide a reasonable initial smooth surface, this process is particularly susceptible to failure under the presence of noise and undersampling. In addition, surface completion areas can only contain copies of the sample set, and therefore may result in poor matches in cases where no appropriate examples exist. While it is possible that the point clouds contain examples of the missing surfaces, it is unlikely that examples of all missing surfaces are contained within the point cloud.

(a) Original Model (b) Marked Area (c) Smooth Filling (d) Context-Based Filling

*Figure 2.20: Illustration showing the differences between smooth hole filling and context-based filling presented by Sharf et al. The original model is shown in (a), and a marked portion that was removed to demonstrate surface completion techniques is shown in (b). The results of smooth filling are shown in (c) and context-based filling are shown in (d). Figures from Sharf et al [68].*

## 2.5 LIDAR

It would be remiss not to mention that image-based SfM techniques are not the only viable method of achieving a 3D point cloud. One of the most influential additions to the remote sensing community has been that of LIDAR (Light Detection and Ranging). LIDAR takes advantage of collimated laser light to gather information about distant objects based on their backscatter radiation. Topographical LIDAR is based on laser ranging coupled with GPS/INS data. When coupling surface interaction points obtained using the laser with the position and orientation information from the GPS/INS, the surface interaction points can be determined in absolute world coordinates, thereby resulting in a 3D map of the scene. Ground-based (close-range) and aerial (long-range) LIDAR systems have been used to generate 3D models in the form of point clouds.

Given that LIDAR is an active imaging system, it is not always a feasible solution to generate surface maps, hence the increased interest in passive based systems using SfM techniques. However, many of the surface reconstruction and hole filling algorithms mentioned previously result from research involving 3D scanners, and most quality metrics and techniques that can be applied to a LIDAR point cloud are directly applicable to SfM point clouds, thus a discussion of such things seems appropriate.

### 2.5.1 Quality Metrics

One thing that exists in the LIDAR community that is lacking in the SfM community is the definition of quality metrics for the point clouds. The level of detail in a LIDAR scan is defined by two primary spatial metrics: point spacing and point density. Point spacing is defined as the nominal 2D spacing between LIDAR points and their neighbors within the point cloud. Point density is defined as the nominal number of LIDAR points per square meter. As defined by the ASPRS (American Society for Remote Sensing and Photogrammetry), nominal is defined to mean that 95% of the data are closer/denser than the value given [69]. Density is often the only metric calculated and is approximated by subdividing the dataset into a regular grid and calculating the metric for each block. Both metrics can be significantly biased by areas that only contain a few points, but the most limiting factor for each of these metrics is that they are calculated from a 2D projection of the 3D data. While multiple definitions of these terms exist, and admittedly they may not be the most effective metrics, both could be applied to point clouds that result from SfM data.

### 2.5.2 LIDAR Derived Geometry

Point clouds are the direct result of LIDAR processing and provide a basis for any further processing to derive scene geometry, such as surface reconstruction. While any of the techniques discussed in Section 2.3 would be suitable for a LIDAR point cloud, there are a few popular techniques that have yet to be mentioned.

#### Triangulated Irregular Network Mapping

One of the most popular reconstruction methods in the field is Triangulated Irregular Network (TIN) mapping [70]. The objective of this method is to reconstruct a continuous surface from the point cloud by connecting all of the points. This is achieved using triangular faces, typically by employing some type of Delaunay triangulation method, effectively giving the TIN map a variable resolution based on point density. The largest limitation of this method is that it is performed on a 2D projection of the 3D point cloud and thus cannot handle overlapping geometries. As such, it is more suitable for digital elevation model (DEM) creation, its intended purpose, than generating models of urban scenes with complex geometries.

### Voxelization

Voxels provide another method for representing 3D information that can handle more complex geometries. Voxels use 3D blocks (*i.e.* small volumetric elements) on a regular grid to represent 3D volumes, much the same way that pixels are used to represent 2D images. Voxels have an inherent ability to be able to handle overlapping objects, such as those that may be found in scene reconstruction. Voxels have been widely used for visualization of data in the medical and scientific communities, in addition to their use in multi-image 3D reconstruction [22]. Some of the surface reconstruction methods from Section 2.3, such as marching cubes and dual contouring, required voxelization as a pre-processing step.

One of the downsides to voxelization of a scene is that details within the scene that are smaller than the voxel size will be removed during the process. This can be mitigated provided there are sufficient input data to support a voxel map of the desired resolution. If straight voxelization is used for reconstruction, stair-step artifacts can be introduced in the resulting models due to the nature of voxels. This does not have to be the case if a more complex method of reconstruction based on voxelization is chosen. Finally, due to the nature of voxels, there is the inherently large storage requirement. Compression techniques could be used to reduce the data volume, given that many of the voxels in a real-world scene will not contain scene structure.

One method for voxelizing a point cloud is hit-counting, where the objective is to measure the amount of occlusion amassed in each voxel based on the number of points it contains [71, 72, 73]. There are two assumptions made in hit-counting: (1) the occlusion of a voxel is proportional to the number of points in the voxel, and (2) voxels that contain no samples are empty. A normalization factor must be used to convert from number of hits to amount of occlusion [74]. Using a normalization factor assumes uniform sampling, which is not true of either a LIDAR or SfM point cloud. In addition, voxels that do not contain points are not guaranteed to be transparent as both truly empty voxels and unsampled voxels will not contain hits.

A better approach would be a transmission approach that uses information about how the point cloud was derived, which can provide better voxelization. By using the sensor position (which will apply to both LIDAR and SfM points), the entire path of the ray can be examined to analyze obscuration [75, 76]. This analysis can be used to define a transmission equation on a per voxel basis [77, 78], given the number of points within the a voxel and the number of rays

that passed through it. This approach requires no normalization factor, and the transmission of a voxel can be refined as more samples and rays are added. In addition, it provides a way to differentiate between truly empty voxels and unsampled voxels.

❄ ❄ ❄

While the generation of 3D models from imagery is continually improving thanks to advancements in SfM techniques, the models often exhibit holes or voids. These holes can be the result of a lack of coverage in the input imagery, but can also be the result of homogeneous, texturally flat areas that fail to generate features, fail to provide consistent matches, and/or fail to achieve an accurate triangulation. Surface reconstruction algorithms that are applied to the point clouds are used to approximate surfaces, and in some cases can adequately estimate the missing regions, provided they are simplistic and small. However, in cases where the missing region is significant and contains complex geometries, surface reconstruction and hole-filling techniques will fail to generate surfaces or the estimated surfaces will be grossly inaccurate.

As feature detectors and dense stereo matching techniques improve, it is possible that the denser point clouds that are produced will result in fewer holes in subsequent models. However, the voids in the point cloud that are a result of lack of coverage in the imagery will see no significant improvements. The problem being addressed here is related to the voids in the point cloud, particularly those resulting from a lack of coverage, and the approach will be two-pronged. First, a method to identify the voids in the point cloud will be established, including techniques to differentiate between the types of void. Second, a method to address filling the voids will be explored. The voxel-based approaches being used in the LIDAR community will provide a foundation for this approach.

# Chapter 3

# Methodology

The objective of 3D reconstruction from imagery is to accurately reconstruct the scene with no a priori knowledge of the world. The previous chapter discussed the Structure from Motion (SfM) process in detail, from generating image-to-image correspondences through dense stereo matching that results in a 3D point cloud. In general, point clouds are not directly usable in most 3D applications, so reconstruction techniques are applied to achieve a surface model. The problem with this workflow is the voids that are often present in SfM reconstructions, where voids arise in regions where there was a lack of coverage in the imagery, or in regions that failed to generate image-to-image correspondences, failed to accurately match correspondences, or failed to triangulate properly.

This chapter develops methods to identify these voids in the point cloud using visibility analysis and identify potential image locations for regions that lacked image coverage. This process takes advantage of information in the reconstruction process that is generally discarded to analyze the visibility of the points in the point cloud from the camera positions used to reconstruct the scene. This visibility information is used to carve out the line-of-sight from cameras in a voxel space, such that voxels can be identified as containing structure or free space; the remaining voxels are unsampled. Voids in the voxel space manifest in the unsampled voxels. Original imagery is exploited to identify homogeneous regions that failed to reconstruct, using a texture metric to separate such regions from others that failed to reconstruct due to a lack of coverage or obscuration. It is believed that inclusion of more imagery in the reconstruction process can fill in the voids in the point cloud that are not a result of textu-

rally difficult regions, and a line-of-sight analysis is used to determine potential locations for additional imagery.

## 3.1 Visibility Analysis

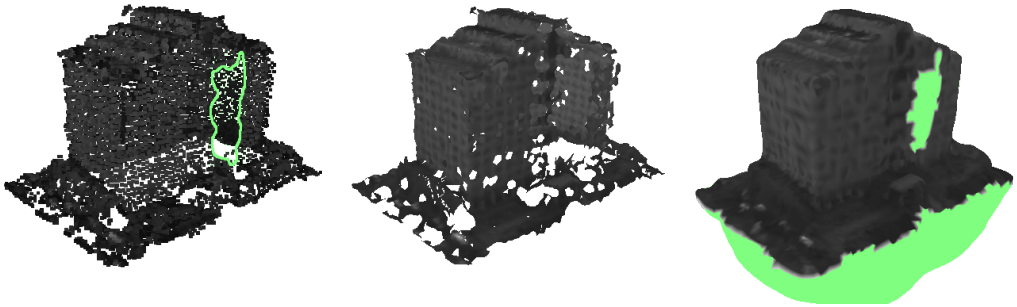
While the 3D structure, derived from SfM techniques, represented by the point cloud provides a good starting place, it is not representative of all the information contained in the data and generally is not an acceptable end point for most 3D applications. In a simplistic representation, a 3D point generated during scene reconstruction is a result of triangulating image-to-image correspondences where the point was visible in the imagery. Thus, the images that contained correspondences for a specific point provide visibility information. More specifically, the rays between the cameras used in reconstruction and the 3D point must be devoid of obstruction such that the path between each camera and the 3D point is clear. This type of visibility information is computed, either implicitly or explicitly, during the reconstruction process, but it is often discarded once the final point cloud is obtained. Typically the point cloud data is then used as input to a surface reconstruction algorithm to obtain a surface-based reconstruction made up of geometric primitives.

### 3.1.1 Surface-Based Models

A surface-based model provides a good basis for many objects because it is inherently simplistic. In addition, it provides a bridge between the mapping and rendering communities. Generating surfaces from points has been the standard method used to build a model from point clouds in both SfM and LIDAR applications, and specific algorithms were discussed at length in Section 2.3.

One potential downside in the context of a surface model is that there is often an implied continuity of the surface. While this may be the case in large-scale applications such as terrain mapping, it is not necessarily the case in urban scenes, particularly in regards to the point clouds of interest here. Forced surface continuity may result in large oscillations in the surface or inaccurate surface estimates.

The voids present in SfM point clouds also provide a challenge for traditional surface-based reconstruction models. As discussed in Sections 2.3 and 2.4, the complexity of the hole deter-



(a) Planar Hole (b) BPA (c) Poisson

*Figure 3.1: An example of a simple, planar void in a point cloud. (a) The Clinton Square building was extracted from the WAMI point cloud in Section 2.1.7, and exhibits a void on one of the sides (highlighted in green). (b) BPA surface reconstruction of the points, performed in Meshlab with automatic settings. (c) Poison surface reconstruction, performed in Meshlab with an octree depth of 12, where surfaces were colored with the associated points except in areas where that information was unavailable and are shown in green. The Poisson reconstruction does a nice job estimating the missing area, filling in the hole.*

mines how accurately that surface can be reconstructed or how well that hole can be filled. A hole on a planar surface is easily filled, but the complex holes with multiple intersecting surface, often present in SfM point clouds, are much more challenging. Three areas of interest from the WAMI point cloud shown in Section 2.1.7 were selected to illustrate the complexity of voids in point clouds: (1) the Clinton Square building, (2) the HSBC building, and (3) the Fredrick Douglass bridge. BPA and Poisson surface reconstructions were performed with built in functions in Meshlab [50], where the radius for each BPA algorithm was automatically selected based on point density, and the Poisson reconstructions were run with an octree depth of 12. Note that due to the nature of Poisson surface reconstruction, additional surfaces are reconstructed outside the bounds of the points in an effort to construct a watertight model.

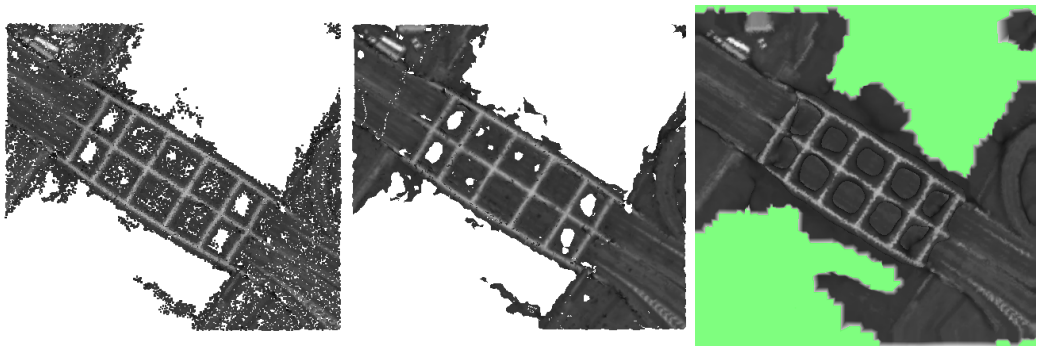
The Clinton Square building, shown in Figure 3.1, is missing a portion of one side, outlined in green for clarity. This particular hole was chosen as it represents a relatively simple missing region in the point cloud that could be filled with a planar surface. The BPA surface reconstruction contains several smaller holes in addition to the larger hole that was meant to be the focus. Iteratively performing BPA with increasing radii would likely fill in some of those holes, while a simple hole filling algorithm would likely be successful with the larger hole due to its planar

(a) Complex Hole (b) BPA (c) Poisson

*Figure 3.2: An example of a complex void in a point cloud. (a) The HSBC building was extracted from the WAMI point cloud in Section 2.1.7, and exhibits two sides that are completely devoid of points. (b) BPA surface reconstruction of the points, performed in Meshlab with automatic settings. This reconstructs surfaces where the density is high, but is unable to reconstruct the missing sides (c) Poison surface reconstruction, performed in Meshlab with an octree depth of 12, where surfaces were colored with the associated points except in areas where that information was unavailable and are shown in green. Though the Poisson reconstruction attempts to fill in all surfaces, the estimation is wildly inaccurate in the areas where the points were missing.*

nature. The Poisson algorithm accurately fills in the hole in this case, though color information is not available for the reconstructed faces.

A more challenging void in the point cloud is presented in Figure 3.2, with the HSBC building, where multiple views of all sides of the building were not present in the input imagery. The building itself has two different roof levels, which are densely populated with points, in addition to one side that is densely populated. One other side of the building contains points at varying densities, while the two remaining sides exhibit a complete lack of points. While it is visually discernible as a building and a human observer would likely be able to fill in the missing parts, this type of void is considered complex in regard to surface reconstruction and hole filling due to the multiple intersecting surfaces that are missing. Voids of this nature are common in SfM point clouds, particularly where multiple views of the region were lacking in the input imagery as was the case here. The BPA reconstruction performs similarly to the previous example; regions that have an adequate sampling density are reconstructed well, and there are holes in the regions with a lower density. Again, this could be mitigated by iteratively running the BPA algorithm, which would fix some of the smaller holes, but likely would have little im-



(a) Bridge (b) BPA (c) Poisson

*Figure 3.3: An example of a bridge in a point cloud with overlapping structure. (a) The Frederick Douglass bridge was extracted from the WAMI point cloud in Section 2.1.7, and is well reconstructed in the point cloud. (b) BPA surface reconstruction of the points, performed in Meshlab with automatic settings. (c) Poison surface reconstruction, performed in Meshlab with an octree depth of 12, where surfaces were colored with the associated points except in areas where that information was unavailable and are shown in green. The Poisson reconstruction does a nice job estimating both the bridge and the surrounding missing area in the river.*

pact on the missing sides of the building. A very large radius could potentially fill in the missing sides, but the results would not be an accurate representation of the surface. The Poisson surface reconstruction of the building does manage to bridge the gaps, but does so with a wildly inaccurate estimation, thereby providing an unacceptable reconstruction.

The final example is the Fredrick Douglass Bridge, shown in Figure 3.3. The bridge is fairly well reconstructed in the point cloud, but provides a unique example of overlapping geometry as both the bridge trusses and street below are visible. Without knowledge of the object or surrounding space, this region could easily be singled out as a potential area that contains holes. In this instance, both the BPA and Poisson surface reconstructions are able to distinguish the bridge trusses from the surface beneath. This may not be the case with other reconstruction methods.

### 3.1.2 Voxel-Based Models

Voxels provide another method for representing 3D information that can handle complex geometries and provide a suitable framework for this application. A voxel is a 3D block (*i.e.* a small

volumetric element) on a regular grid that is used to represent 3D volumes, much the same way that pixels are used to represent 2D images. Voxels make no assumptions regarding overlapping geometries and there is no implied continuity in the resulting geometry. A voxel model is ideal for general purpose object representation and can easily represent complex scene geometry with overlapping objects such as buildings, bridges, trees, and building overhangs.

Due to its simplicity, the most common method used to construct a voxel model is hit-counting, also known as voxel binning, due to its simplicity. During voxelization, the scene volume is divided up into a regular grid and the number of points in each voxel is tallied. For most geometric representations, a threshold of some count is applied to the voxel map to classify voxels as either surfaces (containing sufficient points) or empty (containing insufficient points). This technique was used by Pyysalo *et al.* [72], to perform line-of-sight calculations using a voxel map derived from LIDAR, where any voxel containing LIDAR points was considered to be opaque. Alternatively, the point counts can be used to derive estimates of physical parameters, such as opacity, as done by Levick *et al.*[73] or Mutlu *et al.*[74]

There are two assumptions made in hit-counting: (1) the occlusion of a voxel is proportional to the number of points in the voxel, and (2) voxels that contain no samples are empty. A normalization factor must be used to convert from number of hits to amount of occlusion [74]. Using a normalization factor to convert the number of hits to occlusion, such as Mutlu *et al.*[74], assumes uniform sampling, which is not true of SfM point clouds. In addition, voxels that do not contain points are not guaranteed to be transparent as both truly empty voxels and unsampled voxels will not contain hits.

The primary disadvantage in voxelization is the loss of detail in the gridding process. Objects or details in the scene that are smaller than the voxel resolution cannot be represented. Therefore, a large voxel size (undersampling) could result in a loss of detail. In a similar respect, the resolution of the voxel map is limited by the density of the point cloud. A small voxel size (oversampling) will result in undersampled volumes that do not adequately represent the scene. An additional consideration with the SfM point clouds is the inconsistent sampling density. Densely sampled areas of the point cloud may be able to support a higher voxel resolution than other areas. This could be mitigated by using an adaptive sampling approach by subdividing voxels based on point density, allowing more detail in areas that have the density to support a higher resolution. Due to the increased complexity, this adaptive approach will be reserved for future work.

Another potential downside to using voxels is the inherent large memory requirement. Including fine details in a scene requires a smaller voxel size, thus using more voxels, and consequentially more storage, to represent the same amount of space. Modern computing capabilities can handle reasonably sized scenes, though there may be some difficulties with large scenes depending on the voxel size. Given that much of the voxel space contains open air and no point data, some sort of compression scheme designed for use on the voxel data could mitigate some of the memory requirements, but further discussion of this is beyond the scope of this research.

The approach presented here defines a voxel space with an origin $(X_o, Y_o, Z_o)$, number of elements $(n_x, n_y, n_z)$, and the element size $dv$. For the purposes of this research, only cubic voxels are considered, thus the use of a single element size. The size of the voxel space is determined by the bounding rectangular cuboid of the point cloud, where the bounding cuboid is given using the minimum and maximum coordinate values of the points. Note that the discrete number of elements and the flooring operations used in the computation of the voxel space origin will result in a slightly larger voxel volume than that explicitly defined by the bounding cuboid. Due to the nature of SfM point clouds and their irregular boundaries based on the input imagery, the boundary of the voxel space will contain regions outside of the point cloud due to the enforcement of the cuboid shape.

Once the voxel space is defined, the vertices from the point cloud are used to identify occupied voxels. Given a 3D point $\vec{X}$, the voxel $\mathbf{P}$ that contains the point is given by

$$\mathbf{P} = \left\lfloor \frac{\vec{X} - \vec{O}}{dv} \right\rfloor \tag{3.1}$$

where $\vec{O}$ is the voxel space origin, and $dv$ is the block size of the voxels. At this stage, the point cloud data has been voxelized and is fully represented within the voxel space.

The voxelized models of the simple (Clinton Square building) and complex (HSBC building and Fredrick Douglass bridge) holes are shown in Figure 3.4. The surface voxels were treated as opaque, while all other voxels were treated as transparent as would be the case with a hit counting method. The shading on the surface was added for visualization purposes using Blender [79]. Using just the voxels to represent the surface introduces a stair-step artifact in the models. An algorithm such as marching cubes or dual contouring could also be used to make a model from the voxelized point cloud and this could reduce this artifact. While the voxelized point

(a) Simple Hole (b) Complex Hole (c) Complex Hole - Bridge

*Figure 3.4: Voxelized models of the regions of interest shown in Figures 3.1, 3.2, and 3.3, representing both simple and complex holes in the point cloud. Using hit counting voxel methods alone does not enhance the resulting model, as there are still holes and no additional information has been gained.*

clouds provide different methods to obtain a model, there is not necessarily more information provided than could be obtained from a traditional surface-based model.

At this point, it may seem that voxels do not provide enough added benefit to justify their usage, as surface-based methods can accurately define surfaces without the stair-step artifacts that are introduced with voxelization. However, the primary advantage to using a voxel map in this case is the ability to represent the concept of free space, something that a simple surface-based model cannot handle.

### 3.1.3 Incorporating Ray Origins

As stated previously, visibility information for each point is computed during the generation of the point cloud. Given that a camera was used to reconstruct a point, the point must be visible in the image and the ray between the camera origin and the point must be devoid of obstruction, providing a clear line of sight. Though generally discarded after the final SfM point cloud is obtained, this information can be used to identify free space and the voxel map provides a convenient spatial framework to aggregate this information. The camera position defines a ray origin for the line of sight, and these sightlines can be used to carve out areas surrounding the scene that should be free space. By definition, this free space cannot contain scene structure.

The concept of free space has been applied to voxel models previously in the computer vision community in the development of navigation systems for mobile robots, particularly those operating in unknown environments. Multi-dimensional random fields that maintain estimates of the occupancy state of a cell in a spacial lattice, known as occupancy grids, were

noted to show promise for robot navigation and perception by Elfes [80]. Depth information, derived from either rangefinding or stereo imagery, is used to construct occupancy grids where cells are marked as empty or full. The occupancy grids are updated with new information as additional viewpoints are observed, creating a more complete representation of the local scene. Murray *et al.* [81] presented a correlation-based stereo technique to generate occupancy grids for autonomous robot navigation. Though there is a slight difference in terminology used, the cells of the occupancy grids are equivalent to the voxels of the voxel space.

The implication of empty voxels is an important concept, as it provides more information than could be obtained from just the surface model. By incorporating the idea of free space, it is possible to take advantage of both the voxel representation and the additional visibility information that is usually ignored.

It should be noted that the following approach is based on point clouds for which visibility information is available. In addition to the point cloud data, the following information is required for use of this approach:

- List of cameras with good visibility for each point
- Location of each camera
- Principle axis of each camera

It is important that the ground plane of the point cloud nominally aligns with the $X-Y$ plane. In this case, the geo-accurate transformation used as a post-processing step in the 3D workflow achieves this using the GPS centers of the cameras to compute a transform. In the absence of GPS locations, it may also be possible to perform a principle component analysis to pull out the dominant axes. It is important to align at least the Z-axis of the point cloud with the Z-axis of the fixed Earth-based coordinate system because it is better for visualization purposes and.

Both the Bundler and PMVS algorithms used in the SfM workflow supply the necessary information. Bundler outputs a file that contains estimated scene and camera geometry, where each camera entry contains intrinsic and extrinsic parameters, and each point entry contains the point location, color and a list of views in which the point was visible [82]. PMVS also provides full reconstruction information including files for each camera that contain the corresponding camera projection matrix, and reconstruction information for each point, including

the 3D location and estimated surface normal, a photometric consistency score, the image indicies in which the point is visible and the textures agree well, and the image indices in which the textures may not agree well but the point should be visible [83]. For the purposes of this research, the PMVS output will be used to build the voxel space, as the denser point cloud from PMVS will not limit the voxel resolution to the same extent as the sparse point clouds output by Bundler.

With the incorporation of ray origins into the voxel workflow, inclusion of the camera centers within the boundaries of the voxel space should be considered. Initial algorithm testing included the vertices of the camera centers in the set of points used to define the bounding cuboid of the voxel space. This proved to be useful for testing and visualization purposes, but memory requirements expanded rapidly. Consider an urban scene with the highest point at 300m, imaged at an altitute of 3000m; storage requirements for the voxel space increase 10 times in the z-direction alone by including the space between the scene and the camera. The voxels in the space above the point cloud are likely to be free space, and their inclusion will not provide any additional useful information about the scene below. By using just the bounding cuboid of the point cloud data (and not the bounding cuboid of the point cloud and camera centers), both the memory requirements and the processing time are significantly reduced without a notable impact on the results. Therefore, it is assumed that any voxels above the bounded voxel space are unoccluded.

A fast voxel traversal algorithm [84] can be used to identify voxels on the ray defined by the camera and the surface point by computing the intersection of the ray with the voxel map. Traditionally there are two types of intersection algorithms: space partitioning and bounding volumes. A space partitioning algorithm divides a space up into smaller partitions (e.g. voxels), and looks for intersections in the voxels along the given ray, continuing until either an intersection has been found or the space partition has been completely traversed. A bounding volumes algorithm is based on the premise that if there is no intersection with the bounding volume of a complicated object, then there is no need to intersect the complicated object within. This approach will use a combination of the two techniques.

First consider the traversal of the voxel space itself, given a starting point inside the voxel space as shown in Figure 3.5. Assume the equation of the ray is given by $\vec{r} + \vec{s}t$, where $t >= 0$. Starting at the origin of the ray, $\vec{r}$, the algorithm must traverse through the voxels a, b, c, d, e, f, g, and h in that order to reach the terminating point. A space partitioning algorithm is used to

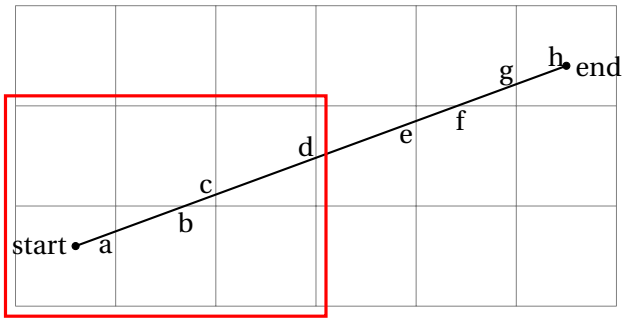



*Figure 3.5: Example of a voxel space, with a ray originating at* **start** *and terminating at* **end** *(adapted from Amanatides [84]). A traversal algorithm must visit voxels a, b, c, d, e, f, g, ang h to correctly traverse the grid. The highlighted red portion is enlarged and shown in more detail in Figure 3.6.*

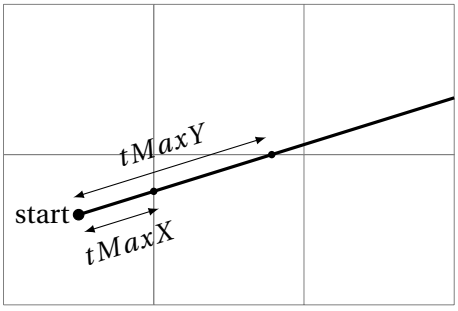



*Figure 3.6: Enlarged view of Figure 3.5, showing the incremental traversal from the point* **start**. *The vertical and horizontal boundary crossings are distances $tMaxX$ and $tMaxY$ from* **start** *respectively.*

identify the voxels along the ray. Traversal is initialized by determining the voxel in which the ray originates, given by equation 3.1. Let the variables $X$, $Y$, and $Z$ be the starting voxel coordinates, and the variables $stepX$, $stepY$, and $stepZ$ indicate whether the voxel coordinates are incremented or decremented when crossing voxel boundaries, determined by the sign of the components of $\vec{s}$. Following the initialization of variables, the incremental traversal through the voxel space is achieved by computing distances, $t$, along the ray, to determine where it crosses voxel boundaries in each dimension. If the stepping direction is positive, the distance to the boundary is given by

$$tMax_i = \frac{\vec{O}_i + dv \cdot (currentBlock_i + 1) - currentPos_i}{\vec{s}_i} \tag{3.2}$$

where $i$ indicates the $x$, $y$, or $z$ direction, $currentBlock$ is a 3 element integer vector with the current voxel indices, and $currentPos$ is a 3 element floating point vector at the current position given by $\vec{r} + \vec{s}t$. If the stepping direction is negative, the distance to the boundary is given by

$$tMax_i = \frac{currentPos_i - (\vec{O}_i + dv \cdot currentBlock_i)}{\vec{s}_i} \tag{3.3}$$

Values are calculated in each of the dimensions, such that $tMaxX$, $tMaxY$, and $tMaxZ$

are obtained. The minimum value indicates the distance that must be traveled along the ray in order to step into the next voxel. The two dimensional case is shown in Figure 3.6, where the distance to cross the vertical boundary is given by $tMaxX$ and the distance to cross a horizontal boundary is given by $tMaxY$. In this case, the algorithm would take a step in the x-direction because $tMaxX < tMaxY$. The basic loop of the incremental phase of the traversal algorithm is shown in algorithm 1.

**Algorithm 1** Voxel Traversal Loop (adapted from Amanatides [84])

**Input:** Voxel map, and a ray with origin $\vec{r}$ and direction $\vec{s}$
**Output:** List of voxels along the ray, and the current position $\vec{p}$

**loop**
  **if** $tMaxX < tMaxY$ **then**
    **if** $tMaxX < tMaxZ$ **then**
      $X = X + stepX$
      $\vec{p} = \vec{p} + \vec{s} \cdot tMaxX$
    **else**
      $Z = Z + stepZ$
      $\vec{p} = \vec{p} + \vec{s} \cdot tMaxZ$
  **else**
    **if** $tMaxY < tMaxZ$ **then**
      $Y = Y + stepY$
      $\vec{p} = \vec{p} + \vec{s} \cdot tMaxY$
    **else**
      $Z = Z + stepZ$
      $\vec{p} = \vec{p} + \vec{s} \cdot tMaxZ$

Given a point in the point cloud and a camera center that was used to reconstruct that point, the ray defined by the two points can be used as input to this fast voxel traversal algorithm, which will return the voxels along the ray. Therefore a method for identifying the free voxels is attained.

As was stated previously, the voxel space does not extend all the way to the camera centers and thus the camera center resides outside of the voxel space. To address this problem, the traversal algorithm is modified by incorporating a bounding volumes algorithm. A ray-box intersection algorithm will intersect a ray with axis aligned bounding volumes[85] (i.e. the voxel

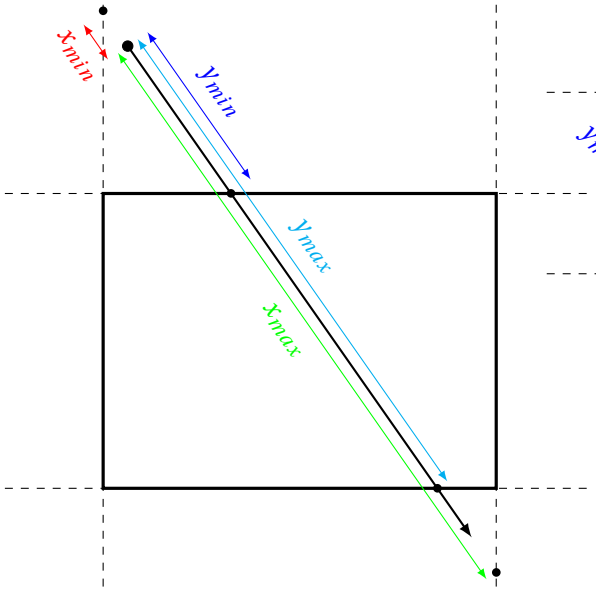



*Figure 3.7: Ray-box intersection in 2D, showing $x_{min}$, $x_{max}$, $y_{min}$, and $y_{max}$, with the ray origin outside the box and the ray intersecting the box twice.*

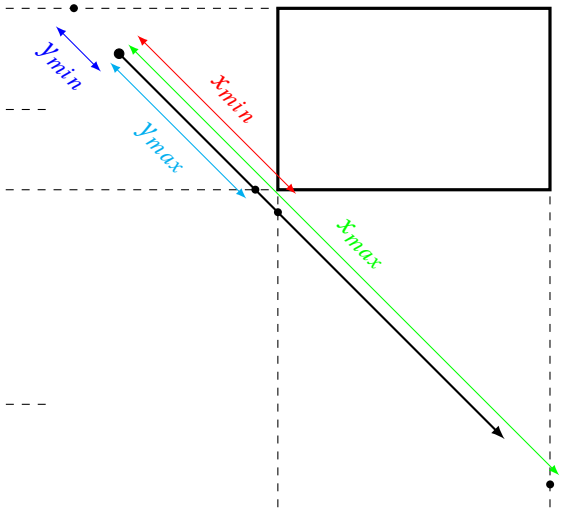



*Figure 3.8: Ray-box intersection in 2D, showing $x_{min}$, $x_{max}$, $y_{min}$, and $y_{max}$, with the ray origin outside the box but not intersecting the box.*

space), identifying a point within the voxel space that can be used as input to the previous traversal algorithm.

The ray-box algorithm is similar to the traversal algorithm. Distances from the ray origin to cross the box boundaries are computed in an effort to determine whether the ray intersects the box and the point of intersection if it exists. An example of a ray-box intersection that intersects the box is shown in Figure 3.7, and one that does not intersect the box is shown in Figure 3.8. The distance from the ray origin to the minimum and maximum box boundary in each dimension is computed. Given $min$ and $max$ vectors defining the minimum and maximum corners of the box, and a ray $\vec{r} + \vec{s}t$, where $\vec{r}$ is the ray origin, $\vec{s}$ is the ray direction, and $0 \leq t \leq \infty$, the distances to the minimum and maximum box boundaries are given by

$$i_{min} = \frac{min_i - r_i}{s_i} \qquad i_{max} = \frac{max_i - r_i}{s_i} \tag{3.4}$$

where $i$ indicates the $x$, $y$, or $z$ direction, and $s_i$ is assumed to be positive. If $s_i$ is negative, the

distances to the minimum and maximum box boundaries are given by

$$i_{min} = \frac{max_i - r_i}{s_i} \qquad i_{max} = \frac{min_i - r_i}{s_i} \tag{3.5}$$

Values are calculated in each dimension, logic is used to determine whether the ray intersects the box, and if so, the intersections are returned. The basic algorithm loop is shown in Algorithm 2.

**Algorithm 2** Ray-Box Intersection (adapted from Williams [85])

**Input:** `min` and `max`, 3-element vectors defining the minimum and maximum corners of the box, and `r`, a ray with a defined origin and direction
**Output:** `tmin` and `tmax`, the distance along ray `r` from its origin to the points of intersection with the box

**if** $\big((x_{min} > y_{max}) \quad \| \quad (y_{min} > x_{max})\big)$ **then**
    **return** `Null`
**if** $(y_{min} > x_{min})$ **then**
    $t_{min} = y_{min}$
**else**
    $t_{min} = x_{min}$
**if** $(y_{max} < x_{max})$ **then**
    $t_{max} = y_{max}$
**else**
    $t_{max} = x_{max}$
**if** $\big((t_{min} > z_{max}) \quad \| \quad (z_{min} > t_{max})\big)$ **then**
    **return** `Null`
**if** $(z_{min} > t_{min})$ **then**
    $t_{min} = z_{min}$
**if** $(z_{max} < t_{max})$ **then**
    $t_{max} = z_{max}$

**return** $t_{min}, t_{max}$

The combined ray-box intersection and voxel traversal algorithms provide an efficient method for identifying free voxels by incorporating the ray origins.

### 3.1.4 Voxel Classification

At this point, the number of points contained within and the number of rays passing through each voxel have been tallied, and the next step is to determine what properties can be derived with this information.

In a study performed by the Army Research Laboratory, Haas [76] attempted to apply a voxel-based ray-tracing approach to change detection between LIDAR scans. An occlusion metric was used to determine the probability $P_{occlusion}$ that a given voxel would produce a LIDAR return, given by:

$$P_{occlusion} = \frac{n_{points}}{n_{points} + n_{rays}} \tag{3.6}$$

where $n_{points}$ is the number of LIDAR return points in a voxel, and $n_{rays}$ is the number of rays that intersected the voxel without producing a return point. The results proved to be inconclusive due to errors in the registration of scans and the study was considered a failure. However, it was noted that the ray-tracing approach appeared to be "a useful counterpoint to the location of the return in defining an evidence-based measure of voxel content." [76]

Yapo *et al.*[75] took the ray-tracing concept further by classifying voxels into three distinct categories: occupied, free, and hidden. In this case, the hidden state was used to define a voxel that could not be reached by a LIDAR instrument, such as those inside a building or underground. Voxels were categorized by ray-tracing, where a ray originating at the sensor location was followed through the voxel space, incrementing the occupied, free and hidden counters as appropriate. The free counter was incremented for voxels lying on the ray between the sensor position and the surface return point, the occupied counter was incremented in voxels where there was a surface return, and the hidden counter was incremented for those voxels lying on the ray beyond the surface return. The final classification of each voxel was then determined using the accumulated counters and a set of probabilistic return functions. The addition of a hidden state is important, as it solves the problem of only using an empty/full classification for voxels in which hidden/unsampled voxels would be classified as empty in addition to truly empty voxels.

Hagstrom *et al.*[78] extended the occlusion probability to compute the transmission of a voxel, given by:

$$\tau\left(V_{x,y,z}\right) = \frac{\sum I_i \text{transmitted through} V_{x,y,z}}{\sum I_i \text{inside} V_{x,y,z} + \sum I_i \text{transmitted through} V_{x,y,z}} \tag{3.7}$$

where $V_{x,y,z}$ is the voxel at position $(x, y, z)$, and $I_i$ is the intensity of the return $i$ from the set of all returns collected. This method relies on the ratio of energy that passes through a voxel to the energy returned from a voxel, where energy is computed from the LIDAR return intensities. Passive image techniques do not have the ability to provide return intensities in the way that LIDAR does, which limits the use of this method of transmission computation to LIDAR derived point clouds.

All of these studies show how the knowledge of the system position can be used to derive additional information, and it is important to find a method for classifying the voxels. In this case, voxels will be classified using a ternary system to define occupied, free, and unsampled voxels. In general, voxels that contain reconstructed 3D points should be classified as occupied voxels, and voxels that lie on the ray between the camera and the point should be classified as free voxels. Those voxels that do not contain points or rays have yet to be identified as occupied or free and remain unsampled. A 2D visualization of occupied, free, and unsampled voxels constructed from a simple scene is shown in Figure 3.9.

This ternary classification system is conceptually simple, however it does not take into account that a voxel is not infinitesimally small. It is possible that the finite space of a voxel will contain multiple interactions. For example, multiple points could be contained in the voxel, multiple rays could pass through the voxel, or a combination of both points and passing rays could be contained in the voxel. This can be equated to an occlusion, or conversely transmission, property of a voxel. A voxel that contains only points can be considered fully occluded, while a voxel that contains only rays can be considered fully transmissive. A voxel that contains many points and only a few passing rays could be considered more occluding, or conversely more transmissive, than a voxel that contains only a few points and many passing rays.

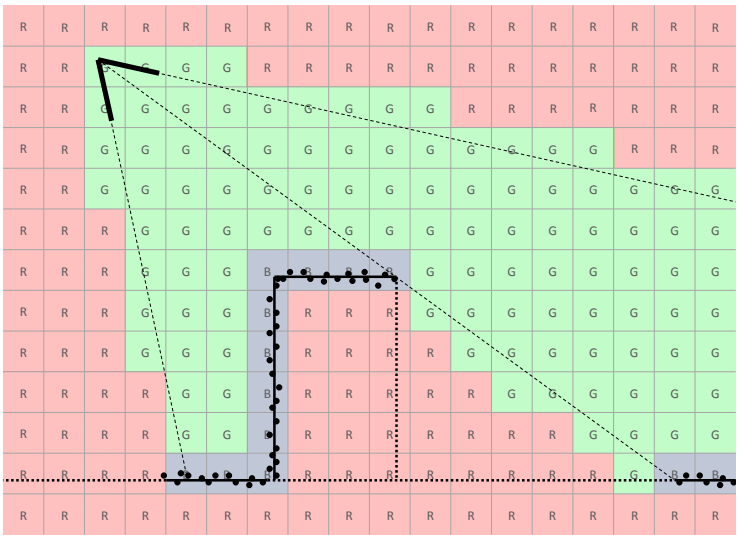


*Figure 3.9: A 2D visualization of occupied, free, and unsampled voxels. Voxels that contain black points are occupied and denoted with blue (B). The camera is located in the top left and free voxels that rays pass through are denoted with green (G); three rays, denoted with dashed lines, are depicted for reference. Unsampled voxels are denoted with red (R). The scene depicts a rectangular surface, such as a building, on a ground surface; where surfaces visible in the image are denoted with a solid line, and non-visible surfaces with a dotted line.*

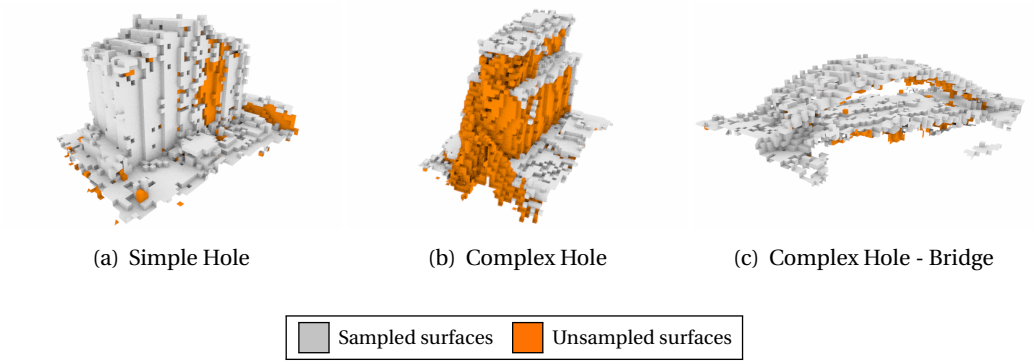
Each voxel has a corresponding number of points and number of rays associated with it that can be used to define a voxel property. A traditional transmission property is not achievable in this case, so a probability of free space $P_{free}$ will be used, where $P_{free}$ is given by:

$$P_{free} = \frac{n_{rays}}{n_{points} + n_{rays}} \tag{3.8}$$

where $n_{points}$ is the number of points in a voxel, and $n_{rays}$ is the number of rays that intersected the voxel. The probability of free space is simply $1 - P_{occlusion}$, where $P_{occlusion}$ is the probability of occlusion given by:

$$P_{occlusion} = \frac{n_{points}}{n_{points} + n_{rays}} \tag{3.9}$$

Using this approach, all voxels that contain points and/or rays have a probability associated with them. Those voxels that are unsampled will be given a probability of -1 to make them easily identifiable in subsequent analysis.




(a) Simple Hole (b) Complex Hole (c) Complex Hole - Bridge

*Figure 3.10: Voxelized models, including unsampled voxels, of the regions of interest shown in Figures 3.1, 3.2, and 3.3, representing both simple and complex holes in the point cloud. The unsampled voxels appear to reasonably define the portions of the buildings that were missing, and the free space between the bridge trusses and road below is correctly identified.*

An advantage to using this type of approach is that the probability of free space can be computed from a single sample in a voxel, and additional samples in a voxel can be used to refine the associated probability. This representation also allows for voxels to be represented with continuous values, rather than only using the occupied/free labels. For simplicity, a threshold will be implemented such that a probability of free space greater than the specified threshold will be considered a free voxel, and a probability less than the threshold will be considered an occupied voxel. Further investigation into the continuous nature of $P_{free}$ warrants future work.

The voxel models shown in Figure 3.4 were updated to reflect the addition of the unsampled voxel state and the results are shown in Figure 3.10. The models were created in Blender, where free voxels are transparent and the unsampled voxels are indicated with orange; shading was added for visualization purposes. As expected, the unsampled voxels fill the space inside the buildings, however, they also appear to reasonably define the portions of the walls that were missing in the point cloud. While there are some unsampled voxels surrounding the bridge trusses, the majority of the region between the trusses and the road below have been identified as free space, something that may be difficult to ascertain from the previous models or point cloud data alone without knowledge of the scene. These models show that the additional information provided by the unsampled voxels can be used to identify the voids in the point cloud.

## 3.2 Void Identification

Once the voxel space has been created from the point cloud data and the ray tracing is complete, the voxel space can be used to extract information about both the structure and voids present in the point cloud. Voids in the point cloud and corresponding voxel space are manifested as unsampled voxels. This was shown theoretically in Figure 3.9, where the right side of the rectangular shape was not reconstructed and thus resulted in a region of unsampled voxels, as well as practically in the models shown in Figure 3.10.

While it may seem that the objective should be to create a voxel space that is free of unsampled voxels, this will never be the case. Due to the nature of surface reconstruction, it is impossible to obtain a model free of unsampled voxels, as there will always be unsampled voxels under the model (e.g. in the interior areas of buildings and below the ground plane) in areas that are impossible to image. Rather, the objective is to obtain a voxel space such that free voxels and unsampled voxels are separated by occupied voxels.

### 3.2.1 Voxel Boundaries

The most pertinent information in the voxel space lies in the boundaries between the voxel types. Given the ternary voxel classification (free, occupied, and unsampled), there are three different types of boundaries to consider: the free-occupied boundary, the free-unsampled boundary, and the occupied-unsampled boundary. An illustration of the boundaries in the voxel space is shown in Figure 3.11.

A voxel face that is shared by a free voxel and an occupied voxel (i.e. the free-occupied boundary) is indicative of known structure in the scene. Similarly, a voxel face that is shared by a free voxel and an unsampled voxel (i.e. the free-unsampled boundary) is indicative of a void or missing structure in the scene. Note that the occupied-unsampled boundary is theoretically complementary to the free-occupied boundary, where the free-occupied boundary will be on the side of the surface model oriented in the general direction of the cameras, and the occupied-unsampled boundary will be on the side of the surface model oriented away from the general direction of the cameras. Inclusion of this third boundary does not gain any useful information and thus only two boundaries, the free-occupied boundary and the free-unsampled boundary, will be considered.

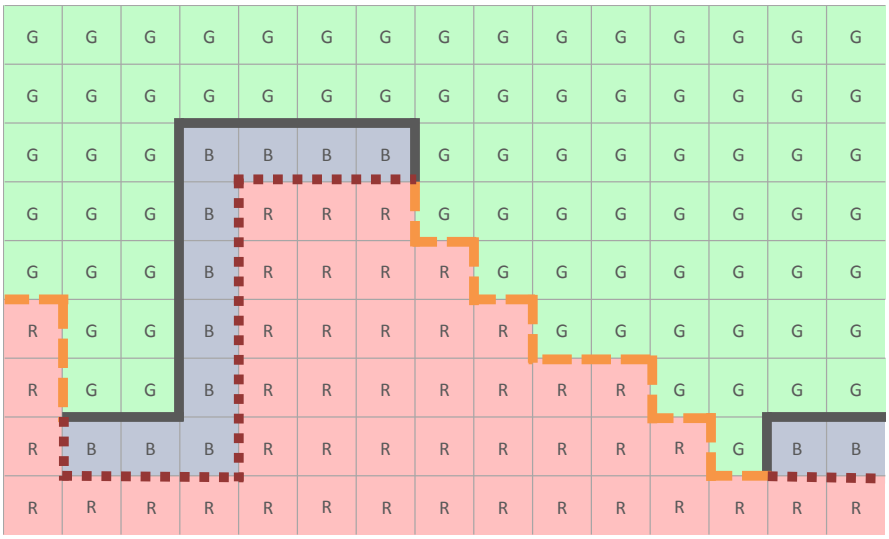



*Figure 3.11: A 2D visualization of the voxel space with the three types of voxel boundaries marked. The free-occupied boundary is shown as a solid gray line, the free-unsampled boundary is shown as a dashed orange line, and the occupied-unsampled boundary is shown as a dotted red line. The voxel space itself is a subset of the voxel space depicted in Fig. 3.9.*

If the free-occupied boundary represents known surfaces, and the free-unsampled boundary represents unsampled surfaces, then the previous objective, to have occupied voxels between the free and unsampled voxels, can be restated such that the objective is to obtain a model with minimal unsampled surfaces.

At this point, it is important to consider whether it is reasonable to assume that the unsampled surfaces represent the most likely location of a surface that was not reconstructed in the SfM process. For this to be the case, the ray tracing must have carved out the free space such that the knowledge of unsampled voxels is complete. While this may be a reasonable assumption in certain cases, it is not inherently true for voxel spaces derived from SfM point clouds, particularly in areas that lacked image coverage. The trivial example of a 2D voxel space derived from a single image, shown in Figure 3.9, illustrates this, where the rays cast from the camera are blocked by the building structure, creating a shadow effect. This same effect can be seen in the voxels spaces from image-derived point clouds where a single face of a building reconstructed from multiple views, as shown in Figure 3.12. This indicates that it is not reasonable to assume that an unsampled surface is a likely location for a surface that was not reconstructed.

The faces of the voxels that lie on the free-unsampled boundary define the unsampled sur-

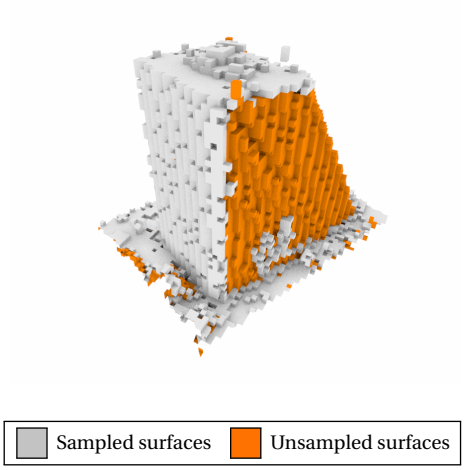



*Figure 3.12: Shadow cast by the building in the unsampled voxels as a result of ray-tracing in the voxel space.*

faces (voids) in the voxel space, and will be the basis for subsequent analysis. Using the voxel faces allows for a distinction to be made between the vertical and horizontal surface boundaries. In addition, each unsampled voxel face can be associated with a normal, such that the normal of the unsampled voxel face is oriented toward the neighboring free voxel. The unsampled face normals will be exploited in methods forthcoming.

### 3.2.2 Distinguishing the Type of Void

Up to this point, there has been no distinction made regarding the type of void in the point cloud. The focus of this work is to identify potential image locations such that inclusion of inclusion of imagery from these locations will result in fewer voids in the point cloud resulting from a lack of coverage. Lack of coverage or constant occlusion of an area in the imagery will result in voids in the point cloud, as was stated previously, and the distinction between the two causes is inconsequential because the inclusion of more imagery would be beneficial in both cases.

However, voids that are a result of homogeneous or texturally difficult areas will likely not benefit from the inclusion of more imagery. A technique, such as the one presented by Saponaro *et al.*[42], to reconstruct textureless regions could be beneficial in such a case, but that is be-

yond the scope of this work. It is possible that a sufficient number of images with a sufficient baseline between images were included in the reconstruction and therefore including more images of the area will not improve the reconstruction. This makes it important to distinguish voids that were a result of texturally difficult regions from other voids.

The voxel space does not have enough information to distinguish the type of void, however the original imagery used to reconstruct the point cloud is available and can be leveraged to identify homogeneous areas. An unsampled voxel face can be reprojected to the image frame, and the local image texture can be analyzed to provide additional information about the cause of the void.

#### Visibility of Unsampled Voxels

The unsampled voxels do not contain any points from the point cloud, and thus there is no visibility information associated with them. In order to accurately analyze the image texture, the camera frames in which the unsampled voxel would have appeared need to be determined. A line-of-sight analysis, similar to the one that was used to create the voxel space, can be used to determine visibility of the unsampled voxel surfaces.

The concept here is simple: if the unsampled voxel falls within the camera field of view, and all of the voxels along the ray between the camera center and the unsampled voxel face are free, then it is assumed the unsampled voxel face was imaged by the camera. If one or more occupied voxels are present along the ray, it is assumed that the camera was unable to image the unsampled voxel face because a surface exists between the camera and the unsampled voxel face that would obstruct the unsampled voxel face from the camera's view. As it has yet to be determined whether the unsampled voxels in the voxel space contain surfaces or not, it is assumed that unsampled voxels are opaque. Thus, the presence of one or more unsampled voxels along the ray also results in an obstructed camera view. For the purposes of this analysis, it is assumed that the content of the voxels is homogeneous, partial transmission of the voxels is ignored and voxels are assumed to be either fully opaque (occupied) or fully transmissive (free) based on the threshold chosen previously. An illustration of the ray-tracing for visibility purposes is shown in Figure 3.13

In addition to the requirement that the line-of-sight between the unsampled voxel face and the camera be free in the voxel space, it is also important to consider whether the ray would

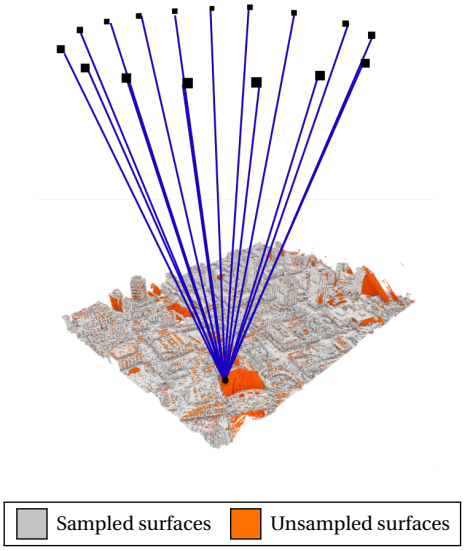
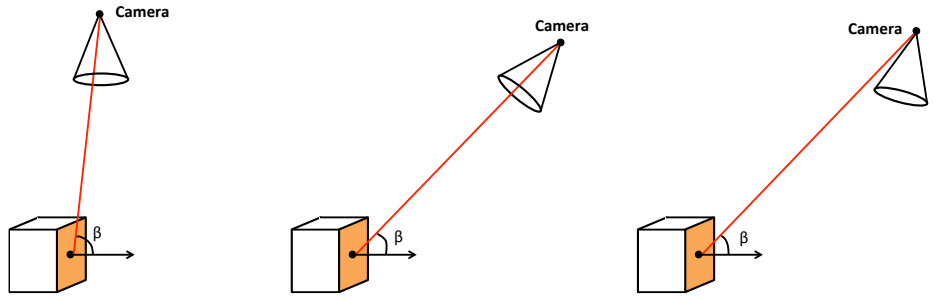



*Figure 3.13: Determining the visibility of the unsampled voxel faces using a line-of-sight analysis. For each unsampled voxel face, a ray is traced back to each camera used to create the point cloud; if there are no occupied or unsampled voxels on the ray, the ray is within the camera field-of-view, and the angle of incidence onto the unsampled surface is not a grazing angle, then it is assumed that the voxel was visible in the frame.*

intersect the cameras focal plane array, as a point outside of the field-of-view of the camera will not be imaged. This means that the pointing angle and field-of-view of the camera need to be considered. A central circular field-of-view was selected for simplicity, such that it did not include areas outside of the focal plane array. Selection of a narrow circular region will limit the effects of geometric distortion in the imagery, but more importantly the implementation is computationally efficient. The circular nature makes it possible to implement this with a dot product to determine the angle between the camera axis and the ray of interest.

Another consideration here is the angle between the unsampled voxel face and the incident camera ray. The angle of incidence can be used when determining whether a camera had potential to image a voxel face, and can indicate instances where the angle is less than favorable for imaging and reconstruction, such as grazing angles. Consider a nadir looking camera, and a missing voxel on the side of a building. In this case, the free-unsampled face of interest is likely oriented such that its surface normal is perpendicular to the angle of incidence from the camera. While the line-of-sight between the camera and the unsampled voxel space may be clear and within the cameras field-of-view, it is possible that the angle of incidence is a grazing angle. Regions that are imaged at grazing angles are not well reconstructed, and therefore visibility of a camera for a particular unsampled voxel face will not be considered if the incident angle is greater than a specified threshold angle.

To summarize, the visibility requirements for an unsampled voxel face are as follows: the ray defined by the unsampled voxel face center and camera center must be within the field-of-view angle of tolerance for a given camera system, and the incident angle between the unsampled voxel surface normal and ray must not be a grazing angle; if both of the previous conditions are satisfied, the ray-tracing algorithm is used to determine whether the line-of-sight is clear. If the path is not clear, the unsampled voxel face is outside the specified circular view of the camera system, or the incident ray is at a grazing angle, it is assumed that the point was not seen by that particular camera. Three examples of unsampled voxel face visibility are shown in Figure 3.14: (a) a grazing incident angle, where $\beta$ is greater than the specified tolerance, (b) a visible unsampled voxel face within the camera field-of-view, where the incident angle $\beta$ is within the tolerance, and (c) an unsampled voxel face that lies outside the camera field-of-view.




(a) Grazing Incident Angle (b) Visible Unsampled Face (c) Outside Camera FOV

*Figure 3.14: There are three criteria used to establish the visibility of an unsampled voxel face from a specified camera view: the incident angle of the ray onto the unsampled surface, the camera field-of-view, and ray-tracing within the voxel space to ensure that only free voxels lie on the ray between the camera and unsampled voxel face. The illustrations here show: (a) a grazing incident angle, (b) a visible unsampled voxel face where the unsampled voxel face is within the camera field-of-view (FOV) and the incident angle is not grazing, and (c) an unsampled voxel face that is beyond the camera field-of-view.*

### Image Texture Analysis

Now that camera visibility for each voxel face has been obtained, the camera projection matrices can be used to get an image location for a particular voxel face. The reprojection equation is given by:

$$\vec{x}_{cam} = \mathbf{P}\vec{X}_{world} \tag{3.10}$$

where $\vec{x}_{cam}$ is the homogeneous coordinate in camera space, $\mathbf{P}$ is the camera projection matrix, and $\vec{X}_{world}$ is the homogeneous coordinate in world space. Once the image coordinate of the unsampled voxel space has been obtained, the next step is to analyze the local image texture. The local image texture can provide context to the unsampled voxels in a way that the voxel space cannot. Figure 3.15 shows two highlighted regions that feature a number of unsampled voxels, and images of the area are shown for context. In the highlighted region in the river, the image does not feature a lot of texture. The region on the eastern portion of the image however does feature some texture. It is important to be able to distinguish between the two regions, so as not to heavily influence the future image location identification with textureless regions that

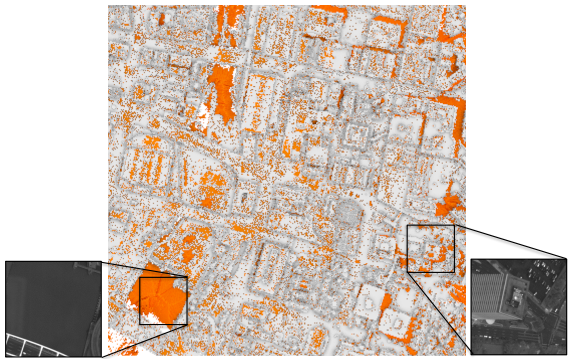


*Figure 3.15: Two regions of the voxel space featuring a number of unsampled voxels are highlighted, and the corresponding portions of imagery are shown for context. The river region exhibits little texture in the imagery, while the other region exhibits considerably more interesting features, suggesting that perhaps this is a void due to lack of coverage.*

will not reconstruct.

A statistical approach, in which texture is seen as a quantitative measure of the arrangement of image intensities, is the most common and widely used approach to texture analysis, particularly regarding natural scenes. One such statistical method is the spatial gray level co-occurrence texture features, developed by Haralick [86], which assumes that the textural information in the image is contained in the overall spatial relationships between the gray tones. Haralick suggested a total of 28 textural features that are functions of gray level and angle, including the angular second moment feature (a measure of homogeneity), the contrast feature (a measure of local variation) and the correlation feature (a measure of gray-tone linear dependencies). A downside of the gray level co-occurrence features is that they do not necessarily correspond to the visual perception of the image. While the Haralick features have been used successfully over a variety of applications on both small scale and large scale imagery, they may be excessive in this application where the objective is not to distinguish between different types of texture but rather only to determine whether there is sufficient texture to identify and reliably match image features. It is likely that a much simpler metric can be used in this case, as the only concern is if there is enough texture to identify image features and reliably match them between frames.

A 3D point cloud, reconstructed from 22 images, served as the basis for a preliminary texture study to determine an appropriate texture metric. The objective was to extract regions of

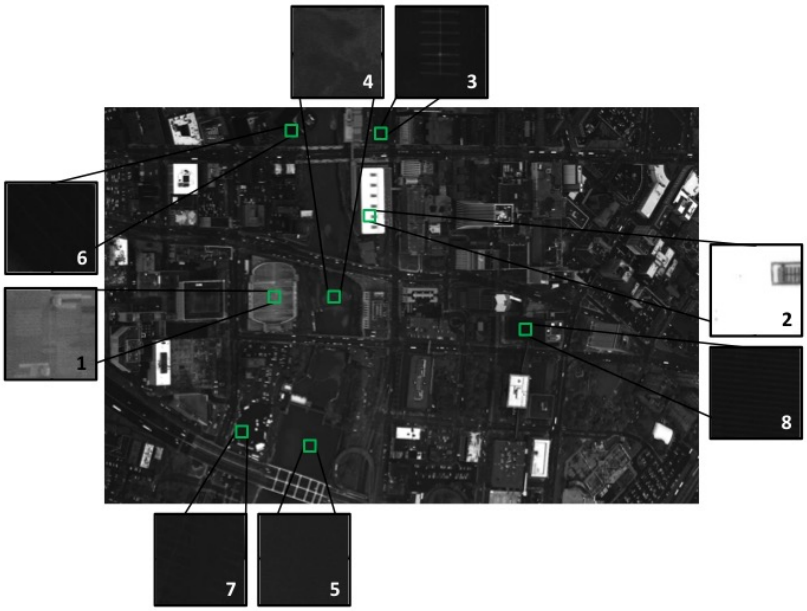



*Figure 3.16: Original WAMI Image of downtown Rochester, indicating 8 different regions of interest that are shown in the smaller images and indicated in green on the original. The regions of interest were used to analyze image texture, and study the relationship between various image statistics that represent texture and the likelihood of reconstruction. Descriptions of the regions and corresponding area of the reconstruction are shown in Table 3.1.*

the imagery that consistently reconstructed well, in addition to regions of imagery that consistently reconstructed poorly, and investigate local statistical image metrics that could be used to identify the regions that did not reconstruct. Eight regions of interest (ROI) were selected, four that were reconstructed well in the point cloud (ROI-1, ROI-2, ROI-3, ROI-4), and four that consistently lacked reconstruction points (ROI-5, ROI-6, ROI-7, ROI-8). The ROIs are shown in Figure 3.18, with an original WAMI image of the downtown area shown for context. The 3D reconstruction point cloud is shown in Figure 3.17, where the blue background was used to be able to differentiate areas that lack points. Descriptions of the regions as well as visual descriptions of the reconstruction are shown Table 3.1. And finally, the images and point cloud regions are shown side by side in Figure 3.18.

Selection of the ROIs in each of the 22 images was performed by manually by selecting 100x100 pixel portions of the imagery. As this process was done by hand, it was not precise

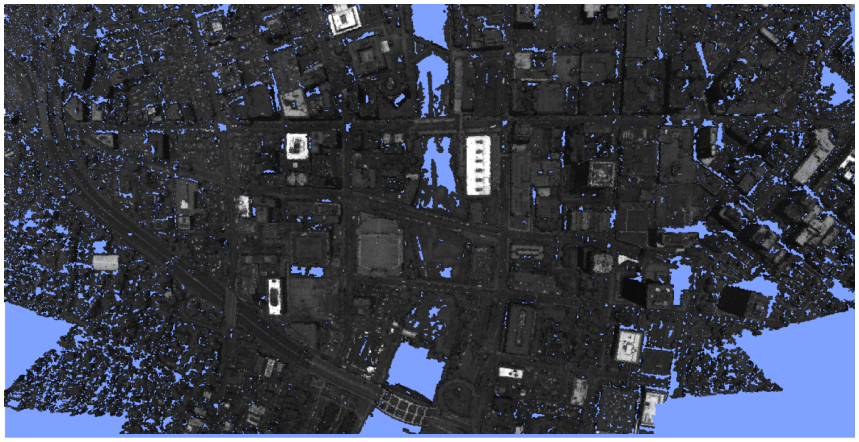


*Figure 3.17: Nadir view of the WAMI point cloud, provided for context in the texture analysis. The corresponding image and regions of interest are shown in Figure 3.16. The blue background was used to highlight the holes in the point cloud so that they would be easily distinguished from the points themselves.*

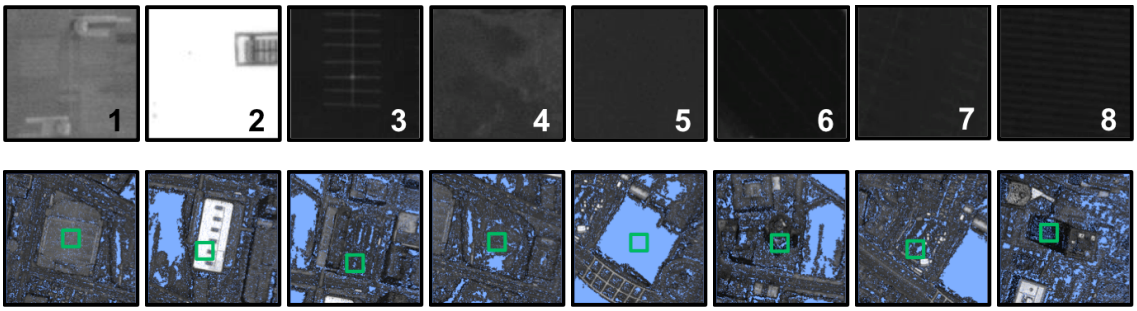



*Figure 3.18: Eight regions of interest for texture analysis in the WAMI imagery. Descriptions of the image region and quality of the point cloud reconstruction are shown in Table 3.1. Top: 100x100 pixel cropped region of WAMI image. Bottom: Corresponding region of the point cloud, where the image area has been highlighted in green.*

*Table 3.1: Names and visual descriptions of the point cloud reconstruction for the eight regions of interest in the texture analysis. The original image context is shown in Figure 3.16, and the corresponding cropped WAMI image sections and respective sections of the point cloud are shown in Figure 3.18.*

| **Number** | **Name** | **Description** |
|---|---|---|
| 1 | Blue Cross Arena Roof | Good reconstruction, dense, minor holes in central portion |
| 2 | Convention Center Roof | Very good reconstruction, dense |
| 3 | Parking Structure North of Convention Center | Good reconstruction around parking lines, holes in driving lanes |
| 4 | Genesee River East of BCA | Good reconstruction in the central portion, some minor holes, larger holes closer to the bridges |
| 5 | Genesee River North of 490 Bridge | No points in point cloud, did not reconstruct |
| 6 | Side of the Former Changing Scenes Restaurant | Poor reconstruction, a few points along the horizontal lines on the sides of the building, but otherwise sides do not have points |
| 7 | Trucking Parking Lot Northwest of 490 Bridge | Sparse reconstruction, points congregate around parking lines, large holes in driving lanes |
| 8 | Side of Xerox Building | Three out of four sides contain points, holes present |

and though care was taken to extract the same general area for each ROI across the images, the perspective distortions and rotation of the imagery resulted in slight differences in the selected regions. It should be noted that different sides were used for the two buildings (ROI-6 and ROI-8), based on which was visible in the imagery, and only 14 of the 22 images contained the Xerox building (ROI-8).

The objective of this study was to determine if a statistical image texture metric could be used to accurately estimate whether a region would generate features that could be used for reconstruction. There are two steps in the 3D reconstruction workflow that employ feature detection and matching: the initial identification of image-to-image correspondences for use in triangulation, and the dense stereo matching. In the 3D workflow used here, initial correspondences are determined with SIFT, and dense stereo matching is performed by PMVS. Potential feature locations for SIFT are identified by looking for local extrema in the difference-of-Gaussian images across scale space, and corresponding descriptors are gradient-based representations of the local image area. PMVS features also use a difference-of-Gaussian approach to look for feature locations, in combination with a corner detection scheme. Because the initial feature detection employed by SIFT and the detection performed by PMVS use similar methods, the following will focus on the number of SIFT features.

For each ROI, SIFT features were computed using VLFeat [41]. In this implementation, the peak threshold parameter can be used to filter the peaks in the difference-of-Gaussian scale space that are too small in absolute value, where small peaks can be indicative of unstable features. Fewer features are obtained as the peak threshold increases. SIFT features were computed with peak threshold values of 0.0 and 0.5, and the number of features for each ROI was tallied. The average number of features across the set of 22 images, as well as the minimum and maximum number of features for each ROI are shown in Table 3.2. As expected based on the reconstruction, ROI 1-4 average a large number of features, while ROI 5-8 average fewer features overall, and the minimum and maximum are indicative of the spread of values. The trend is even more pronounced when the peak threshold is increased to 0.5. Note that ROI-3 (the parking structure) has at least one image that obviously has very few features, with a minimum of 10, but also has at least one image with a lot of features, with a maximum of 81. The average value is consistently high, even after the peak threshold is increased. It may seem that ROI-6 (the former Changing Scenes restaurant ), and ROI-7 (the trucking lot) have a high average number of features at 31.32 and 35.86 respectively at a peak threshold of 0.0. However, a peak

*Table 3.2: Number of SIFT features computed for each ROI using VLFeat [41] for a peak threshold of 0.0 and 0.5. The average value was computed across 22 images (except for ROI-8, for which there were only 14 images available), and the minimum and maximum values are indicative of the range. As expected, ROI 1-4 exhibit more features than ROI 5-8, and this trend is even more pronounced with an increased peak threshold.*

| | Threshold 0.0 | | | Threshold 0.5 | | |
|---|---|---|---|---|---|---|
| | avg. | min. | max. | avg. | min. | max. |
| 1. Blue Cross Arena | 48.32 | 36 | 58 | 28.23 | 20 | 37 |
| 2. Convention Center | 40.18 | 24 | 57 | 39.18 | 24 | 55 |
| 3. Parking Structure | 43.45 | 10 | 81 | 32.86 | 0 | 65 |
| 4. River - North | 45.5 | 33 | 56 | 24.77 | 14 | 37 |
| 5. River - South | 13.05 | 3 | 25 | 0 | 0 | 0 |
| 6. Rotating Building | 31.32 | 17 | 48 | 2.23 | 0 | 6 |
| 7. Trucking Lot | 35.86 | 15 | 55 | 1.05 | 0 | 6 |
| 8. Xerox Building | 7.29 | 3 | 16 | 0 | 0 | 0 |

threshold of 0.5 reduces the average number of features for ROI-6 and ROI-7 to 2.23 and 1.05 respectively. The trends exhibited here confirm what was seen in the reconstruction, validating the use of these particular image regions in the texture study.

Now that the number of features for each ROI has been established, a statistical texture metric that can be used to reliably determine whether a particular area will be reconstructed must be determined. Two metrics were chosen as possible candidates: an average local gradient and a local image variance. The average local gradient was chosen because of the use of gradient-based feature descriptors while the local variance was chosen for its relative ease of computation and intuitive relation to image texture.

The first metric to be explored is the average local gradient. The computational formula for the magnitude of an image gradient is given by:

$$\| \bigtriangledown f \| = \left( \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right)^{\frac{1}{2}} \tag{3.11}$$

where $\frac{\partial f}{\partial x}$ is the gradient in the $x$-direction, and $\frac{\partial f}{\partial y}$ is the gradient in the $y$-direction. The gradient was computed for each pixel in a given image, and the average of all the gradient values in the neighborhood, in this case the entire 100x100 image, was used as a measure of texture.

*Table 3.3: Average image gradient computed for each ROI at full resolution, half resolution, and quarter resolution. The reported average value is an average across 22 images (except for ROI-8, for which there were only 14 images available) of the average image gradient for each ROI, and the minimum and maximum values are indicative of the range. Note that the average gradient seems to increase for ROI 1-4 as the resolution decreases, but decrease for ROI 5-8.*

| | Full Resolution | | | Half Resolution | | | Quarter Resolution | | |
|---|---|---|---|---|---|---|---|---|---|
| | avg. | min. | max. | avg. | min. | max. | avg. | min. | max. |
| 1. Blue Cross Arena | 2.00 | 1.76 | 2.30 | 1.95 | 1.72 | 2.10 | 2.24 | 1.82 | 2.53 |
| 2. Convention Center | 4.94 | 3.48 | 6.22 | 6.99 | 5.18 | 9.16 | 10.87 | 7.41 | 15.33 |
| 3. Parking Structure | 1.72 | 1.36 | 1.98 | 2.06 | 1.38 | 2.41 | 1.90 | 0.98 | 2.46 |
| 4. River - North | 1.45 | 1.09 | 2.21 | 1.41 | 1.25 | 1.74 | 1.78 | 1.52 | 2.29 |
| 5. River - South | 0.77 | 0.56 | 1.21 | 0.60 | 0.55 | 0.68 | 0.42 | 0.39 | 0.46 |
| 6. Rotating Building | 0.94 | 0.71 | 1.24 | 0.96 | 0.84 | 1.18 | 0.93 | 0.74 | 1.31 |
| 7. Trucking Lot | 1.01 | 0.77 | 1.35 | 0.95 | 0.88 | 1.07 | 0.82 | 0.55 | 0.96 |
| 8. Xerox Building | 1.47 | 0.96 | 1.85 | 1.45 | 0.76 | 2.20 | 0.69 | 0.59 | 0.82 |

The average gradient results computed for the 100x100 region are shown in Table 3.3, indicated as full resolution. The initial trends at full resolution did not correlate to the number of features as expected. When computing feature locations, SIFT employs an image pyramid to achieve scale invariance, and the effect of this pyramid could potentially account for the unexpected behavior from the average gradient, which in this case was only computed for a single scale. The gradient has some response at the highest resolution, but that response may start to grow as the resolution decreases, and it is possible that the average gradient value could change drastically at various scales. To simulate the effect of the image pyramid, the average gradient values were also computed for half resolution (50x50), and quarter resolution (25x25), where the degraded resolutions were computed by downsampling the original image area; the average gradient results from the downsampled images are also shown in Table 3.3.

Looking at the results, the average local gradients for ROI 1-4 are higher than those for ROI 5-7, but ROI-8 would fall into the higher category at both the full and half resolutions. At the various resolution levels, there is no distinct trend for the average local gradient; in some cases the gradient increases as the resolution decreases, while in others the gradient decreases as the resolution decreases. At quarter resolution, the trend for the average local gradient correlates well with the number of features expected in those regions, but this pattern is not consistent

across all of the resolution levels. Interpreting from these results, there is no distinct separation between the regions that generate a high number of features, and those that generate a low number of features. A downside to the use of the average local gradient is the lack of intuition at what the value of the average local image gradient represents.

The average local gradient was chosen as a possible candidate for a texture metric because many of the feature detectors take advantage of image gradients. While SIFT is a gradient-based feature descriptor, the detection of possible feature locations is performed by looking for extrema in difference-of-Gaussians. SIFT also employs an image pyramid such that it is possible to search for features at a variety of scales. In this case the average local gradient is computed between neighboring pixels, and the response of the gradient can change drastically at different image scales or resolutions. The unpredictability of the local gradient response at varying image scales, as well as the lack of separability between the reconstructed and non-reconstructed image regions makes the local gradient less than ideal as a possible texture metric for determining failed reconstruction as a result of homogeneous regions.

Local image variance provides another possible texture metric. The computational formula for variance is given by:

$$\begin{aligned} \sigma_X^2 &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\ &= \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2 \end{aligned} \tag{3.12}$$

where $\mathbb{E}$ is the expected value; it can be thought of as the mean of the square minus the square of the mean. Variance can be implemented by blurring the image with box filters, where a normalized box filter is used to represent the expected value of the neighborhood of interest. Note that the unit of variance in this case would be squared gray-level, whereas the unit of standard deviation would be simply gray-level, as standard deviation is the square root of variance. The following results are given as standard deviations because that value is more intuitive. The standard deviation was computed for each ROI at full resolution (100x100), as well as half resolution (50x50), and quarter resolution (25x25), and the results are shown in Table 3.4.

The average standard deviations for ROI 1-4 are higher than those for 5-8, and there is a distinct separation between the reconstructed and non-reconstructed ROI at all of the resolutions. The standard deviation for ROI-2, the Convention Center, is significantly higher than the other

*Table 3.4: Standard deviation computed for each ROI at full resolution, half resolution, and quarter resolution. The reported average value is an average across 22 images (except for ROI-8, for which there were only 14 images available) of the standard deviation for each ROI, and the minimum and maximum values are indicative of the range. The standard deviation decreases as the resolution decreases, but there is distinct separability between regions that reconstructed well (ROI 1-4), and those that did not (ROI 5-8) at each resolution.*

| | Full Resolution | | | Half Resolution | | | Quarter Resolution | | |
|---|---|---|---|---|---|---|---|---|---|
| | avg. | min. | max. | avg. | min. | max. | avg. | min. | max. |
| Blue Cross Arena | 4.58 | 3.91 | 5.28 | 4.26 | 3.54 | 4.90 | 3.91 | 3.19 | 4.58 |
| Convention Center | 42.34 | 35.27 | 49.64 | 42.19 | 35.17 | 49.50 | 41.75 | 34.80 | 48.95 |
| Parking Structure | 3.74 | 2.83 | 4.78 | 3.47 | 2.38 | 4.56 | 2.90 | 1.89 | 3.98 |
| River - North | 3.69 | 3.09 | 4.32 | 3.48 | 2.93 | 4.14 | 3.34 | 2.79 | 4.01 |
| River - South | 1.11 | 0.81 | 1.44 | 0.88 | 0.71 | 1.09 | 0.73 | 0.56 | 0.96 |
| Rotating Building | 1.88 | 1.42 | 2.83 | 1.72 | 1.33 | 2.76 | 1.53 | 1.16 | 2.58 |
| Trucking Lot | 1.57 | 1.30 | 1.91 | 1.31 | 1.13 | 1.46 | 1.06 | 0.84 | 1.18 |
| Xerox Building | 2.08 | 1.45 | 2.70 | 1.79 | 1.12 | 2.46 | 1.01 | 0.77 | 1.32 |

regions due to the high contrast of the black and white on the rooftop. The minimum standard deviations for ROI-3 and ROI-4 are slightly lower than ROI-1, which may indicate that in some images there was not sufficient structure, but likely there was enough in the other images that it was not a problem. Similarly, the higher maximum standard deviations for ROI-6 and ROI-8 may indicate that at least one image had adequate texture, but that there were not a sufficient number of images with adequate texture for proper reconstruction. The local standard deviation has a higher response at the higher resolutions, decreasing as resolution decreases. However it appears to have a more uniform response across the image scales than the local average gradient.

This shows that some texture metric, in this case standard deviation, can be used as an indicator of whether an area will be rich with possible features to match, or deficient. As the standard deviation appears to show a better correlation than the average local gradient to the number of features that are generated for each of the regions of interest, it will serve as the metric to determine whether a region in the voxel space was not reconstructed due to a lack of texture in the imagery. Unsampled voxel faces that have an average standard deviation less than a specified threshold will be removed from consideration in subsequent analysis.

In this case, it is better to have a false alarm (indicating a region has adequate texture for reconstruction when in fact it does not), rather than a miss (indicating a region does not have adequate texture for reconstruction when in fact it does) given an objective to make the voxel space as complete as possible by minimizing the number of unsampled surfaces. It is important to note that this texture metric will be dependent on both the window size for which the standard deviation is calculated, as well as the threshold itself.

## 3.3 Future Image Location Identification

Now that voids that are a result of lack of coverage and occlusion have been identified, the next step is to determine a method for filling in these surfaces. Due to the nature of this type of surface reconstruction, it is impossible to obtain a model that is devoid of unsampled voxels, as there will always be unsampled voxels under the surface model. An ideal model would be free of unsampled surfaces, such that free voxels are separated from unsampled voxels by occupied voxels in all areas.

As was discussed in Sections 2.3 and 2.4, traditional surface reconstruction and hole filling algorithms will not work in most cases in the point cloud. It is believed that the inclusion of imagery in the reconstruction process that covers the void areas will result in a point cloud with fewer voids that are a result of lack of coverage in the imagery. Therefore it becomes important to identify potential camera locations that could provide optimal views of the void areas, particularly those that have not been seen in multiple views previously.

There are two ways to approach the problem of finding optimal imaging locations:

1. **The Backward Approach:** What is the field-of-view of each voxel? Given a voxel of interest, determine the solid view angle that is unobstructed by projecting out into the voxel space.

2. **The Forward Approach:** What voxels are visible from a specific point in space? Given a specific location and viewing angle, determine which voxels would be visible by tracing through the voxel space.

While both approaches are designed to address the same problem, where is the best view of the voxels of interest, they differ in geometric complexity. Starting with the backward approach, the

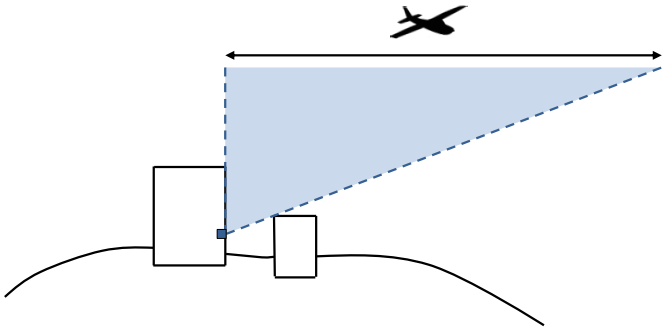
geometry of each are explored in the following sections in order to determine which method should be used to proceed.

### 3.3.1 The Backward Approach

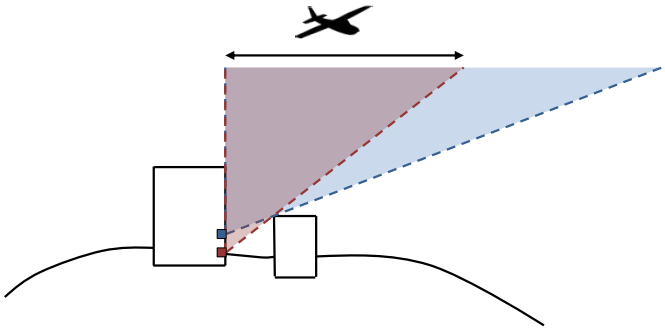
This approach may be the more intuitive of the two, and admittedly was the first one to be considered. For simplicity, consider the problem first in two dimensions. Given a voxel face of interest, rays can be cast out into the voxel space in such a way as to determine the field-of-view of the voxel face. If a ray meets an obstruction then that region is removed from the potential field-of-view. Because the content of unsampled voxels remains unknown, an obstruction in this case is defined to be either an occupied or unsampled voxel that intersects the ray. This process is repeated until the potential field-of-view of the voxel has been adequately sampled. In 2D, a completely unobstructed voxel face could potentially have a field-of-view of 180 degrees.

A 2D diagram of the backward approach is shown in Figure 3.19, where there is a progression starting with a single voxel of interest in (a) and moving to three voxels of interest in (c). Determining the optimal viewing location for each voxel face could potentially lead to hundreds of images, depending on the number of unsampled voxel faces. Depending on the proximity of the unsampled voxel faces, this could also lead to narrow baselines between images. It is more feasible to determine where the fields-of-view from multiple voxels overlap to predict a viewing location, while also keeping in mind the camera pointing angle and camera field-of-view. In Figure 3.19, as the number of voxels increases, the optimal field of view where there is overlap narrows, potentially converging to an optimal view solution. In the example given however, the optimal positions predicted by combining the three voxel fields-of-view will result in grazing views of the voxel faces, due to the fact that the three voxels are on parallel facing surfaces. While the ray tracing may indicate that the voxel face can be seen, grazing angles are not optimal for reconstruction, and severe grazing angles may even result in the surface not appearing in the image.

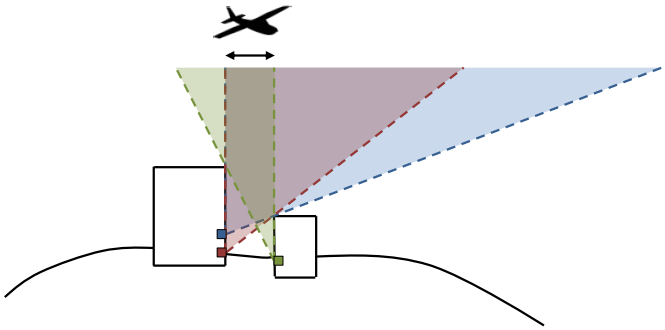
Just as the angle of incidence between the camera ray and the unsampled surface normal was used in determining the unsampled voxel visibility to eliminate grazing angles, it can also be used to mitigate against grazing angles when considering potential future image locations. There are two potential ways to take advantage of this information: (1) using a threshold to



(a) One Voxel



(b) Two Voxels



(c) Three Voxels

*Figure 3.19: A 2D diagram of the "backward approach," shown with one, two and three voxels in (a), (b), and (c) respectively. The voxel field-of-view is limited by the buildings in each case, and the potential viewing locations are shown with the arrow. As additional voxels are added, the field-of-view to be able to image all of the voxels narrows.*

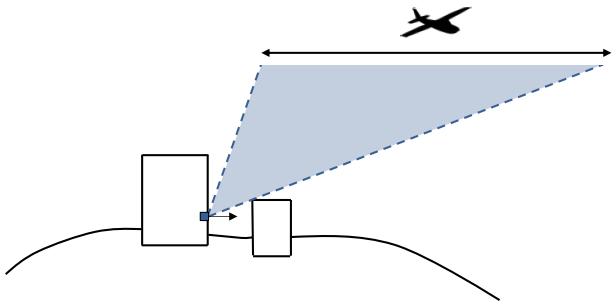
limit the angles allowed, or (2) using some sort of functional distribution to weight selection such that grazing angles are given a low probability for selection. The following approach will use the single threshold for simplicity. Figure 3.20 shows the impact that the addition of surface normals and an angular threshold would have on the previous scenario. In this case, it indicates that multiple images are required in order to adequately capture all of the missing voxels.

Using the previous figures, the backward approach seems to be relatively straightforward in 2D. However, the pointing direction of the camera and its field-of-view have been neglected here, and would contribute additional complexity. The biggest challenge in the backward problem however is the addition of the third dimension, which requires the projected cone from the voxel face to be a 3D solid angle. An illustration of the 3D case is shown in Figure 3.21. As has been stated previously, the solution space for the possible location is six-dimensional because the solution needs to consider not only the location of the imaging platform, but also the pointing direction. The difficulties are exacerbated in the backward approach because the solution lies in angle space.

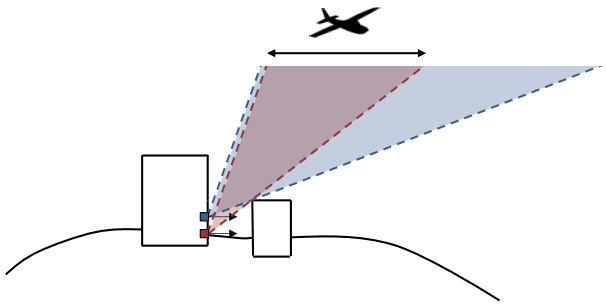
### 3.3.2 The Forward Approach

While the solution will still have six parameters, the forward approach is conceptually more manageable. The idea in the forward approach is that, given a location and pointing direction in addition to knowledge of the camera field-of-view, ray tracing can be used to determine which unsampled voxel faces are visible. A 2D diagram of this approach is shown in Figure 3.22, where the aircraft is shown at 3 different positions. Again, the surface normals of the unsampled voxel faces can be taken into account to mitigate grazing angles, such as those that would likely result from position 2 shown in Figure 3.22(b).

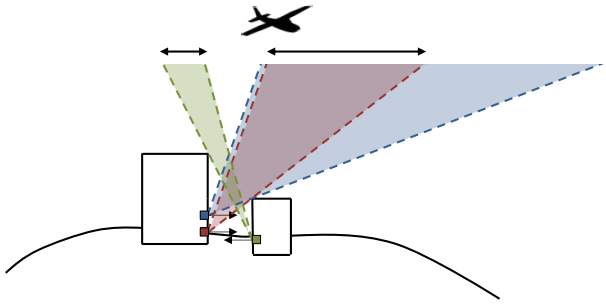
From a computational perspective, the forward approach is readily extensible to 3D by simply using a 3D location and viewing angle. An illustration is shown in Figure 3.23; note that the solution space here, shown with the blue '+' signs, is in 2D as only the $X$ and $Y$ positions of the aircraft are varying. The biggest challenge is that the solution space is large and the ray tracing is computationally expensive, but this is also true of the backward approach. Ultimately, the forward approach was selected for ease of implementation.



(a) One Voxel

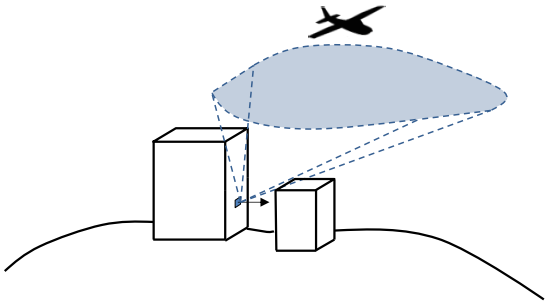


(b) Two Voxels

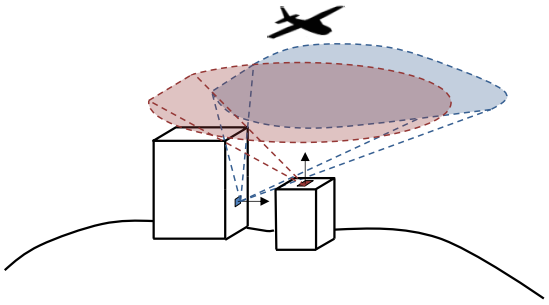


(c) Three Voxels

*Figure 3.20: A 2D diagram of the "backward approach," shown with one, two and three voxels in (a), (b), and (c) respectively. In this case however, the normals of the voxel faces are used to limit the potential field-of-view to avoid grazing angles, as they are not optimal for reconstruction purposes. The one and two voxel cases are similar to those in Figure 3.19, with just slightly narrower fields-of-view. Due to the geometry shown here though, the addition of the third voxel indicates that it will not be possible to obtain views of all three voxels and multiple images must be considered.*

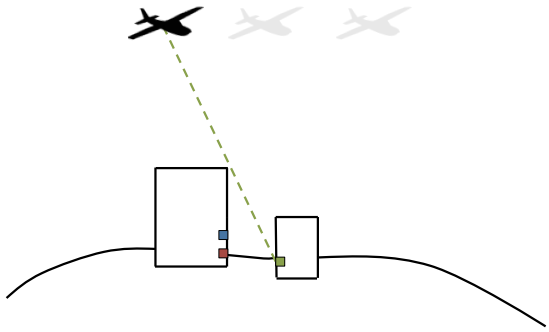


(a) One Voxel

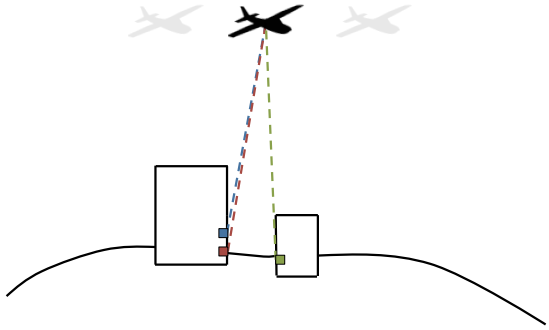


(b) Two Voxels

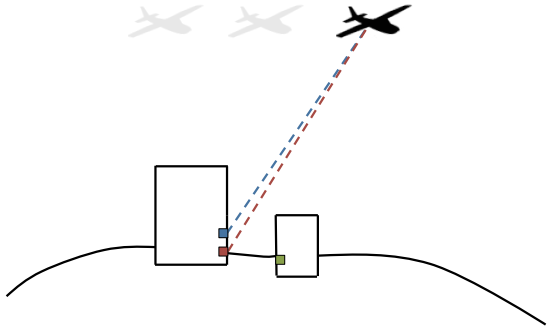
*Figure 3.21: A 3D diagram of the "backward approach," shown with one and two voxels in (a) and (b) respectively. The backward approach becomes much more complex in 3D, dealing with solid angles.*



(a) Position 1

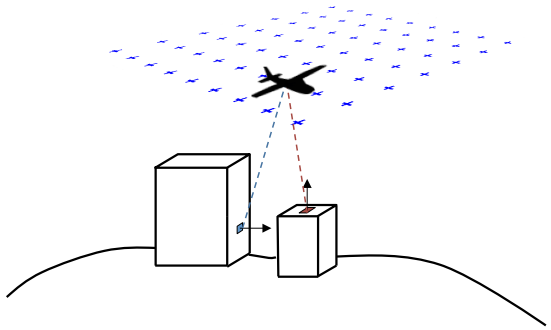


(b) Position 2

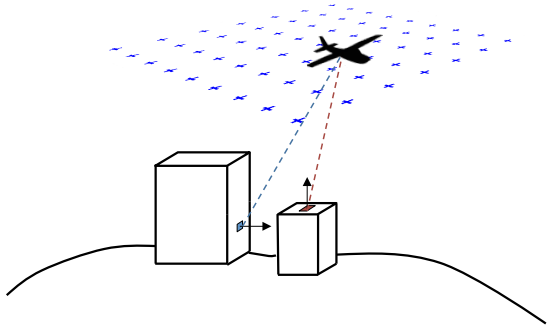


(c) Position 3

*Figure 3.22: A 2D diagram of the "forward approach," shown at three different positions in (a), (b), and (c).*



(a)



(b)

*Figure 3.23: A 3D diagram of the "forward approach" for two different positions. Note that the solution space shown here with the blue positions is only 2D, given that only the X and Y positions are varying.*

### 3.3.3 Cost Function

The next step is to determine a criteria that can be used to resolve optimal imaging locations, and this is where a cost function comes into play. Ultimately, the goal of this process is to use additional imagery in the 3D workflow, such that the resulting reconstruction contains more points and the subsequent voxel space contains fewer unsampled surfaces. It is impractical that the criteria of the cost function be based on the resulting reconstruction, as this would require imagery from all possible locations to have previously been acquired. Therefore the cost function must be based on some aspect of the current voxel space.

The most obvious choice is to determine the position and pointing angle at which the most unsampled voxel faces are visible, giving the best probability for improving the reconstruction. It should be noted that the image acquired from this location may be from a vastly different perspective than the ones previously used in the 3D reconstruction, and therefore there is no guarantee that use of a single image will improve the reconstruction based on the criteria here. The objective of this research is to find the location from which the most unsampled voxel faces are visible and more than just this single image may be required for improvements to the 3D reconstruction, presenting an area for future work.

The same method that was used to determine voxel visibility in Section 3.2.2 can be applied here. In order for an unsampled voxel face to be visible from a specific location, it must be within the camera field-of-view. Again, a circular field-of-view will be used for simplicity. Additionally, the line-of-sight from the potential camera location must be clear. The line-of-sight is considered clear if the voxels that lie on the ray between the unsampled voxel face and the potential camera location are all free; the presence of any occupied or unsampled voxel on the ray renders that unsampled voxel face not visible. Both of these criteria are either true or false.

The angle between the surface normal of the unsampled voxel face and the incident ray provides another criteria. Previously in determining visibility for the texture analysis, this angle was used to employ a threshold such that grazing angles could be eliminated from consideration. In this case, it could be used to weight the contribution of each unsampled voxel face based on probability of reconstruction.

Due to the iterative nature of the voxel traversal algorithm, a purely analytical cost function is improbable. As such, an algorithmic approach has been taken and is shown in algorithm 3.

**Algorithm 3** Algorithmic criteria for a given image location.

**Input:** Voxel map, and a camera with origin $\vec{r}$ and direction $\vec{s}$
**Output:** List of visible unsampled voxel faces

**for each** unsampled voxel face **do**
  Compute the ray $\vec{p}$, defined by the unsampled voxel face center and the camera origin
  **if** $\vec{p}$ is inside the camera FOV **then**
    **if** the line-of-sight along $\vec{p}$ is clear in the voxel map **then**
      Compute the angle of incidence, $\theta$, between $vecp$ and the surface normal
      Contribution of the unsampled voxel face is weighted with $w(\theta)$

The result of algorithm 3 is a list of unsampled voxel faces that are visible from the given location, along with the corresponding angle of incidence. The contribution of each unsampled voxel face will be weighted based on the angle of incidence, and the sum of the weights will be used to determine the best image location. A discussion of the weighting function can be found in Section 4.3.1.

### 3.3.4 Constraints

As was stated previously, there are six degrees of freedom when considering the camera location $(X, Y, Z)$ and pointing $(\omega, \phi, \kappa)$, and the solution space is large. Because of this, identifying optimal imaging locations is not a computationally tractable problem. To make the problem more manageable, three constraints to the camera location and pointing will be applied.

#### Circular Camera Field-of-View

The constrained circular field-of-view for the camera that was used to determine line-of-sight for the unsampled voxel faces (Section 3.2.2) will be used again here. The circular field of view is computationally efficient, because a dot product can be used to determine if an unsampled voxel face is within the field-of-view of the camera. This eliminates one degree of freedom because the camera pointing can be defined by a unit vector in the direction of the camera's principle axis. Because the circular field-of-view is designed to fall within the bounds of the focal plane array, any arbitrary choice of the image plane axes when designing flight plans will be able to see the unsampled voxel faces.

### Altitude

For the purposes of this research, it is assumed that the potential image locations are constrained to an airborne imaging platforms, and therefore potential altitudes can be bounded given that there is a limited range of flying altitudes for an aircraft. Beyond that, there are also numerous reasons to restrict the altitude to a single flying height. For flight planning purposes, it is traditional that an aircraft maintain altitude, rather than climbing and diving at random. Changes in altitude will affect the ground resolution of the imagery, which may or may not impact the 3D reconstruction.

Another consideration is the effect of altitude on the cost function. The criteria used to determine the best potential imaging locations is based upon maximizing the number of unsampled voxel faces that are visible from a specified location and pointing vector. Simple geometry dictates that the footprint of the imaging system on the ground is a function of the aircraft altitude, and a higher altitude will have a larger footprint than a lower altitude. Based on the "number of unsampled voxel faces visible" criteria, higher altitudes will always be favored due to the fact that their ground footprint is larger and thus they have the potential to see more of the unsampled voxel faces. Because of this, a constant altitude will be used to determine the optimal imaging locations, thereby eliminating one more degree of freedom.

### Sensor Pointing

Additional constraints on the pointing angle of the imaging system can be employed based on the capabilities of the sensor platform. Two types of camera pointing will be considered here: fixed pointing, and fixed stare point. A system with a stationary camera has fixed pointing, such that the principle axis of the camera remains unchanged in relation to the aircraft. A system with a mobile camera can employ a fixed stare point, such that the principle axis of the camera changes in relation to the position of the aircraft relative to the fixed stare point in the scene. Each case defines the camera pointing, thereby eliminating the remaining two degrees-of-freedom associated with the camera pointing.

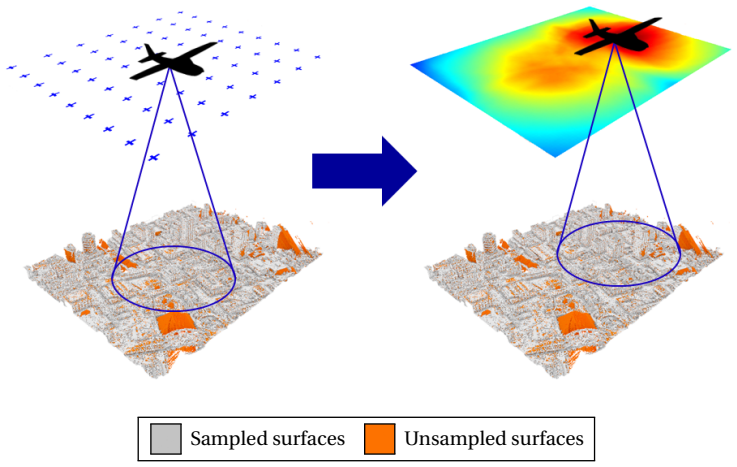



*Figure 3.24: An illustration of the process used to obtain imaging locations. A regular grid of potential positions for the aircraft are shown above the model (left). At each location the number of voxels on the free-unsampled boundary that are visible is computed, which is used to make the heat map (right), where locations shown in red can see more voxels than those shown in blue.*

### 3.3.5 Sensor Positions

With the previous constraints implemented, the solution space has been reduced to only two degrees of freedom, the $(X, Y)$ location of the aircraft. Due to the nature of the cost function, and the amount of computations required for each $(X, Y)$ location, the locations will also be quantized. Quantization should have little impact on the results, provided that the distance between locations is small, given that positioning of an aircraft in flight is less than exact depending on flight control capability.

For each camera location, the ray-tracing algorithm is employed for every voxel within the circular field-of-view of the camera to determine whether the line-of-sight is clear. The process is repeated in different locations, allowing for the development of a map, where the value at each location is indicative of the number of boundary voxels of interest that were visible from that location using the specified pointing angle. This map can then be used to determine optimal imaging locations by looking for the locations that saw the greatest number of unsampled voxels. An illustration of this process is shown in Fig. 3.24.

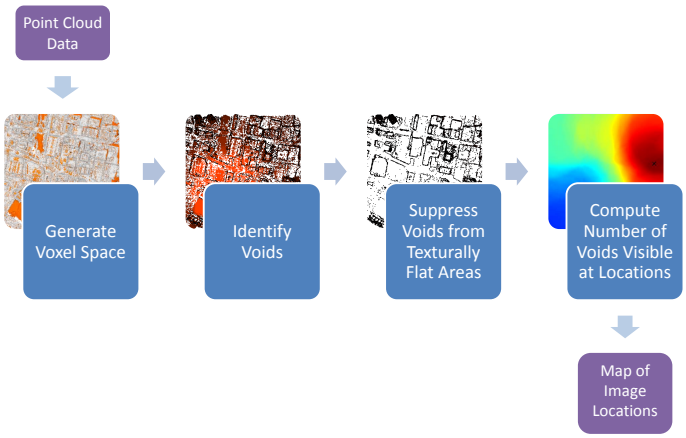



*Figure 3.25: High level diagram of the work flow used to obtain future aircraft image locations from point cloud data.*

## 3.4 Voxel-Based Workflow

An end-to-end work flow diagram of the procedure used to obtain potential aircraft image locations from point cloud data is shown in Figure 3.25. Using the point cloud as input, a voxel space is generated using the procedure from Section 3.1. Voids in the voxel space are identified as voxels that lie on the free-unsampled boundaries and voids that are a result of homogeneous areas are removed from consideration using the procedure described in Section 3.2. Finally the remaining voxels of interest are used to identify potential image locations as described in Section 3.3, and the result is a map of potential future image locations, where the value at each location corresponds to the number of unsampled voxels that can be seen from that location and pointing angle.

# Chapter 4

# Results and Analysis

The following Chapter presents results using the voxel-based workflow. Generation of the voxel spaces was tested on both real and synthetic data to verify it was performing as expected. Detailed descriptions of the datasets used is available in Appendix A.

## 4.1 Voxel-Based Visibility Analysis

### 4.1.1 Validation of Approach with DIRSIG Data

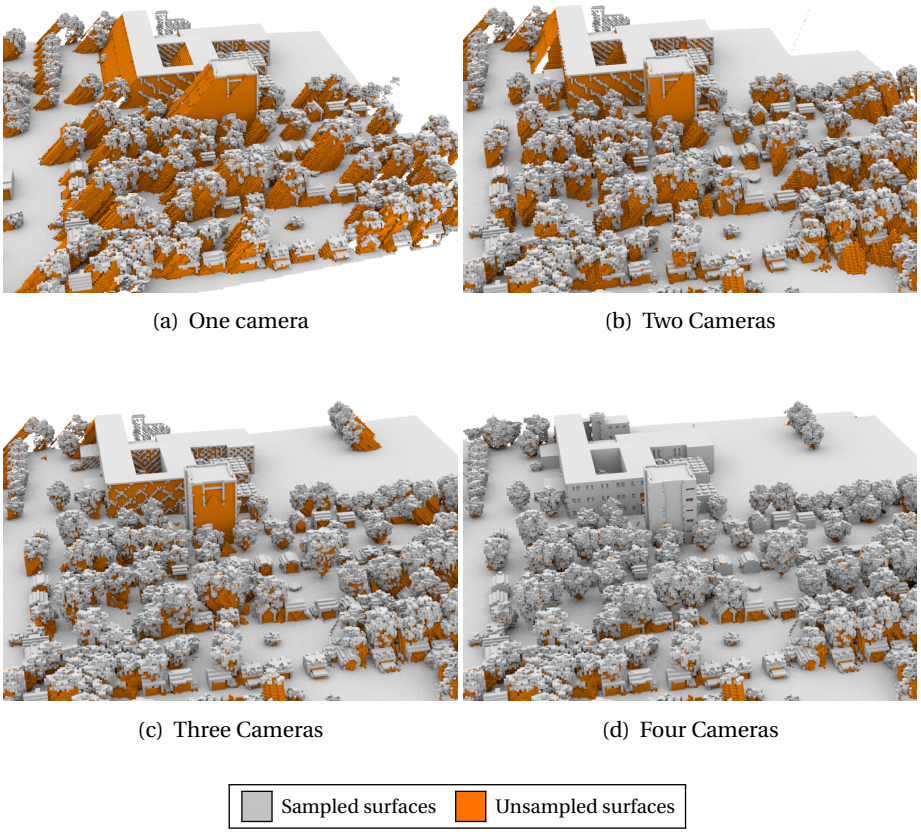
The algorithm to generate the voxel space, outlined in Section 3.1, was applied first to truth data from a synthetic data set that was generated using DIRSIG. The DIRSIG dataset is unique, in that each image is accompanied by truth data which provides a 3D location and point normal for every pixel. The truth data can be used to generate a point cloud for each image such that every pixel in the image has a corresponding point in world coordinates. These point clouds can then be used to test the voxel workflow, without relying on the 3D reconstruction process to generate a point cloud, to ensure that the voxel space creation is functioning as expected. The DIRSIG truth data provided a set of noise-free initial data to test, and using the truth data made it possible to study a point cloud derived from a single image and combinations of vantage points that would not produce results through the 3D workflow due to lack of correspondences.

Voxel spaces were computed using truth data sets from one, two, three, and four images (e.g. the truth point clouds from the corresponding images were combined), where the im-

*Figure 4.1: Nadir view of th synthetic for reference, shown for comparison purposes with the DIRSIG results.*

ages were captured facing cardinal directions. Given that the important information in the voxel space lies in the boundaries, as discussed in Section 3.2.1, 3D models of the space can be created by visualizing the free-occupied boundaries (i.e. sampled surfaces), and the free-unsampled boundaries (i.e. unsampled surfaces). This type of visualization is shown in Figure 4.2 (created with Blender [79]), where the sampled surfaces are shown in grey, and the unsampled surfaces are shown in orange. In this case, any voxel that contained points from the point cloud was treated as occupied. A nadir view of the scene for comparison purposes is shown in Figure 4.1.

In the one-camera case, sampled surfaces only exist on the side of the model facing the camera. On the side of the model that is oriented away from the camera, large areas of unsampled surfaces are evident in the shadowing that results from the ray-tracing. As more views are added, the number of unsampled surfaces in the model decreases and the shadowing effects begin to disappear as views overlap. When the four views are combined, the majority of the voxel space is sampled surfaces, and the unsampled surfaces that remain exist under the tree canopies, or in areas where the four views did not overlap completely. The results presented here using simple cases indicate that the voxel space generation algorithm is performing as expected.




(a) One camera (b) Two Cameras

(c) Three Cameras (d) Four Cameras

*Figure 4.2: Visualization of the DIRSIG voxel spaces in 3D, where sampled surfaces are shown in gray, and unsampled surfaces are shown in orange. In the single-camera case (a), the shadowing from the ray tracing is readily apparent. This effect is reduced when the second camera is added in (b), and more surfaces are filled in. In the four-camera case (d), most of the unsampled voxels lie beneath the trees and on the outskirts of the model where all four views did not overlap.*

### 4.1.2 Generation of Voxel Spaces from Image-Derived Point Clouds

Expanding the voxel space generation algorithm for results from the workflow is a straightforward problem. The patch files, output by the PMVS algorithm, indicate which cameras were used to reconstruct each point. The ray-tracing algorithm is used to determine the free space between the point and each camera used to reconstruct the point. The following results depict downtown Rochester, NY, and the point cloud imagery was collected with the Exelis WAMI sensor. The point cloud is the same one that was shown in Section 2.1.7, and more information on the sensor is available in Appendix A.

Using the WAMI point cloud data, a voxel space was derived using 4.0m voxel resolution, where all voxels containing points were treated as occupied. The results are shown in Figure 4.3 along with the corresponding point cloud. The most notable feature in the voxel space are the "walls" of unsampled surfaces that surround the model. The irregular shape is due to the boundaries of the point cloud, but the walls themselves are not a mistake in the process. The walls are indicative of a free-unsampled boundary around the point cloud, which is created because the voxel space is defined as the bounding cuboid of the point cloud, which encompasses regions beyond the irregular shape of the point cloud. The rays to the points on the boundary of the point cloud will carve out the free space, but beyond that there is no information to be added, and therefore those voxels remain unsampled creating the free-unsampled boundary walls around the bounds of the point cloud. This effect can be seen in the illustration in Figure 3.9, where voxels that are outside of the field-of-view of the camera are unsampled, and those inside are free, resulting in the creation of a free-unsampled boundary. The walls are angled in this case due to the oblique nature of the imagery.

As the walls are not of particular interest here, an edge condition was defined to mitigate their existence. An unsampled voxel is considered to be an edge voxel if the edge of the voxel space can be reached in either the $X$ or $Y$ direction without encountering a sampled voxel. This removes a majority of the walls surrounding the point cloud. Methods have also been implemented to extract portions of the completed voxel space that define a particular region of interest.

Another point of interest here is some of the shadowing effects that are evident along the edges of the model where not as many views were used in the reconstruction, indicating that these areas could benefit from more views from different perspectives. Unsampled surfaces are

also scattered throughout the model, as expected based on the lack of data in some regions of the point cloud. At this point it is important to consider the effect that the parameters of the voxel space impact the number of unsampled surfaces.

### 4.1.3 Investigation of Voxel Space Parameters

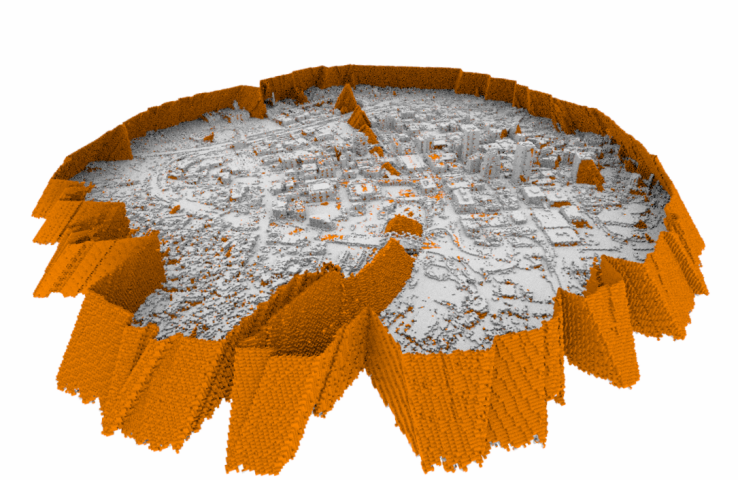
Construction of the voxel space is currently dependent on two user-defined parameters: the voxel size and the threshold on the probability of free space, used to classify the voxels into the three classes. The effect of those parameters is investigated here, using the WAMI dataset.

#### Voxel Size

A voxel space is defined by the size of the voxels, where the voxels are cubic in nature for the purposes of this work. There are trade-offs to consider between the size of the voxel, the processing time, and the amount of data to store. Decreasing the voxel size by a factor of 2 increases the memory required to store it by a factor of 8, in addition to increasing the number of voxels that a ray must pass through, thereby increasing processing time. The limiting factor in the processing time is the voxel traversal algorithm, which increases as $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$ as voxel size decreases, depending on the data. Depending on the size of the point cloud, it has been necessary to limit the voxel resolution in some cases due to limitations in memory capacity.

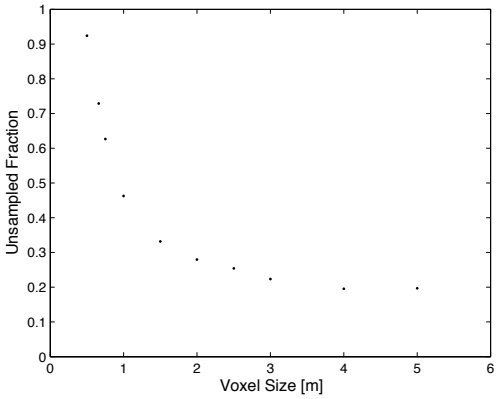
While it may seem intuitive that smaller voxels will provide an increased level of resolution in the voxel space, this is only true if the dataset is sufficiently dense to support this higher resolution. The effect of the voxel size as a function of the number of unsampled surfaces in the model is shown in Figure 4.4. The unsampled fraction of voxel surfaces was computed using a subset of the WAMI point cloud voxel space. The fraction was computed for two different subsets and the results were very similar. The relationship between the unsampled fraction and the voxel size exhibits an exponential decay. The point of interest here is the rapid inflation of the unsampled surface fraction as the voxel size approaches zero. While it is important for voxels to be sufficiently small to capture details in the scene, it is also important that the voxels are sufficiently large such that the free space in the scene can be adequately sampled. The rapid increase in the number of unsampled surfaces in the scene as the voxel size decreases is a result of undersampling the free space. Undersampling the free space results in spurious unsampled voxels, thus creating an influx of unsampled surfaces.

(a) Image Derived Point Cloud




(b) Voxel Space with 4.0m Voxels

*Figure 4.3: The WAMI image derived point cloud is shown in (a), and the sampled and unsampled surfaces of the voxel space (computed with a voxel size of 4.0m) are shown in (b). Note the apparent walls of unsampled surfaces that surround the point cloud. This is a result of the ray tracing algorithm and the oblique nature of the imagery.*




| Voxel Size [m] | Unsampled Fraction |
|---|---|
| 0.5 | 0.92402 |
| 0.66 | 0.72868 |
| 0.75 | 0.62661 |
| 1 | 0.46273 |
| 1.5 | 0.33201 |
| 2 | 0.27972 |
| 2.5 | 0.25431 |
| 3 | 0.22344 |
| 4 | 0.19546 |
| 5 | 0.19693 |

*Figure 4.4: Plot of the fraction of unsampled voxel faces versus the voxel size [m]. Note that the number of unsampled voxel faces rapidly increases as the voxel size decreases.*

The effect of voxel resolution is more apparent visually, particularly in regard to the spurious voxels at small voxel sizes. The model is shown in Figure 4.5 with voxel sizes ranging from 0.5m to 4.0m. At 4.0m resolution, the model is blocky. More details appear in the model at resolutions of 2.5m, 2.0m, and 1.5m, and while the number of unsampled surfaces increases, the model itself is not compromised. The unsampled surfaces lie predominantly on the building surfaces and the ray tracing still appears to have carved out the free space nicely. At 1.0m resolution, more surfaces appear to be unsampled and spurious unsampled voxel appear in what should be free space around the model. As the voxel size decreases further, the spurious unsampled voxels increase and it becomes obvious that the point cloud data cannot support such fine voxel resolutions. The point density of the voxel space, as well as the rays that are used to carve out free space, dictate the lower limits of resolution in a voxel space.

At this time, there is no method for automatically determining the voxel size for a given point cloud as it is not a straightforward calculation. Multiple voxel spaces could be generated, and the unsampled surface fraction computed to determine the optimal size, as was the case to generate Figure 4.4, but computation times are long which would make this expensive. The voxel size can be loosely based on the point density, but this is a 2D metric, and will not necessarily correlate to the 3D voxel space. The most important thing in determining the voxel size is that the free space around the model is carved out and spurious voxels are minimal. This is easily accomplished visually, and in practice a voxel size can be determined from a few runs.

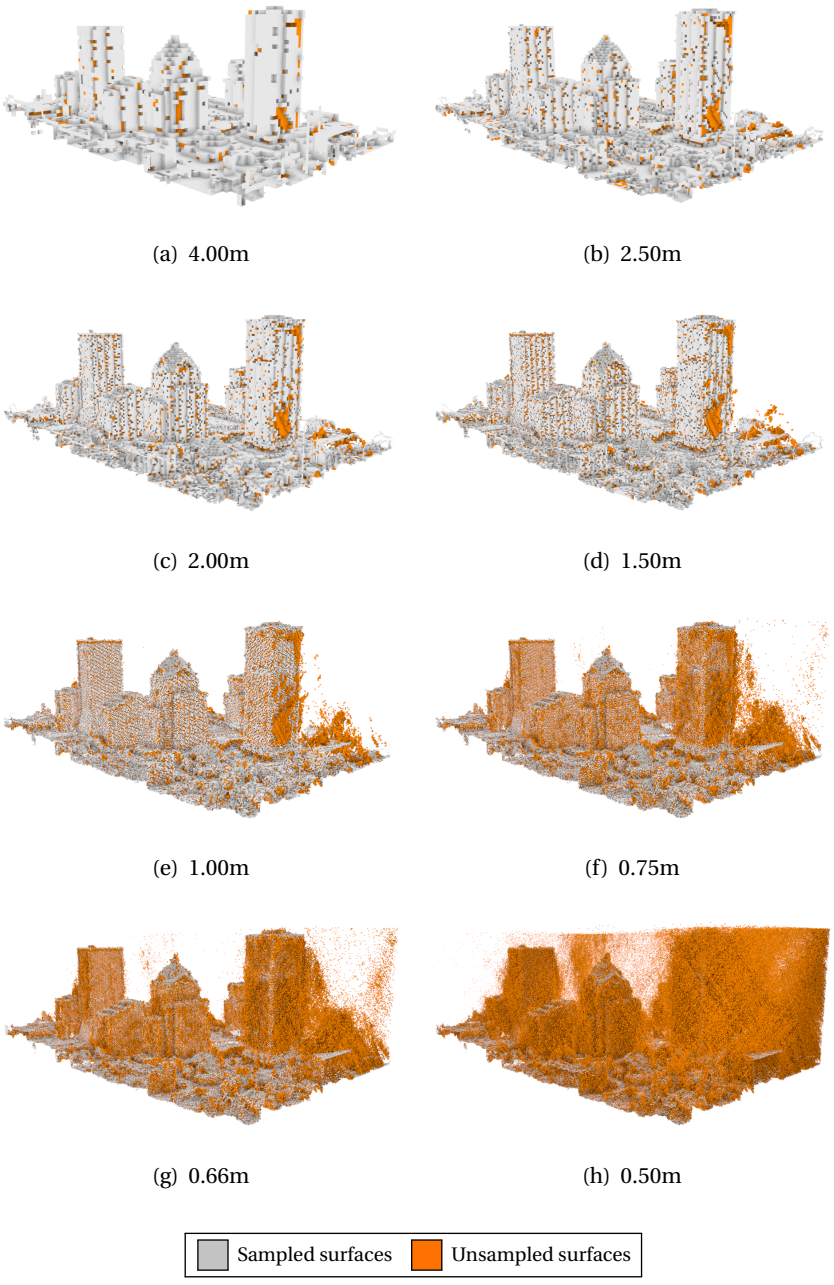



*Figure 4.5: The effect of voxel resolution on unsampled regions. Surface voxels are shown in grey and unsampled voxels in orange. There are few extraneous unsampled voxels at voxel sizes above 1.50m, though the number of unsampled voxels on the buildings increases as the voxel size decreases. Additionally, spurious unsampled voxels begin to appear around the model in areas that should be free space as the voxel size decreases. At 0.50m, it is obvious that the voxel space cannot support this resolution.*

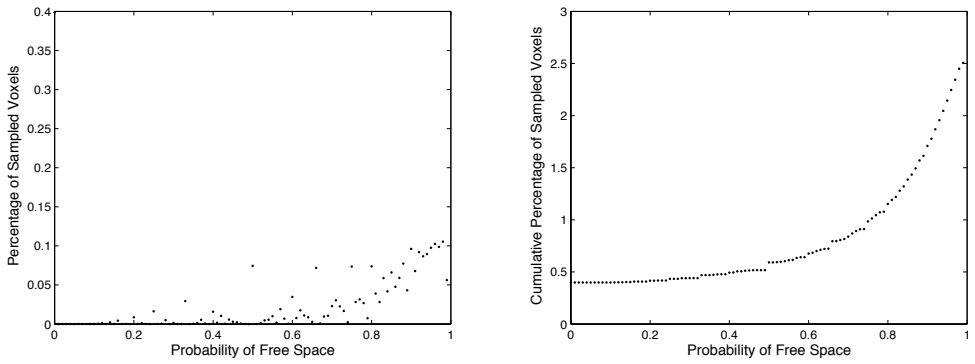


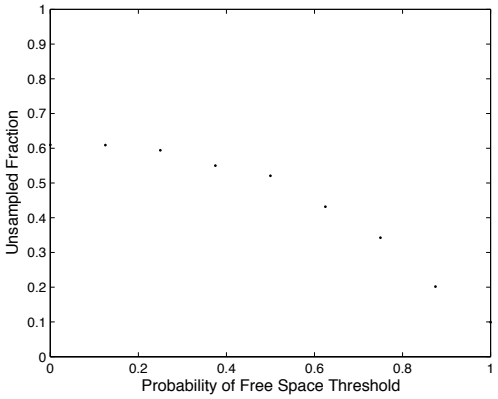

*Figure 4.6: Left: Plot of the percentage of sampled voxels as a function of their probability of free space. Right: Plot of the cumulative percentage of sampled voxels as a function of their probability of free space.*

**Probability of Free Space**

The other user-defined parameter required for the creation of the voxel space is the threshold on the probability of free space, used to classify the occupied and free voxels. The threshold is employed such that a voxel with a probability greater than or equal to the threshold is classified as free, and a voxel with a probability less than the threshold is classified as occupied. Unsampled voxels are unaffected by the threshold. A subset of the WAMI voxel space, computed at 2m resolution, was extracted for analysis.

For a total voxel count of 23.4 million voxels, 67.25% were sampled, and 32.75% were unsampled. While this may seem like a high percentage of unsampled voxels, consider that all voxels that lie underneath the first visible surface will remain unsampled, regardless of the completeness of the voxel surface model. Considering only the sampled voxels, 97.49% contained only ray interactions, 0.40% contained only point interactions, and 2.11% contained both point and ray interactions. Again, it may seem that the percentage of voxels containing only ray interactions is high, but all of the voxels that lie above the first visible surface should be identified as free space. Thus the overall percentage of occupied voxels is expected to be small in comparison to both the percentage of free voxels and the percentage of unsampled voxels.

The important thing to consider here is the 2.11% of sampled voxels that contained both point and ray interactions, where the probability of free space will be greater than 0, but less than 1. These voxels are important because they can affect the number of unsampled surfaces




| $P_{free}$ Threshold | Unsampled Fraction |
|---|---|
| 0.000 | 0.60965 |
| 0.125 | 0.60910 |
| 0.250 | 0.59418 |
| 0.375 | 0.55014 |
| 0.500 | 0.52080 |
| 0.625 | 0.43184 |
| 0.750 | 0.34262 |
| 0.875 | 0.20188 |
| 1.000 | 0.09909 |

*Figure 4.7: Plot of the fraction of unsampled voxel faces versus the probability of free space threshold.*

in the model, depending on the probability of free space threshold that is used to classify them. A more detailed breakdown of the percentage of sampled voxels as a function of their probability of free space is shown in Figure 4.6. While the number of unsampled voxels will not change as a function of the threshold, the number of unsampled surfaces is dependent on the threshold, because an unsampled surface is defined as a transition between a free voxel and an unsampled voxel. A voxel classified as occupied at one threshold, may be considered free at a lower threshold, thus creating new unsampled surfaces if any neighboring voxels are unsampled.

Again, using a subset of the WAMI voxel space, the number of unsampled surfaces was computed as a function of the threshold on the probability of free space. The downtown subset was used, and the results are shown in Figure 4.7. Assuming all voxels with point interactions are occupied ($P_{free} < 1.0$ is occupied), less than 10% of the surfaces are unsampled. This increases to over 60% when it is assumed that all points that have ray interactions are free ($P_{free} > 0.0$ is free). This plot was computed for multiple subsets, and while the absolute unsampled fraction differed, the general trend was the same. However it is difficult to discern from this plot exactly what the effect is on the voxel space.

The effect of the threshold is more apparent visually. The model is shown in Figure 4.8, with $P_{free}$ thresholds, $\tau$, ranging from 0.0 to 1.0, where a voxel is classified as occupied if $P_{free} < \tau$. Note that for the 0 case, a small positive number was used such that voxels that contained only point interactions were the only ones classified as occupied. At $\tau = 1.0$, a majority of the model

is comprised of sampled surfaces, which appear to provide an accurate representation of the scene. As the threshold decreases, the number of sampled surfaces decreases. The sampled surfaces that are lost as a result of the decreased threshold appear to become unsampled surfaces, and the shape of the model does not appear to be significantly different.

In this case, the increase in unsampled surfaces in the model would negatively impact the identification of void areas as well as the identification of future image locations. As can be seen with at $\tau = 1.0$, the sampled surfaces provide an accurate model and to classify them as voids would result in future image location predictions that are heavily influenced by previously sampled areas. For this reason, the threshold used on $P_{free}$ to classify the voxels as occupied/free will be unity. This parameter could potentially be used to develop partially transmissive models, but that will be left for future work.

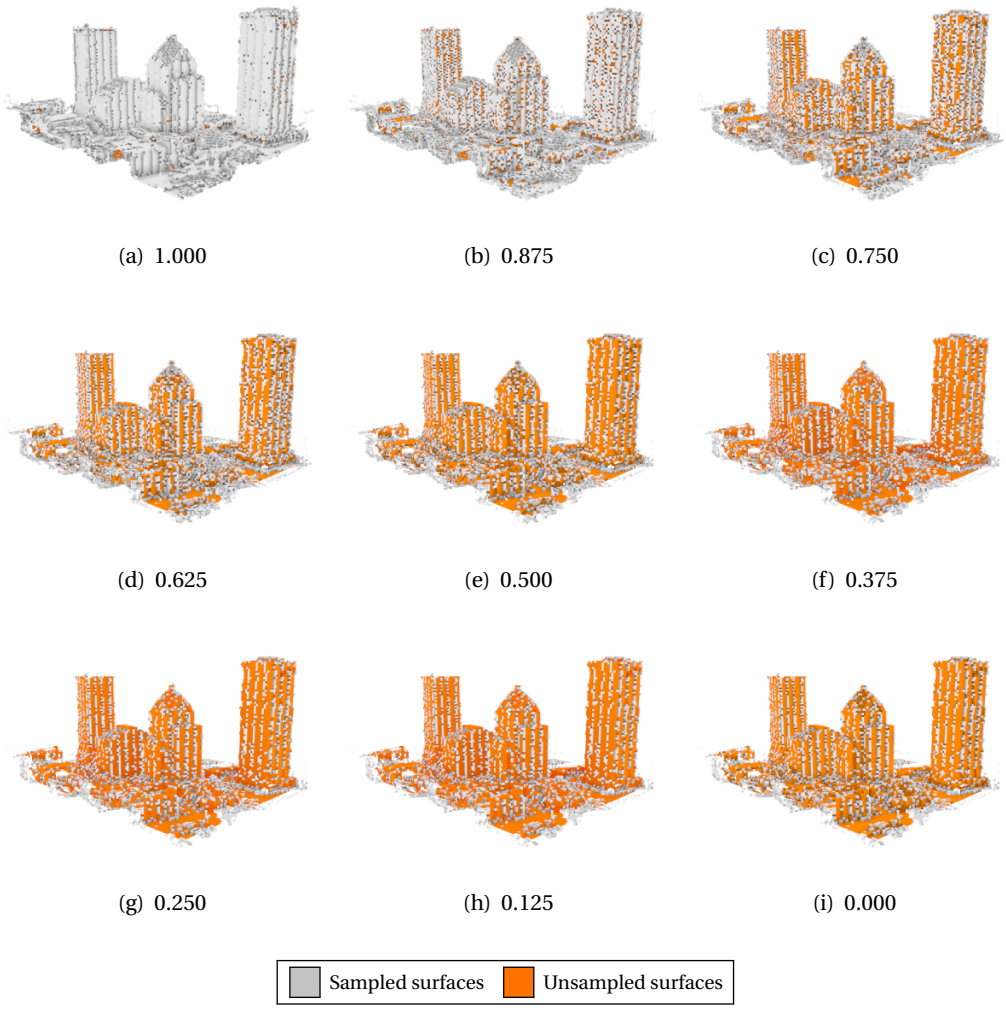



*Figure 4.8: The effect of threshold $\tau$ of the probability of free space $P_{free}$ unsampled regions, where a voxel is classified as occupied if $P_{free} < \tau$. Voxels with a Sampled surfaces are shown in grey and unsampled surfaces in orange. A decrease in the threshold results in more unsampled surfaces, however the general shape of the model remains unchanged.*

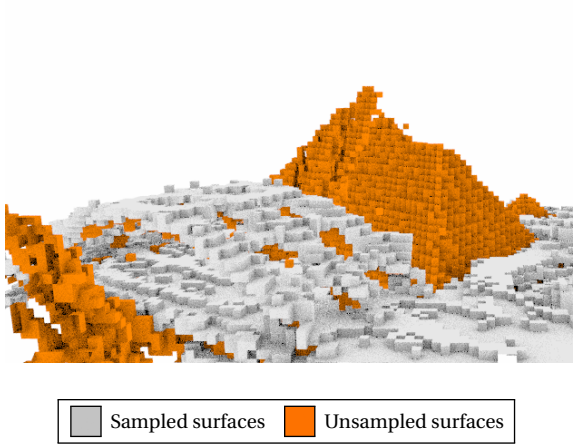



*Figure 4.9: Example of the pyramid structure that appears in the unsampled surfaces where large areas lacked points, such as the Genessee River shown here. The pyramid structure is a result of the ray-tracing methods and the oblique nature of the imagery. The shape is dependent on the imaging geometry, which will be explored in more detail with additional datasets.*

## 4.2 Identification of Voids in the Voxel Space

Once the voxel space has been created, the voids are identified by finding voxels that lie on the free-unsampled boundary. The sampled and unsampled surfaces, created from the free-occupied and free-unsampled boundaries respectively, have been shown in Figures 3.10, 3.12, 4.3, 4.5, and 4.8. As was discussed in reference to the shadow effect shown in Figure 3.12, the ray-tracing methods employed and the oblique angle of the imagery creates some unique features in the unsampled surfaces in the voxel space. This effect produces the "shadow" effects on buildings, but also produces a pyramid-like structure in holes near the ground plane. An example of such a pyramid is shown in Figure 4.9, in the Genessee River. Because there were no points at the ground level in the river, ray-tracing to the edges of the river left unsampled voxels in the pyramid shape. An illustration of this phenomenon is shown in Figure 4.10. While these effects appear strange, they are expected, and serve to further illustrate why it is not reasonable to assume that unsampled surfaces can be used as an approximation to real surfaces in all cases. Note that the shape of the unsampled surface structure is dependent on the imaging geometry, which will be explored further in Section 4.5 with additional datasets.

As detailed in Section 3.2, a list of voxel faces that lie on the free-unsampled boundary is

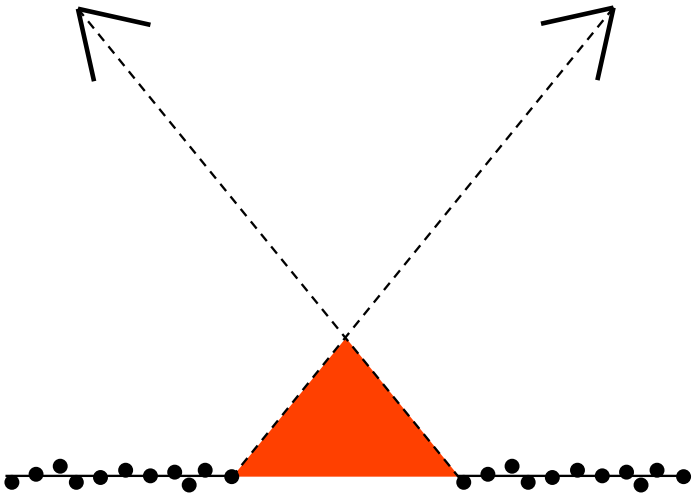


*Figure 4.10: A 2D illustration of how a lack of points in an area can result in pyramid-like structures. Note only a central ray from each camera to the edge of the missing data area has been shown for clarity.*

generated, where each voxel face is defined by its center point and a surface normal, oriented in the direction of free space. Before using these unsampled faces to generate potential future imaging locations, the visibility of the points in reference to the current cameras must be determined such that a texture metric can be computed for each unsampled voxel face.

### 4.2.1 Visibility Analysis

The centers and normals of the unsampled faces are used to determine which cameras used in the 3D reconstruction process had a clear line-of-sight of the unsampled surface. This computation of visibility is necessary, because there is no visibility information associated with the unsampled voxels. A detailed description of what constitutes visibility of a face for a given camera is available in Section 3.2.2.

A histogram of the count of unsampled voxel faces as a function of the number of views in which they were visible is shown in Figure 4.11, computed for 2m voxels. The histogram is skewed right with the exception of a dramatic spike between 25 cameras and 26 cameras (where 26 cameras total were used in the reconstruction). These voxels are likely voxels on rooftops and

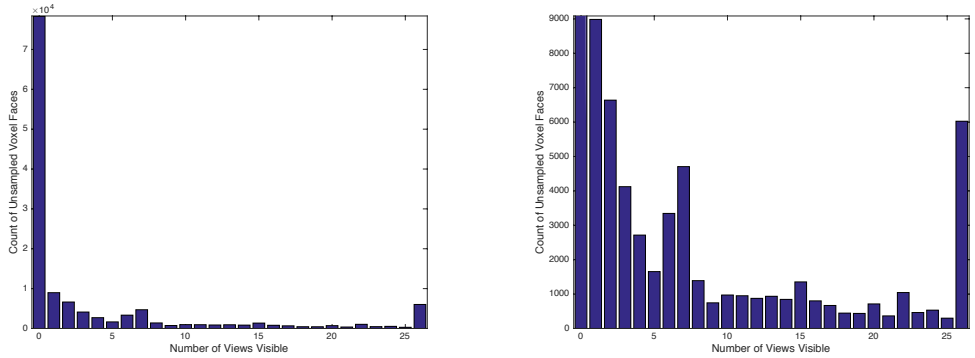



*Figure 4.11: Histogram of the count of unsampled voxel faces as a function of the number of views in which they were visible. Note that the plot is dominated by voxels that were not visible in any frames (left); another scale is shown (right) to see the detail above the zero value.*

in the river area that had a clear view of the sky and thus could be seen by all cameras. More than half of the unsampled voxel faces were not visible in any views. It is important to note that the plot is based on a single dataset and the shape will be dependent on the voids in the input point cloud. A visualization of the unsampled voxel centers, shown as points, is presented in Figure 4.12, where the brightness is indicative of how many cameras could have imaged each point. Note the samples in the river are bright, indicating they were seen in many views.

Unsampled voxels that were imaged by two or fewer views are guaranteed not to be reconstructed, given the constraints used in the 3D workflow. Also note, depending on the angular disparity between the camera views used in the creation of any point, that reconstruction is not guaranteed just because a point was seen by 3 or more views.

### 4.2.2 Texture Analysis

As was stated previously, voids that are a result of texturally difficult areas will not benefit from the inclusion of more imagery. Therefore one of the goals of this stage is to identify the unsampled voxel centers that were likely the result of a texturally difficult area and remove them from consideration when identifying future image locations. An image-based method for doing this, based on the local standard deviation, was outlined in Section 3.2.2.

To verify that the reprojection of the unsampled voxels was working correctly, the unsam-

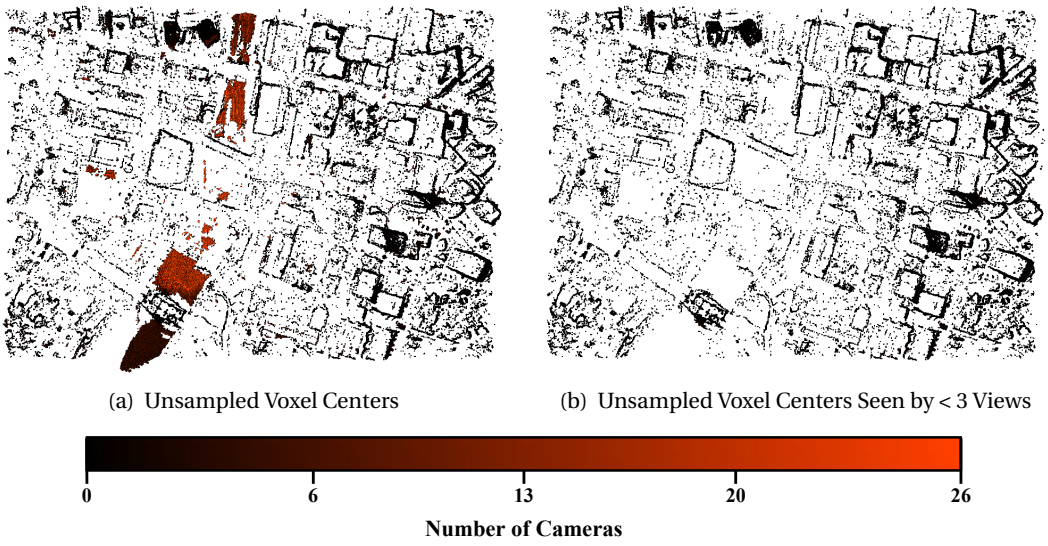



*Figure 4.12: Unsampled surface centers are shown as points for voxel resolutions of 2m, where the brightness is indicative of how many cameras could have imaged each point. Left: All unsampled surface centers. Right: Unsampled surface centers seen by less than three cameras.*

pled voxel centers were reprojected onto a WAMI image. The results are shown in Figure 4.13. The unsampled voxel centers appear to accurately reproject into the image based on a visual observation of their locations. The circular nature of the reprojected voxels is a product of using a circular field-of-view for camera when determining if an unsampled voxel is visible from a particular viewpoint. Using a 100x100 window size, the local standard deviation was computed for each unsampled voxel. As the unsampled voxels are often visible in multiple images, the local standard deviation was computed in each view in which the unsampled voxel was visible, and the results were averaged across the views. Unsampled voxel centers with a local standard deviation greater than 4.00 digital counts are shown in green, and those less are shown in red. The threshold was chosen based on a visual inspection of the reprojected voxels, such that a majority of the river region had a standard deviation that was less than the threshold to suppress those voxels from influencing the identification of future image locations.

The purpose of the texture metric is to identify textureless regions that would fail to generate correspondences in the densification process, as inclusion of more imagery will not benefit these regions. The standard deviation texture metric is dependent on the window size and the

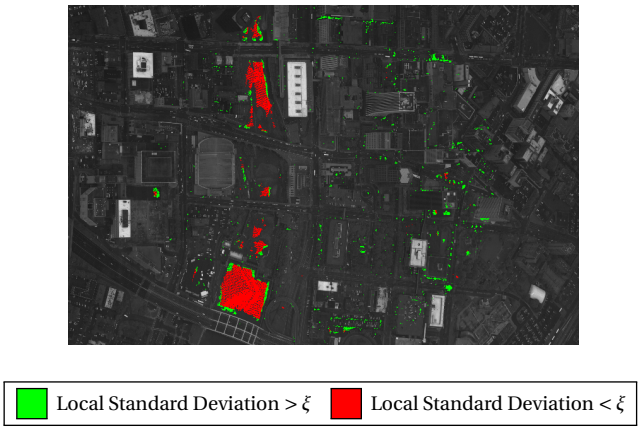



*Figure 4.13: The unsampled voxel centers have been reprojected into a WAMI image, and those shown in green have a local standard deviation greater than 4.00, while those shown in red have a local standard deviation less than 4.00 for a given window size of 100x100. The circular nature of the unsampled voxels is a result of the circular camera model implemented.*

threshold, $\xi$, where voxels with a local standard deviation less than $\xi$ are not considered in the future image location identification process. The effects of these parameters were investigated, using the same WAMI image shown in Figure 4.13.

The local standard deviation of a WAMI image was computed for various window sizes, shown in Figure 4.15; a 2D view of the point cloud is shown in Figure 4.14 for context and the holes are highlighted by the blue background. A threshold of 2.00 digital counts was used to highlight the effect that the window size has, given the same threshold. The river is easily pulled out at each window size, using the given threshold. Note that as the window size is reduced to a 25x25 pixel region, there is a significant increase in the number of regions that are identified using the 2.00 threshold.

Using the 100x100 pixel window, different thresholds were used to determine the effect of the threshold on the texture metric; the effect of the threshold is shown over a threshold range from 2.00-4.00 digital counts, where again a local standard deviation less than the threshold $\xi$ is highlighted with green. As the threshold increases, more regions are identified as possibly textureless regions. At a threshold of 2.50, textureless regions begin to appear on the roof of the Blue Cross Arena, which was reconstructed in the point cloud, and the regions expand as the threshold is increased. While this would be considered a false alarm, it would have no impact

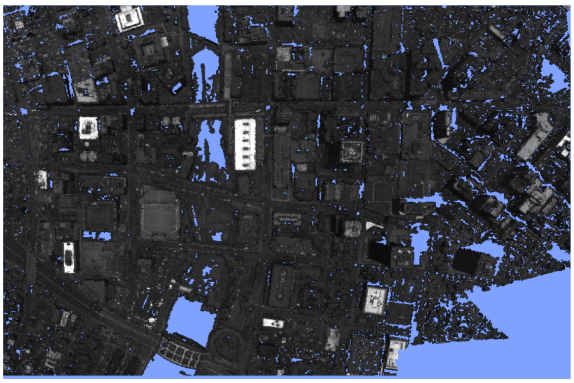


*Figure 4.14: Nadir view of the WAMI point cloud, provided for context in the texture analysis, where the blue background was used to highlight voids.*

on the final results because there are no unsampled voxel surfaces in that region. However it could potentially impact regions that were not reconstructed if they are misidentified.

The effect of the window size and threshold in Figures 4.15 and 4.16 was computed for a single image. When computing the texture of an unsampled voxel, the local standard deviation is averaged across the views in which the unsampled voxel was visible. A histogram of the count of unsampled voxels as a function of local standard deviation is shown in Figure 4.17; this histogram was computed for all unsampled voxels, and unsampled voxels that were seen by 3 or more cameras. As mentioned previously, voxels that were visible in 2 or fewer views are guaranteed not to be reconstructed regardless of their image texture due to constraints of 3D reconstruction algorithms. Note the reduction in the count of values in the tail on the right side of the histogram when voxels that were only visible in 2 or fewer views are not included.

There is a complex relationship between the window size and threshold, and the subsequent effect on the identification of textureless regions. The window size and threshold are tied to each other, but also to the size of the sensor, the ground sample distance (GSD), and the feature detection and densification schemes. At this time, it is unclear the best method for determining a window size and/or threshold to use to suppress unsampled voxels from textureless regions. The effect of the window size and threshold on the future image location predictions will be investigated in Section 4.3.

It may be possible to eliminate the window size parameter by considering groups of voxels that make up continuous voids. Continuous regions of unsampled surfaces could be identified

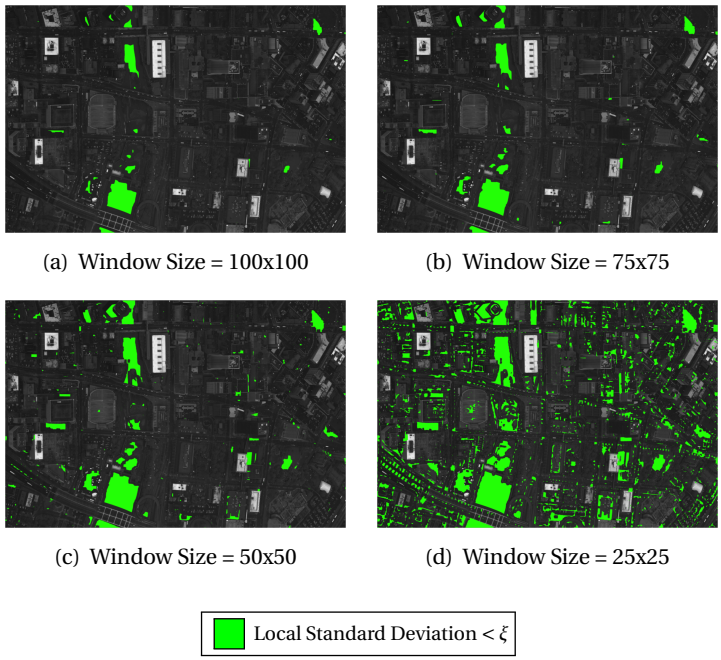



*Figure 4.15: The local standard deviation was computed for a WAMI image with a window size of indicated and local standard deviations less than* $\xi = 2.00$ *are highlighted in green.*

as a void region, and the centers of those unsampled surfaces could be reprojected into the imagery, where the local standard deviation could be computed from the pixels onto which the unsampled surfaces reproject. In this case, the standard deviation would be computed using just pixels that are identified as being a part of the void region, and that standard deviation could be assigned to all of the pixels in the void. There aresome potential issues with this technique. The first is in voids that are made up of a very few unsampled surfaces, resulting in a local standard deviation computed from just a few pixels. In this case, it may be safe to eliminate voids that are less than a specified size, and use the unsampled surfaces as estimates of the missing surfaces if the voxel space is used to create a surface model. Other surface modeling techniques, such as ball-pivoting or Poisson surface reconstruction could also be used to fill in these small holes. Another potential issue is the shadowing effect in the voxel space. Unsampled voxel surfaces, particularly in large regions, are not guaranteed to represent actual surfaces in the scene, as shown in the building shadow (Figure 3.12) and the river pyramid (Fig-

ure 4.9). While they may be identified as a single void, it is possible that only portions of these unsampled surfaces would be visible in any given image, and a method to handle this would have to be implemented. Unsampled voxels in these regions, where the unsampled surfaces compose surfaces that are not actually present in the scene, may reproject into the imagery onto a region that is not the real missing surface. In this case, the local standard deviation that is computed would not be representative of the missing region. An illustration of this is shown in Figure **??**, where voxels that make up a building shadow are reprojected into the image and fall partly on the desired missing side of the building, but also partly on the parking lot behind the building. Due to the reasons presented here, this process was not implemented, but may warrant future investigation.

It is undeniable that including textureless regions in the identification of future image locations will have an impact on the predicted locations, particularly if these regions are large and composed of a significant number of unsampled surfaces. While it is clear that a change in threshold and/or window size has an impact on the regions that are identified as textureless, it is difficult to predict the effect that a slight change in threshold would have on the predicted future image locations. Thus this matter will be investigated further in subsequent results. It should be noted that this method was designed to identify homogeneous image regions. Spatially repetitive regions, reflective regions, and possibly others will also have textural difficulties in the SfM workflow, but the standard deviation texture metric will likely not identify such regions. Another method would be to use the visibility analysis, and set a threshold based on the number of cameras that imaged an unsampled voxel space. The basis of this is that if numerous images of the region were collected but it was not reconstructed, it is likely a texturally difficult region. This is less intensive computationally, as the visibility must be computed already, and will also be explored further in subsequent results.

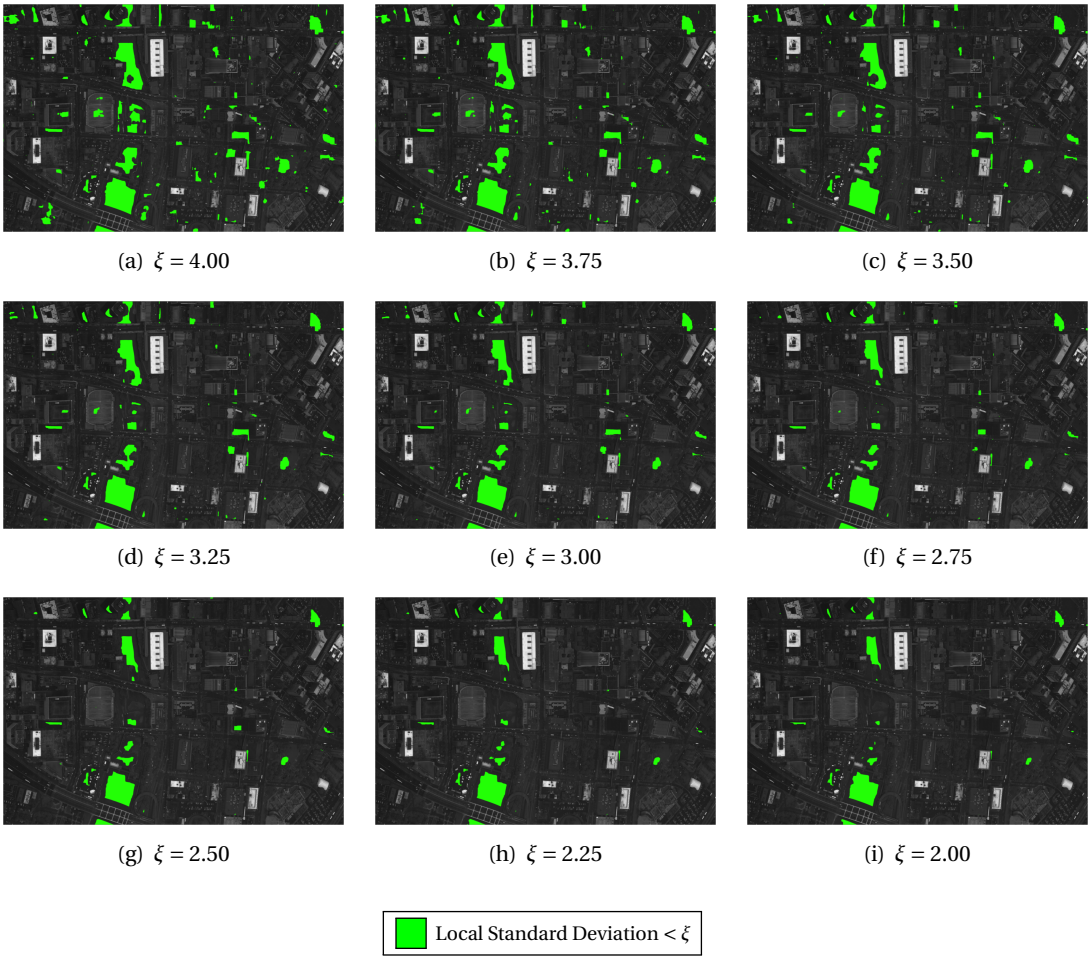
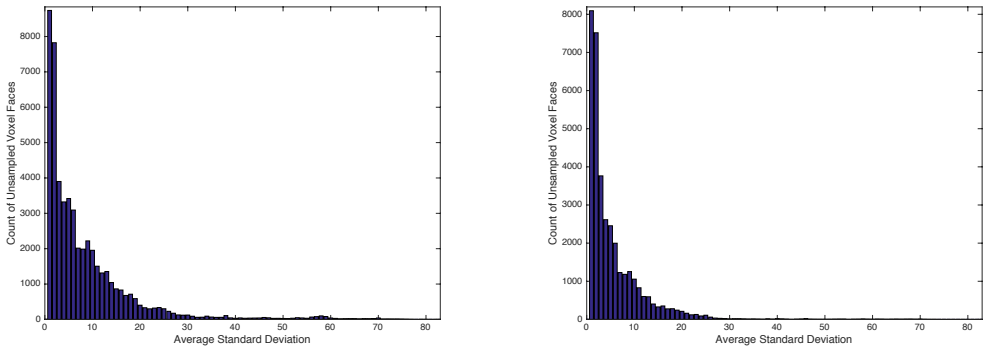


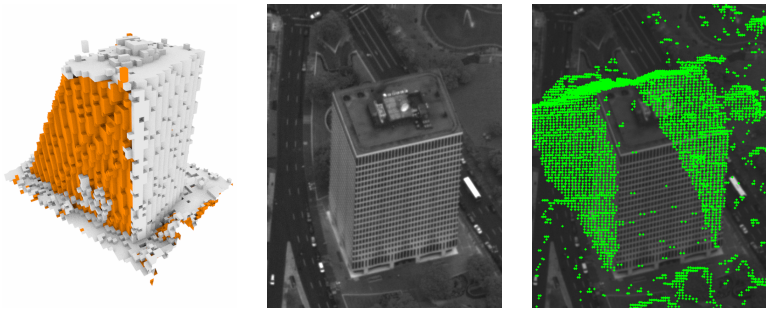

*Figure 4.16: The local standard deviation was computed for a WAMI image with a window size of 100x100; the effect of the threshold is shown over a range of thresholds, where local standard deviations less than $\xi$ are highlighted in green. Note that the river is easily pulled out, and more green regions appear as the threshold increases.*




(a) Unsampled Voxel Centers

(b) Unsampled Voxel Centers Seen by < 3 Views

*Figure 4.17: Histogram of the count of unsampled voxel faces as a function of the average local standard deviation computed from the views in which they were visible. The histogram is shown for all unsampled voxels (left) and for unsampled voxels seen by 3 or more cameras (right).*



(a) Voxel Space Building with Shadow

(b) Reference Image

(c) Reprojected Voxel Points

*Figure 4.18: A building with shadowing effects in the voxel space (a), a reference image (b), and the unsampled voxel points reprojected on the image (c), shown in green. This is an example of how unsampled voxels may reproject onto regions of the image that are not representative of the missing surfaces in the voxel space.*

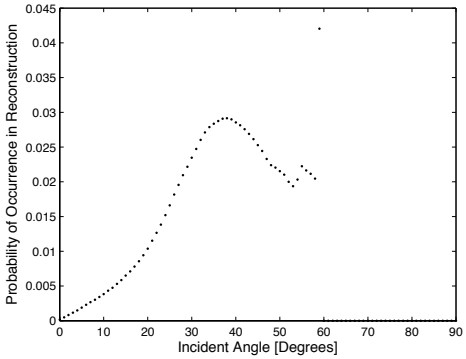



*Figure 4.19: Plot of the probability of occurrence of incident angles in the 3D reconstruction using the WAMI data, captured at 40 degrees off-nadir. The 60 degree cutoff is a function of the PMVS algorithm.*

## 4.3 Identification of Future Image Locations

Once the unsampled voxel centers have been identified, the next step is to use another line-of-sight analysis to determine how many unsampled voxels could be seen from each potential imaging location. For the purposes of this analysis, occupied and unsampled voxels are considered to be opaque. First a discussion of the weighting function is presented, followed by results for both fixed pointing and fixed stare point sensors.

### 4.3.1 Weighting Function

As was mentioned in Section 3.3.3, a weighting function can be applied to the contributions of each unsampled voxel based on the angle of incidence. This is based on the assumption that grazing angles are not well suited to 3D reconstruction of a surface. Because the optimal incident angle for surface reconstruction is not intuitive, point cloud data from a 3D reconstruction was used in an attempt to derive a data-driven weighting function. Each 3D point obtained from the PMVS point cloud is associated with an estimated surface normal, in addition to the camera visibility information. Thus for each point, an incident angle was computed for each camera view in which the point was visible by computing the angle between the estimated surface normal and the incident ray, defined by the camera center and the point. This method was performed on the WAMI point cloud, and the incident angles were tallied to compute a probability of occurrence for each angle; the results are shown in Figure 4.19.

The results of the probability of occurrence of each incident angle of the WAMI point cloud at first seem counter intuitive. While the lack of angles approaching 90 degrees (i.e. grazing) is expected, the peak of the curve would intuitively be expected to be at 0 degrees, as it would seem that a straight-on view of a surface would be best. However, consider that the WAMI sensor is looking at an oblique angle, approximately 40 degrees off-nadir and the scene reconstruction is mostly urban. Surface normals in this environment would be primarily upward for roof-tops and ground surfaces (dominant Z-component) and horizontal for building sides (weak Z-component). Considering the 40 degree off-nadir viewing angle of the WAMI sensor, the peak around 40 degrees makes a little more sense. The cause of the abrupt cutoff at 60 degrees is the result of a threshold in the PMVS process.

The same process was used to derive the probability of occurrence of each incident angle in a 3D reconstruction using WASP data that was captured at nadir; the results of that are shown in Figure 4.20. A 3D reconstruction derived from nadir data of an urban scene contains mostly points on the ground plane and on building rooftops, points on the building sides are notably missing due to their lack of visibility in the nadir imagery. Using the logic applied to the WAMI point cloud, the majority of the surface normals in the urban scene should be upward facing (dominant Z-component). With the expected surface normals and the nadir looking sensor, it would be expected that 0 degrees would be the incident angle with the highest probability of occurrence. It is evident in the plot shown in Figure 4.20 that this is not the case, and in fact it is the angle with the least probability of occurrence when excluding those beyond the 60 degree cutoff. There is a break in the plot at 45 degrees, where suddenly there is a rapid increase in the probability of occurrence in the reconstruction from about 45 to 60 degrees. The cause of peak is unknown. It was initially thought that it may have some significance in relation to the baseline of the imagery, however considering that the majority of surfaces that are reconstructed in the WASP point clouds have dominant Z-components and the WASP sensor rays would be at most $\pm 18$ degrees when capturing at nadir, this is more likely a result of incorrect normal estimations.

The behavior of the incident angles in regard to the WASP data was unexpected, and warranted further investigation. The creation of these maps was based on the assumption that the estimated surface normals for each point from PMVS was reasonably accurate. Upon further analysis, it was discovered that this assumption was incorrect. The estimated surface normals resulting from PMVS are not an accurate representation of the expected surfaces. To illustrate

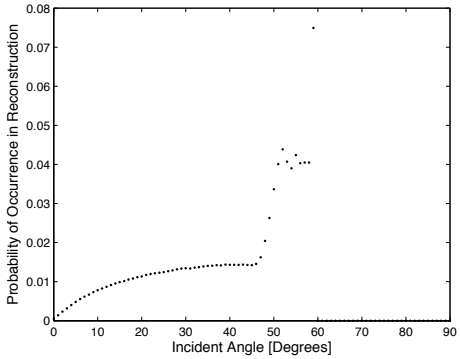
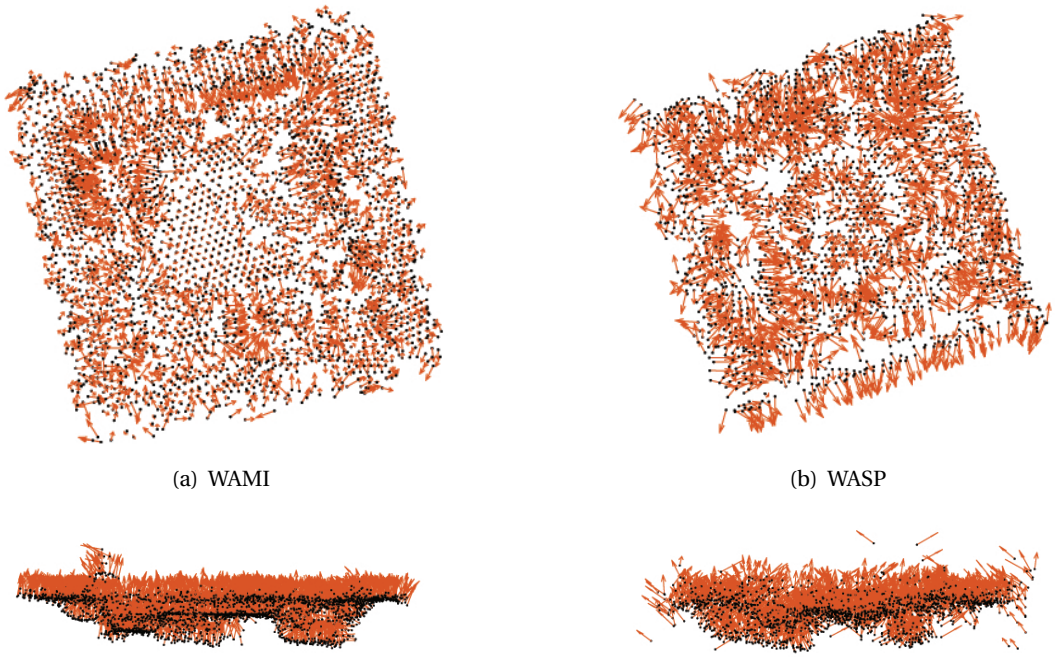



*Figure 4.20: Plot of the probability of occurrence of incident angles in the 3D reconstruction using the WASP data, captured at nadir. The 60 degree cutoff is a function of the PMVS algorithm.*

this, the rooftop of the Chase building was extracted from both the WAMI and WASP point clouds. The surface normals were plotted in conjunction with the points, and the results are shown in Figure 4.21. It is expected that the rooftop normals would be upward facing, oriented in the Z-direction with small X and Y components. However, it is evident from the overhead view that this is not the case. The estimated surface normals do not behave as expected and do not represent the surface accurately. Based on this information, the incident angle plots, shown in Figures 4.19 and 4.20 were based on inaccurate surface normals, and do not provide accurate estimates of the distribution of incident angles. Therefore, the data-derived incident angle weighting functions will not be used.

Upon further consideration, the distribution of incident angles is dependent on a variety of factors. As mentioned in the discussion of the results from the WAMI normals, the angle of incidence distribution is highly dependent on the sensor and look angle. The distribution is also a function of scene content, as the surface normals in an urban scene will differ from those in a rural scene. The densification process, PMVS in this case, would also play a role based on which points are reconstructed. Finally, it is important to consider the requirements for reconstruction, namely the number of images and the baseline between them. A minimum of three images is required for reconstruction of a surface. Given one image taken at 0 degrees with respect to a surface, the other images need to be separated enough to get an accurate triangulation but not so much so that feature detection will fail. Thus there is a complex relationship to consider between likelihood of reconstruction and the incident angle on a surface. Because



(a) WAMI (b) WASP

*Figure 4.21: Estimated surface normals from PMVS for each point on the rooftop of the Chase building for both the WAMI and WASP point clouds. Top: View from directly overhead, down the Z-axis with the X-axis extending horizontally and the Y-axis extending vertically. Bottom: View from the side, down the Y-axis with the X-axis extending horizontally and the Z-axis extending vertically. It is expected that the surface normals would be oriented such that the Z-component dominates (i.e. upward pointing), however as can be seen, this is not the case. The estimated surface normals from PMVS are highly susceptible to noise and do not provide an accurate representation of the surface.*

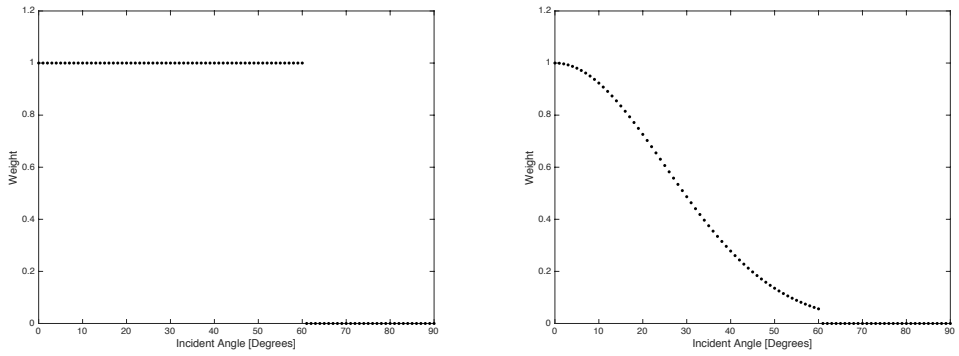



*Figure 4.22: Left: Plot of the uniform weights as a function of the angle of incidence. Right: Plot of the Gaussian weights ($\sigma = 25$) as a function of the angle of incidence. A 60 degree cutoff has been employed in each case.*

of the associated complexity, a data-driven weighting function, even one derived with accurate surface normals, would be unreliable.

For the purposes of this research, two weighting functions will be considered: uniform weighting and Gaussian weighting. The uniform weighting function will weight all observations equally, provided that the angle of incidence between the camera and surface is less than a given cutoff angle to account for grazing effects. The Gaussian weighting function will be computed with a specified standard deviation, with unity occurring at an incident angle of 0 degrees; a cutoff angle will also be employed. Examples of the weighting functions are shown in Figure 4.22. A 60 degree cutoff on the incident angle will be used, due to the cutoff that exists in PMVS. The effects of the different weighting functions will be investigated in subsequent results.

### 4.3.2 Fixed Pointing: Nadir

The first configuration that was tested was a sensor located at 10,000ft (nominal height of the original WAMI collection), with a nadir fixed pointing angle. A subset of the voxel space was used to reduce computation time while allowing the focus of the future image locations to be within a central area of interest in the point cloud. The footprint of the voxel space was broken up at altitude into 25m x 25m blocks, and the number of unsampled voxel centers visible from each location was computed. Unsampled voxel face centers at the free-unsampled boundaries,

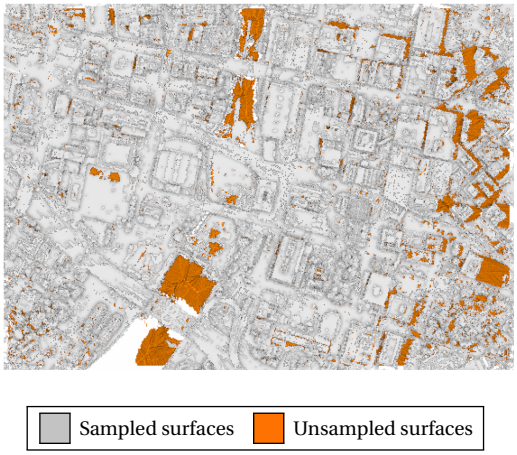
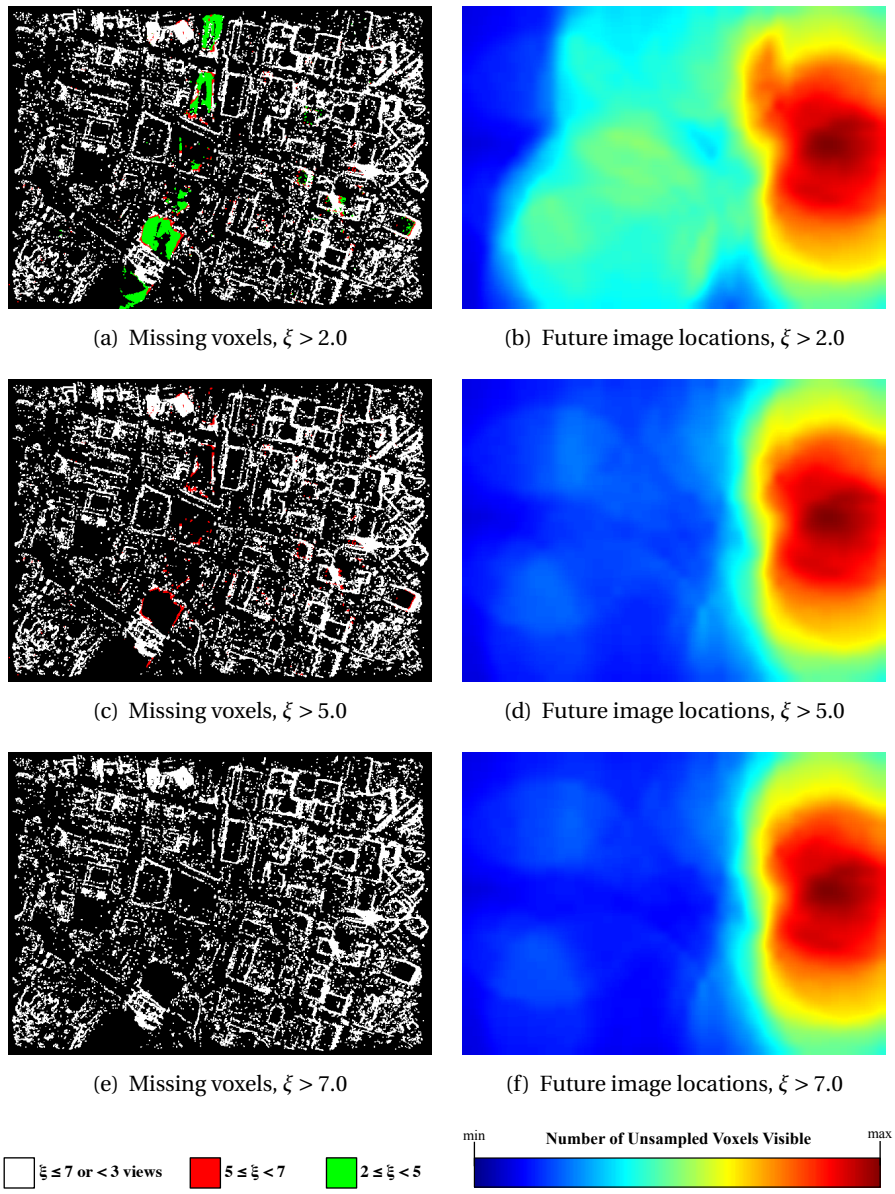



*Figure 4.23: Orthographic view of the voxel space for reference.*

and the results of the potential image location calculations are shown in Figure 4.24, where an orthographic view of the voxel space has been included for reference in Figure 4.23. Results are shown for three different texture thresholds (2.0, 5.0, 7.0), where the local standard deviation was computed from a 100x100 pixel image region.

The circular region toward the lower left was the location of the fixed stare point for the WAMI dataset, and that is likely why the region is not as densely populated with unsampled voxel centers. The majority of the unsampled voxel centers are from the homogeneous regions of the river. Higher concentrations of unsampled voxel centers are found on the right portion of the scene, which was located on the "outskirts" of the voxel space. As such many of the buildings are missing multiple sides and exhibit shadowing effects due to a lack of west-facing views. This region also has a large number of unsampled voxel faces that were not visible in any camera views, and are therefore not affected by the application of the different texture thresholds. As expected, this region corresponds to the highest concentration of unsampled voxels that are visible from the potential image locations, because of the density of the unsampled voxels. While there are slight differences noticeable in the potential image location maps based on the number of visible unsampled voxels, particularly in views in which the river would have been visible, the locations where the maximum number of unsampled voxels were visible was the same for each threshold. Results were computed using a larger 200x200 window size, and again the location was the same, even across different thresholds.




(a) Missing voxels, $\xi > 2.0$

(b) Future image locations, $\xi > 2.0$

(c) Missing voxels, $\xi > 5.0$

(d) Future image locations, $\xi > 5.0$

(e) Missing voxels, $\xi > 7.0$

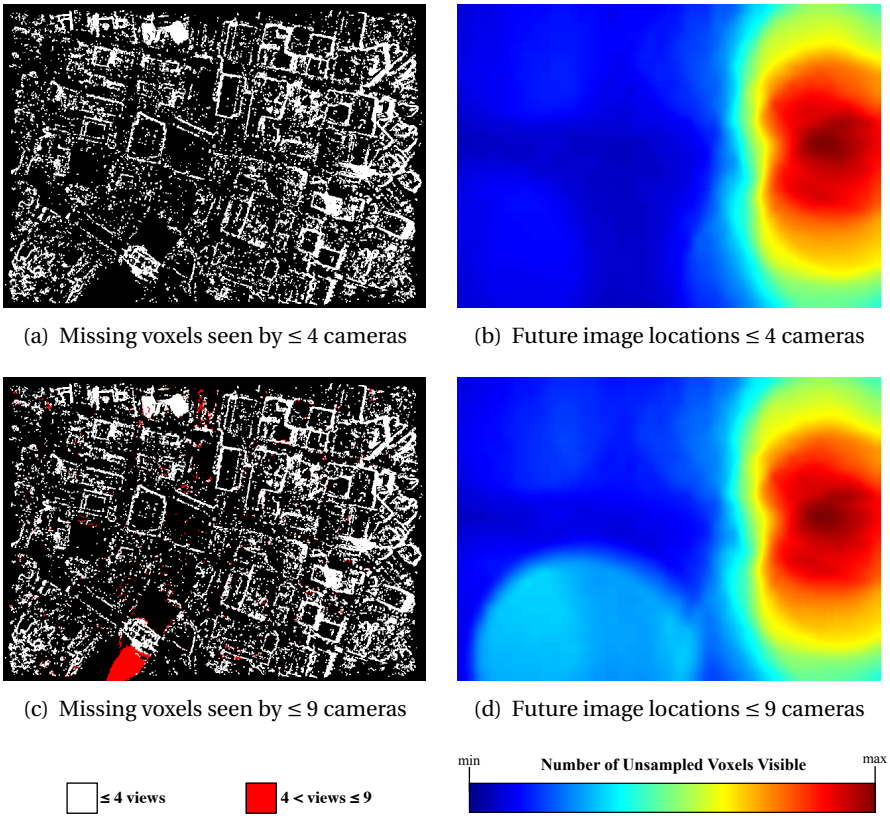
(f) Future image locations, $\xi > 7.0$

*Figure 4.24: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras, in addition to unsampled voxels that were seen by 3 or more cameras and had a standard deviation of greater than 2.0, 5.0, and 7.0 in (a), (c), and (e) respectively. The standard deviation window size was 100x100. Corresponding heat maps of future aircraft image locations are shown in (b), (d), and (f), where a nadir pointing angle and aircraft altitude of 10,000ft have been specified. While there are slight differences in the appearances of the maps, based on how many unsampled voxels were greater than the specified threshold, the location where the most unsampled voxels were visible was unaffected.*

Because the predicted optimal location remained unchanged based on the texture threshold, the future image location code was also run using a threshold on the number of cameras that could see an unsampled voxel. Note that this is computed for visibility purposes, and was used in preliminary results as a way to identify textureless regions. In this case, unsampled voxel faces that were visible in more views than the specified threshold are excluded from consideration, where it is assumed that voxels that appeared in numerous previous images will not benefit from the addition of more imagery. Unsampled voxel face centers used at the free-unsampled boundaries, and the results of the potential image location calculations are shown in Figure 4.25. Results are shown for two different thresholds, where unsampled voxels were visible in four or fewer views, and nine or fewer views.

Again, there are slight differences in the appearances of the maps as the number of unsampled voxel centers included in the threshold increases, but the location where the most unsampled voxels were visible was unaffected, and remained the same as the one computed using the texture threshold. The fact that the location does not change, regardless of using the texture metric or just a threshold on the number of views, suggests that perhaps a threshold on the number of views a voxel was visible in is sufficient criteria for inclusion or exclusion in the future image location process.

It should be noted that all of the results shown here were computed using a uniform weighting function and a Gaussian weighting function, as shown in Figure 4.22, however there was no perceivable difference. This is a result of the distribution of normals in the voxel space, where normals are distributed either vertically or horizontally based on the face of the voxel. The incident angle, computed between the voxel normal and ray defined by the camera and the unsampled voxel face center, was computed for each unsampled voxel within the field of view of a camera for a single pointing location; the resulting histogram is shown in Figure 4.26. The distribution is distinctly bimodal, with peaks near 0 and 90 degrees. Horizontal unsampled surfaces have a normal with only a Z-component, and using a nadir-pointing camera, these angles will fall within the angle of the sensor field-of-view (5 degrees in this case), hence the peak close to zero. Conversely, vertical surfaces in the voxel space have either an X-component or a Y-component, and would be approximately 90 degrees different from a nadir pointing angle, where again the distribution is a result of the different angles across the sensor field of view. With the 60 degree cutoff in the weighting function, vertical surfaces are given weights of 0, and thus do not contribute to the total of visible voxel faces. The components that do contribute




(a) Missing voxels seen by ≤ 4 cameras

(b) Future image locations ≤ 4 cameras

(c) Missing voxels seen by ≤ 9 cameras

(d) Future image locations ≤ 9 cameras

*Figure 4.25: Unsampled voxels at free-unsampled boundaries that were seen by four or fewer cameras and nine or fewer cameras are shown in in (a) and (c) respectively. Corresponding heat maps of future aircraft image locations are shown in (b) and (d), where a nadir pointing angle and aircraft altitude of 10,000ft have been specified.*

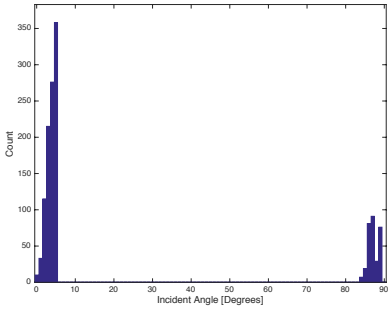



*Figure 4.26: Incident angle histogram computed for a single location with a nadir pointing angle.*

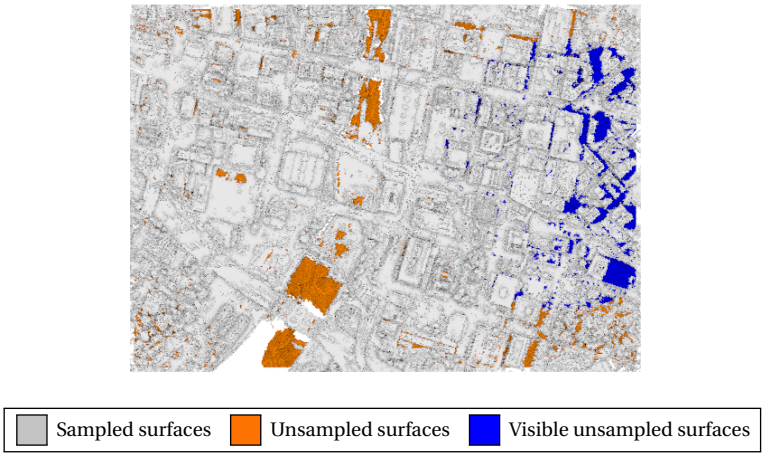



*Figure 4.27: View of the voxel space where voxels that were visible from the indicated location are shown in blue; voxels shown in orange are indicative of unsampled surfaces that were either not visible or did not meet the given texture criteria.*

are distributed so closely to 0, that the difference between the uniform and Gaussian weighting functions is negligible. For off-nadir pointing angles, it is expected that these peaks would shift closer together as the sensor pointing angle moves further off nadir, and thus the effects of the weighting function may become more noticeable, particularly for a fixed stare point system.

A view of the voxel space is shown in Figure 4.27, where the unsampled voxels that are visible from the indicated location are shown in blue. The blue voxels are enclosed in the circular shape of the camera view, and are located on the right side of the scene, as expected.

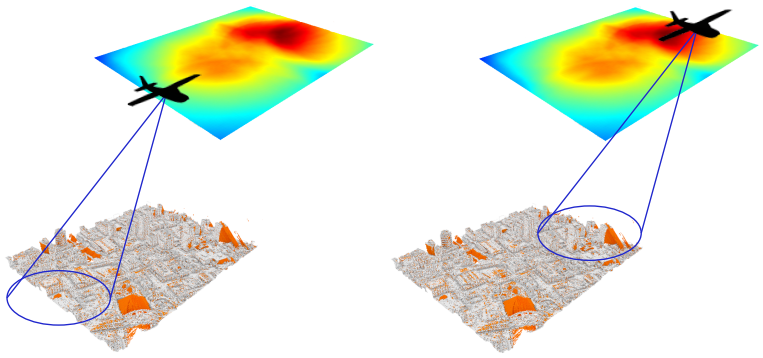


*Figure 4.28: Diagram of the shift in the plane space to account for the off-nadir pointing angle.*

### 4.3.3 Fixed Pointing: Off-Nadir

Admittedly, most of the horizontal surfaces in the voxel space that would be visible from nadir have already been sampled, as indicated in Figure 4.23, and therefore nadir views may not be the most beneficial for reconstruction purposes. It was noted that a majority of the unsampled voxels were missing sides of buildings on the eastern portion of the voxel space, where west-facing sides were reconstructed but east-facing sides were not. The next sensor configuration that was tested was a sensor located at 10,000ft, but this time with a fixed pointing angle 40 degrees off-nadir in the westward-direction (eastern faces will be visible with a westward pointing angle). This was to mimic the original WAMI collection, flown at 10,000ft and 40 degrees off-nadir. Again, a subset of the voxel space was used to reduce computation time while allowing the focus of the future image locations to be within a central area of interest in the point cloud. The footprint of the voxel space was broken up at altitude into 25m x 25m blocks, shifted eastward to account for the off-nadir pointing angle; a diagram of this is shown in Figure 4.28.

The unsampled voxel face centers at the free-unsampled boundaries, and the results of the potential image location calculations are shown in Figure 4.29. A standard deviation threshold of 7.0 was used, with a 100x100 window size. This was chosen because it seems to work well with the given voxel space in the nadir results. The circular camera field-of-view becomes more oval shaped on the ground at off-nadir angles, and as a result there is a more distinct ovular shape in the heat mapping results. The results are still skewed to one side, and this is expected due to the high concentration of unsampled voxels in that portion of the scene.

Again, the uniform and Gaussian weighting functions were tested, and while the returned

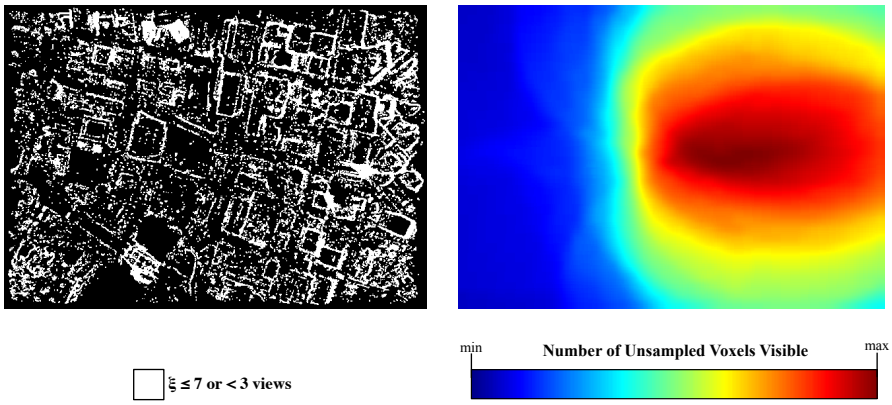


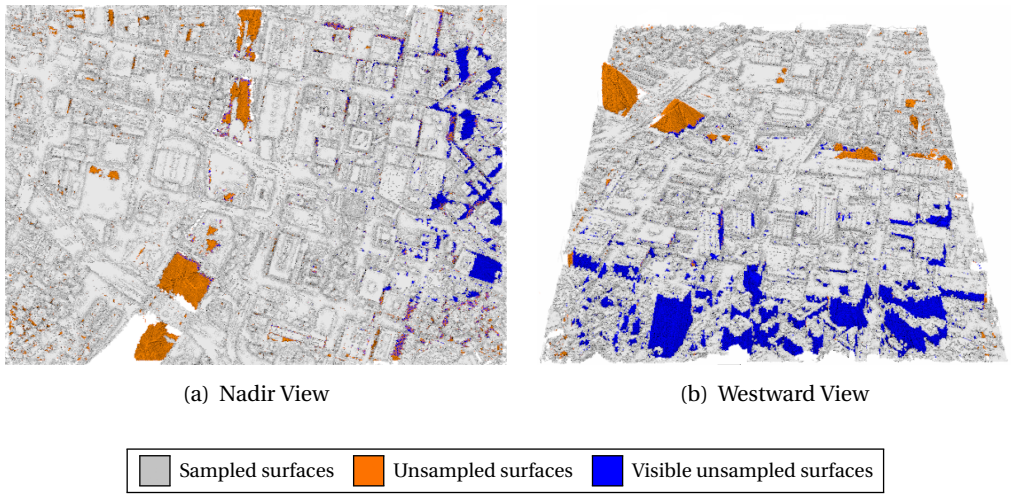

*Figure 4.29: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras, in addition to unsampled voxels that were seen by 3 or more cameras and had a standard deviation of greater than 7.0 in (a), where the standard deviation window size was 100x100. Right: A corresponding heat map of future aircraft image locations, where a 40 degree westward off-nadir pointing angle and aircraft altitude of 10,000ft have been specified.*

values differed due to the Gaussian weighting, the results were not perceptibly different and the predicted maximum between the two only differed by 25m (shifted one block in plane space). The WAMI sensor being modeled has a field of view of 10 degrees, so the angles off center only differ by ±5 degrees. Again, because the normal vectors in the voxel space are confined to align with the axes, this results in a spread of incident angles of only 5 degrees (as shown in Figure 4.26). Changing to an off-nadir pointing vector will shift these peaks toward each other, but will not have an effect on the spread. Because the normals are distributed so closely together, the weighting functions have little effect.

A view of the voxel space is shown in Figure 4.30, where the unsampled voxels that are visible from the indicated location are shown in blue; both a nadir view and westward view (similar to the predicted camera location) are presented. The unsampled voxels that are visible confirm that the predicted view encompasses many of the unsampled voxel faces.

### 4.3.4 Fixed Stare Point

The last sensor configuration to be tested was a fixed-stare point sensor. The aircraft altitude was defined to be 10,000ft, and the stare point was chosen to be a central point in the scene. The same subset of the voxel space was used to reduce computation time. The footprint of the voxel




(a) Nadir View (b) Westward View

*Figure 4.30: View of the voxel space where voxels that were visible from the indicated location are shown in blue; voxels shown in orange are indicative of unsampled surfaces that were either not visible or did not meet the given texture criteria.*

space was tripled and broken up at altitude into 25m x 25m blocks. The results are shown in Figure 4.31, where the unsampled voxels are shown, scaled relative to the future image location results.

This is the first time in which the weighting functions have had a significant impact on the results, as the predicted maximum locations differ depending on the weighting function used. In both cases, the results are heavily influenced toward those views on the eastern side of the map, which would have had westward facing pointing angles. This is in agreement with previous maps, as a majority of the unsampled voxels are found in that portion of the voxel space. One thing that may seem strange here is that the predicted values seem to increase continually from left to right. This is a result of the sensor footprint on the ground, and the effect of expanding area as the pointing angle moves off-nadir. At a nadir pointing angle, the footprint of the sensor on the ground is circular, as defined in Section 3.3.3. As the pointing angle moves off nadir, the footprint on the ground becomes larger; this effect is shown in Figure 4.32. With a larger footprint on the ground in the voxel space, more unsampled voxels are potentially visible, thereby biasing the calculation toward pointing angles that are far off-nadir.

A view of the voxel space is shown in Figure 4.33, where the unsampled voxels that are

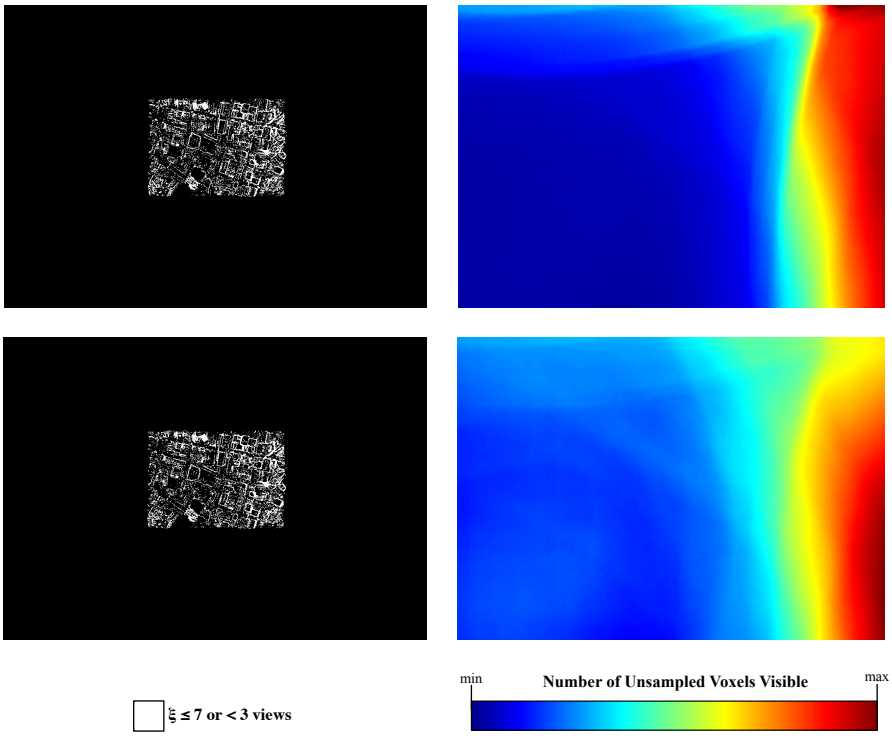



*Figure 4.31: Unsampled voxels at free-unsampled boundaries that were seen by less than 3 cameras, and those with a standard deviation greater than 7.0 are shown in (a) and (c) respectively, where the relative size has been scaled to match the corresponding heat maps of future aircraft image locations are shown with a uniform and Gaussian weighting function in (b) and (d). The sensor was located at 10,000ft with a central fixed stare point in the scene.*

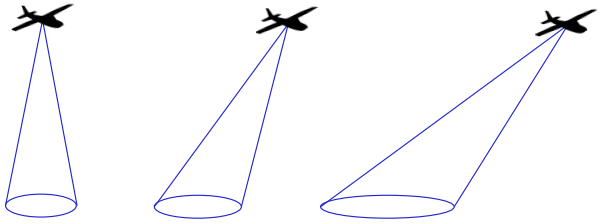
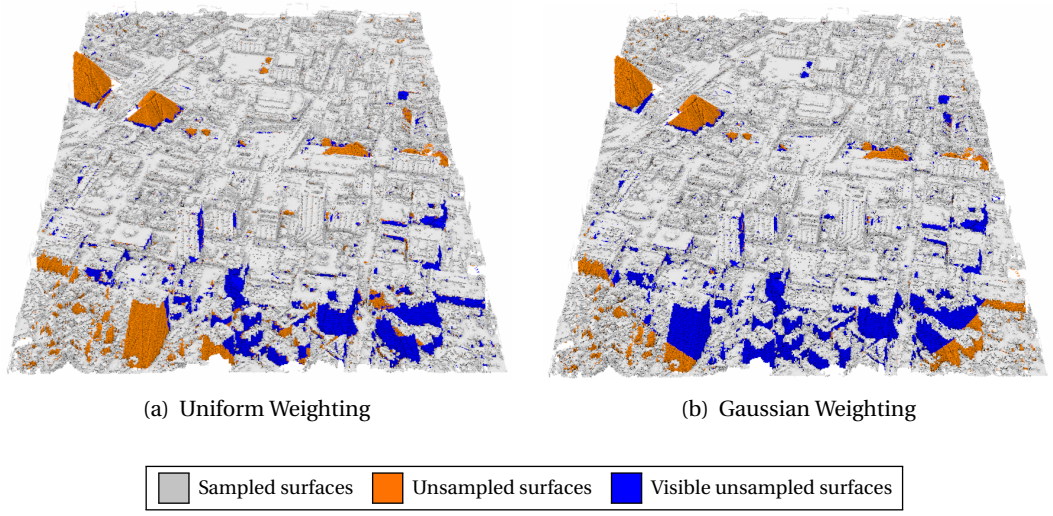


*Figure 4.32: Illustration of the effect an off-nadir angle has on the area included in the field-of-view on the ground. At increased angles, the sensor footprint on the ground is larger. Considering the cost function is based on the number of unsampled voxels visible from a given viewpoint, this will affect the results of of a fixed stare point system, giving preference to views with more dramatic angles due to the fact that the potential visibility is larger.*




(a) Uniform Weighting (b) Gaussian Weighting

*Figure 4.33: Left: View of the voxel space where voxels that were visible from the indicated location using a uniform weighting function are shown in blue. Right: View of the voxel space where voxels that were visible from the indicated location using a Gaussian weighting function are shown in blue. Voxels shown in orange are indicative of unsampled surfaces that were either not visible or did not meet the given texture criteria.*

visible from the indicated location are shown in blue; the results using the uniform weighting function and Gaussian weighting function are presented. The unsampled voxels that are visible confirm that the predicted view encompasses many of the unsampled voxel faces. A majority of the unsampled voxels that are visible are the same between the two weighting functions, with the slight differences being a result of the shifted camera positions.

To mitigate against the bias toward larger off-nadir angles due to their larger footprint area on the ground, it is recommended that the fixed stare-point configuration be used with a radius. The radius would be used to indicate a circular flight pattern, and the footprint of the sensor on the ground would be comparable. Using this method, the best view in the desired circle could be identified. A radial function could also be used to weight the results, such that there is higher cost for more severe angles.

## 4.4 Proof of Concept

Unfortunately, as was previously discussed in Section 2.1.5, there are no methods readily available to assess the accuracy and completeness of a point cloud. Additionally, two point clouds generated from the same set of input imagery are not guaranteed to contain the same points due to the way that the point clouds are generated in the 3D workflow. Due to the use of CMVS in the workflow, some input images are discarded and not used in the final reconstruction with PMVS, and this process is unpredictable. The choice of which images to include contains a random component, such that multiple runs of the same set of input images can result in different sets of images being chosen, and thus the final point clouds will differ, despite being extracted from the same initial set of input imagery. There is an option to force inclusion of all input imagery, however the point clouds that result are poorer in quality, and in some cases contain significant errors.

Additionally, no method for bundle adjustment or reconstruction exists at this time such that inclusion of additional imagery does not change the location of points in the results. Methods for 3D reconstruction often employ use of algorithms such as RANSAC, and the random component of that can alter the output, including changes to 3D point locations. New images can also be used to further refine triangulation, which would also change the location of a 3D point. Because of this, it is not possible to directly compare one point cloud to another. It may be possible to leverage the sampled and unsampled surface fractions in the voxel space to determine if a model is more complete with the inclusion of more imagery.

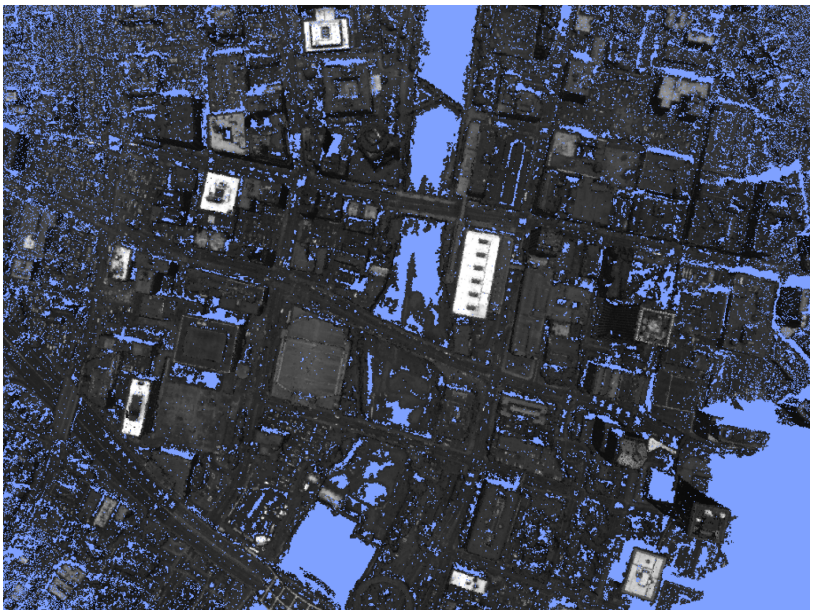
The voxel-processing and prediction of future image locations is not conducive to adding additional imagery in its current state, due to the constraints enforced. The objective in the prediction of future image locations is to maximize the number of unsampled voxel faces seen from specified locations and pointing angles. As a result of this, the predicted location may not tie into the current dataset if the viewpoint is drastically different. Therefore it is recommended that future work investigate the development of flight lines using the maps.

For the purposes of this work, the voxel-based processing will be tested by limiting the input imagery in the current 3D workflow. The WAMI data thus far has encompassed the entire circular flight pattern; in order to test proof of concept, a point cloud was generated by limiting the input imagery to half of the circle. When only using half of the circular flight pattern, the input imagery is lacking major vantage points that are necessary to complete a surface model

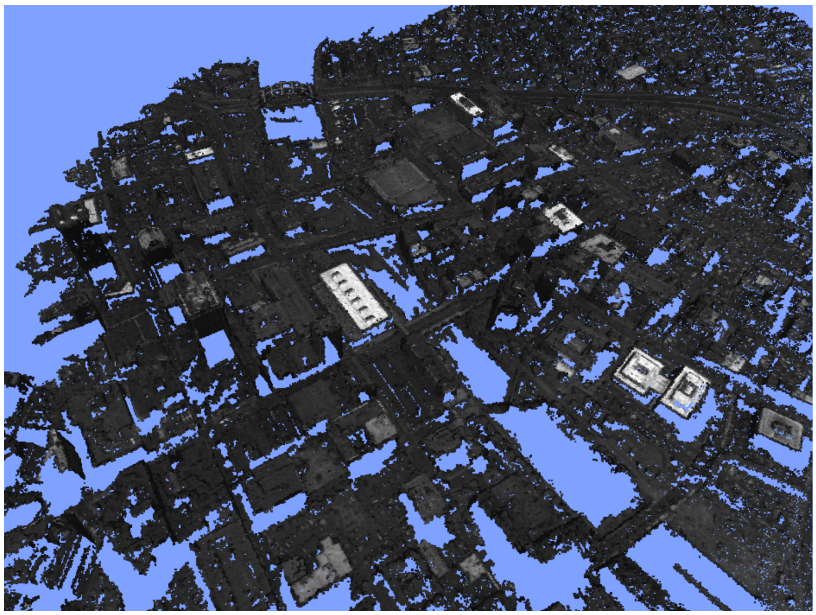
on all sides. The point cloud that was computed using the 3D workflow is shown in Figure 4.34. A voxel space was created using 2m voxels, and the result is shown in Figure 4.35. A majority of the views used in the reconstruction were north facing, and the north facing view of the voxel space shows that many of the surfaces visible from that direction are sampled. Conversely, the south facing views were removed and as a result were not reconstructed, and the voxel space shows that many of the surfaces visible from that direction remain unsampled.

Future image locations can be determined using the voxel workflow to see if it predicts imaging from the side of the point cloud where views were missing, as would be expected. Nadir and off-nadir pointing angles were tested, using an altitude of 10,000ft in all cases.

The results for the nadir sensor are shown in Figure 4.36. As predicted, the concentration of views where the most unsampled voxels were visible are located in the northern portion of the scene and the spread is distributed in an east-west fashion. It appears that there are at least two distinct local maximums in the scene, indicating that multiple views may be beneficial. The results for a southward pointing sensor, 40 degrees off-nadir, are shown in Figure 4.37. Views of the voxels that would be visible from the nadir pointing angle, and southward pointing angle are shown in Figure 4.38. Many of the north-facing sides of the buildings are indicated as visible in each of these views, confirming the hypothesis that the predicted image locations should come from the half of the circle that was eliminated in the 3D reconstruction. Note that even if the predicted view point was available, adding it to the 3D workflow may not improve the reconstruction as it may not be tied to the current imagery in a way that features can be reliably tracked. It is up to the user to make this connection when planning flight lines.

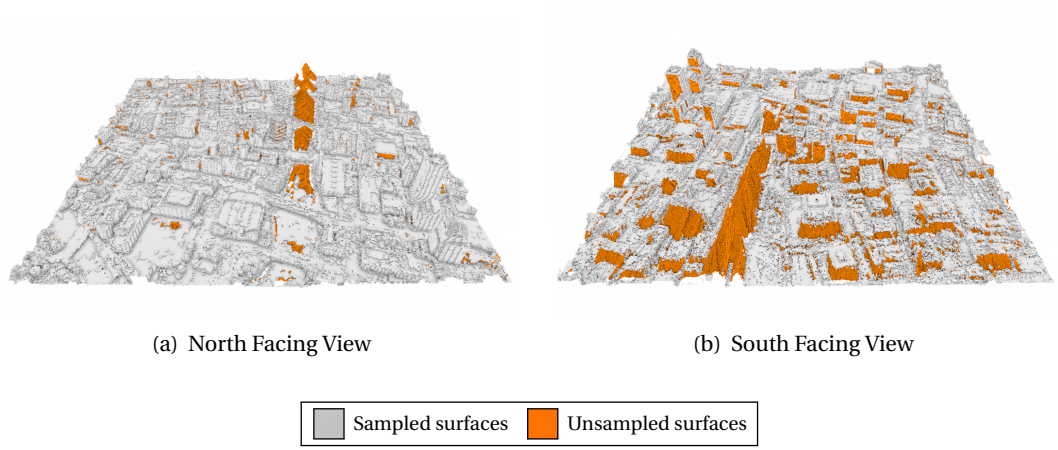


(a) Nadir View



(b) South-East Facing View

*Figure 4.34: Point cloud computed using the 3D workflow, where the input imagery was from the WAMI sensor, but was limited to half a revolution so that major vantage points necessary to complete a surface model were guaranteed to be missing. The blue background was used to highlight the holes in the point cloud.*




(a) North Facing View (b) South Facing View

*Figure 4.35: Voxel space computed from the point cloud shown in Figure 4.34, using 2m voxels, where all voxels with points are considered occupied. North and south facing views are presented to illustrate the effect that only using half a revolution of the WAMI imagery has on the point cloud. Note that most of the views that were included were north facing views, so there are more sampled surfaces when viewing from that direction.*

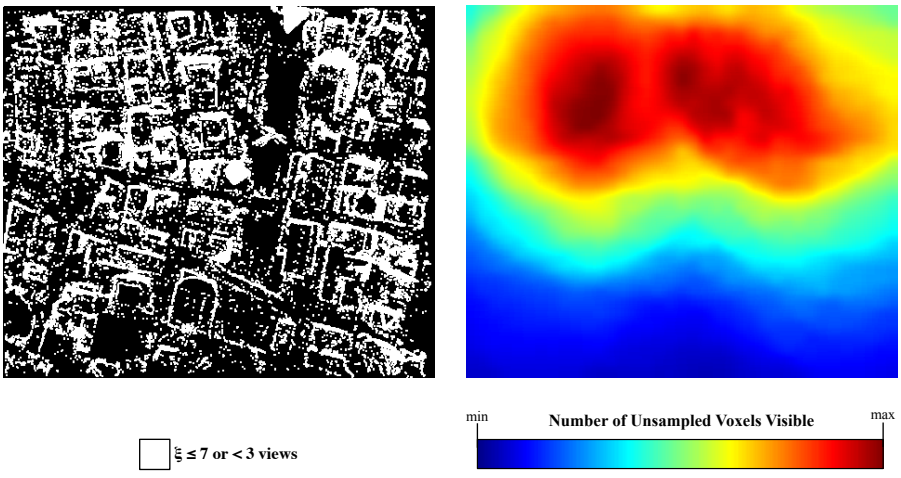



*Figure 4.36: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras, in addition to unsampled voxels that were seen by 3 or more cameras and had a standard deviation of greater than 7.0 in (a), where the standard deviation window size was 100x100. Right: A corresponding heat map of future aircraft image locations, where a nadir pointing angle and aircraft altitude of 10,000ft have been specified.*

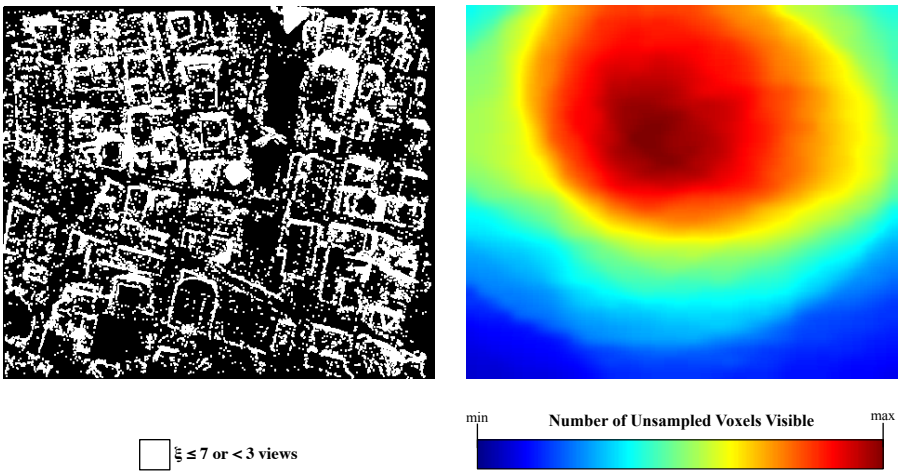
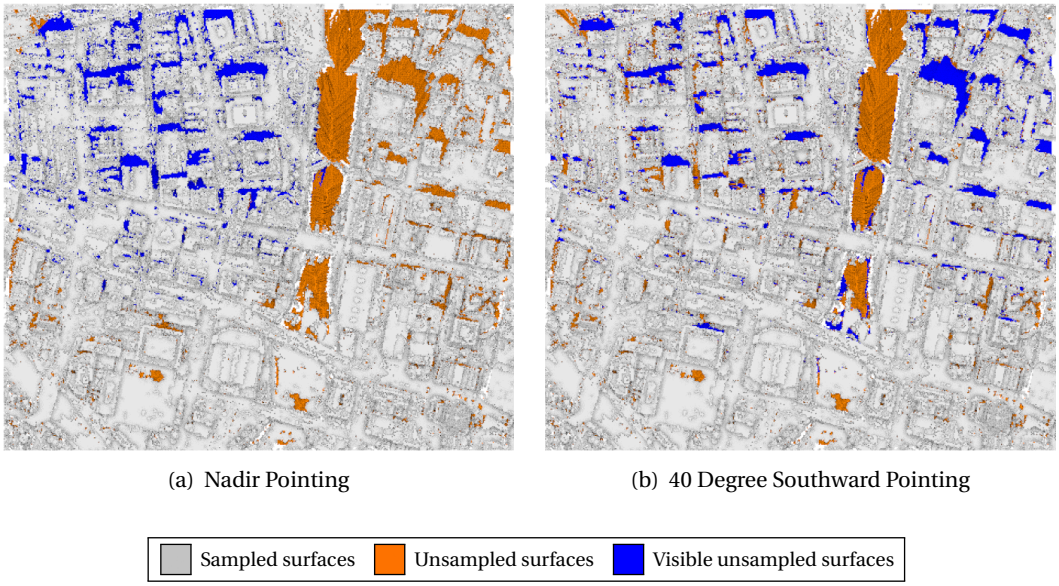



*Figure 4.37: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras, in addition to unsampled voxels that were seen by 3 or more cameras and had a standard deviation of greater than 7.0 in (a), where the standard deviation window size was 100x100. Right: A corresponding heat map of future aircraft image locations, where a southward 40 degree off-nadir pointing angle and aircraft altitude of 10,000ft have been specified.*




(a) Nadir Pointing (b) 40 Degree Southward Pointing

*Figure 4.38: Left: View of the voxel space where voxels that were visible from the indicated location using a nadir pointing angle are shown in blue. Right: View of the voxel space where voxels that were visible from the indicated location using a southward 40 degree off-nadir pointing angle are shown in blue. Voxels shown in orange are indicative of unsampled surfaces that were either not visible or did not meet the given texture criteria.*

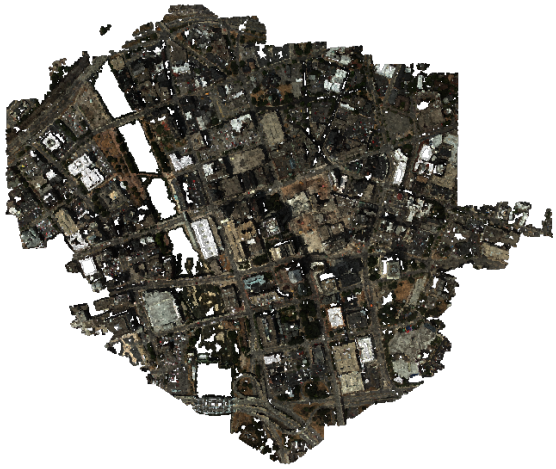


*Figure 4.39: Point cloud of downtown Rochester, NY, generated with 23 images from the WASP dataset.*

## 4.5 Additional Datasets

Up to this point, only point clouds derived from the Exelis WAMI data have been presented. Additional datasets were tested to gain insight into the voxel processing to better understand the conditions in which it performs well, and those in which it is likely to fail. More information on the datasets is available in Appendix A.

### 4.5.1 WASP: Downtown Rochester, NY Dataset

In addition to the WAMI imagery of downtown Rochester, NY, collections of WASP imagery have also been taken. The WASP sensor is a color sensor, and imagery is collected at nadir. Though the scene content is the same, the nadir viewpoint is different and could potentially impact the creation of the voxel space and future image location identification and thus it was important to study. A point cloud was generated using the 3D workflow and 23 WASP images, and the point cloud results are shown in Figure 4.39. The major difference in this point cloud is the lack of points on building sides, as buildings are defined only by their rooftops. This is because the nadir view does not often capture the sides of buildings, and what is not visible in the imagery cannot be reconstructed. Additionally, the extent of the point cloud is smaller than that of the WAMI cloud, and the results are slightly noisier.

A voxel space was created using the WASP point cloud, where the voxels were approximately 3m in size. The result is shown in Figure 4.40, where the point cloud has been shown for reference. Note that from overhead, most of the visible surfaces in the voxel space are sampled, as these surfaces were visible in the imagery. In the side view, it is evident that not many points in the point cloud fell on the sides of buildings, and thus those surfaces remain unsampled, though the ray tracing appears to have done a good job carving out the free space around the buildings in the scene. The "pyramid" structures that existed in the areas of voids in the WAMI point cloud are also evident in the WASP data, however they are much larger in this case, and this is a direct result of the nadir collection angle. The field-of-view of the WASP sensor is at most $\pm 18$ degrees off nadir, resulting in rays that are nearly perpendicular to the ground surface that do not intersect to the same extent as those from an off-nadir collection, therefore more unsampled space results above a void. This is particularly evident in the river, where large voids in the point cloud exist. The result here is unsampled surfaces that extend in the direction of the camera, surrounding the void.

This becomes problematic when determining the visibility of the unsampled surfaces. Because of the way that the ray-tracing carves out the free space with the nadir view angle, the surfaces on the free-unsampled boundary surrounding the voids are perpendicular to the camera viewpoints due to the discrete nature of the surfaces in the voxel space. When performing the visibility analysis for the unsampled voxel faces that surround the void, the perpendicularity indicates that the surface was not visible due to grazing angles. This is independent of any standard deviation threshold that is set. Because the surfaces were indicated as not visible, these surfaces are included in the future image location identification stage, and depending on the size of the voids can significantly influence the results.

Results from the future image locations for a southward pointing sensor, 20 degrees off-nadir, are shown in Figure 4.41. The pointing angle was chosen such that some building sides would be visible. Only unsampled voxels that were seen by less than three cameras were used in the computation; this included voxels on the sides of buildings as well as many of the vertical voxels above the river area. The threshold was set for the visibility of the unsampled voxels, because the texture metric was not shown to be more accurate at removing regions and using just the visibility reduced the computational load. A value of 3 was chosen because unsampled voxels that were visible in fewer than 3 views are guaranteed to be a result of lack of coverage in the input imagery due to the nature of the 3D reconstruction algorithms used. The voxels that

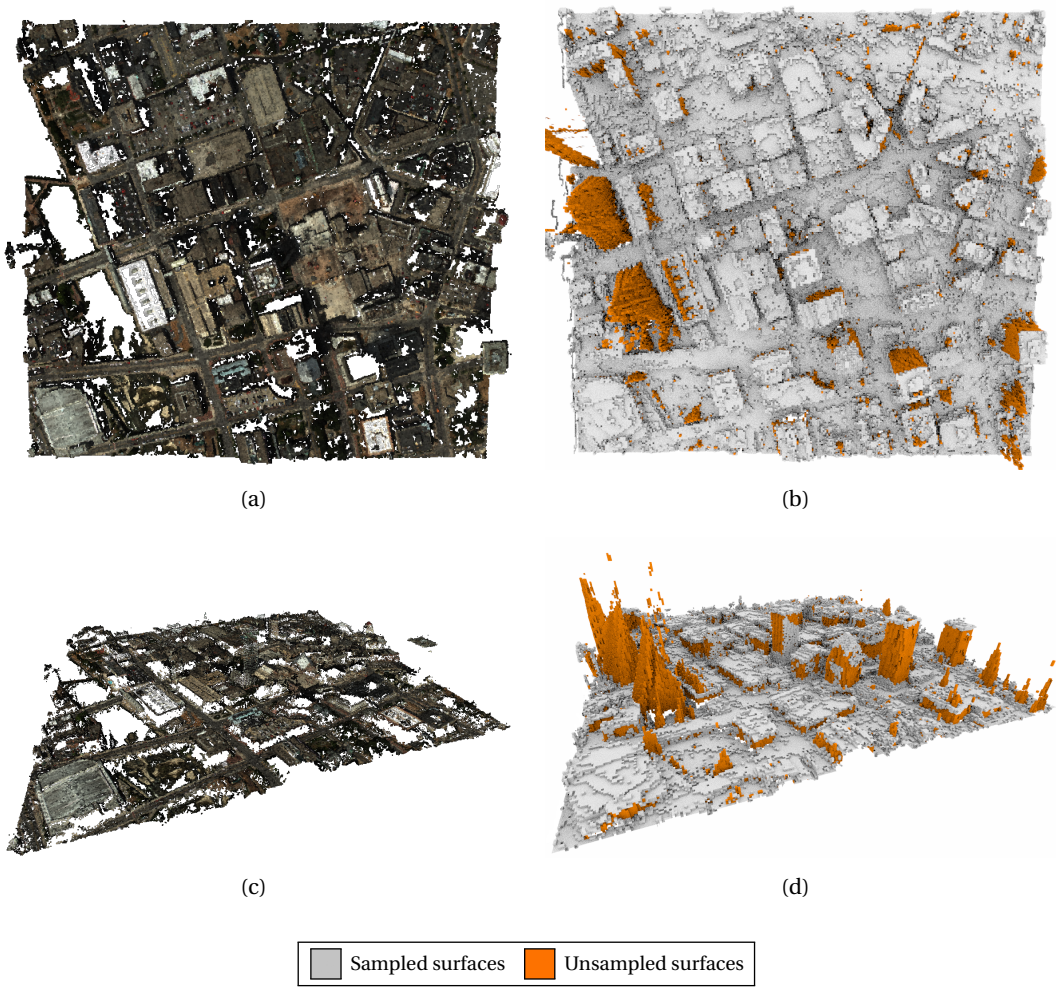



*Figure 4.40: The voxel workflow was used to create a voxel space using a point cloud derived from nadir WASP imagery; voxels are approximately 3m. The WASP point cloud is shown in (a) and (c) for reference, and the voxel space is shown from overhead in (b) and the side in (d).*

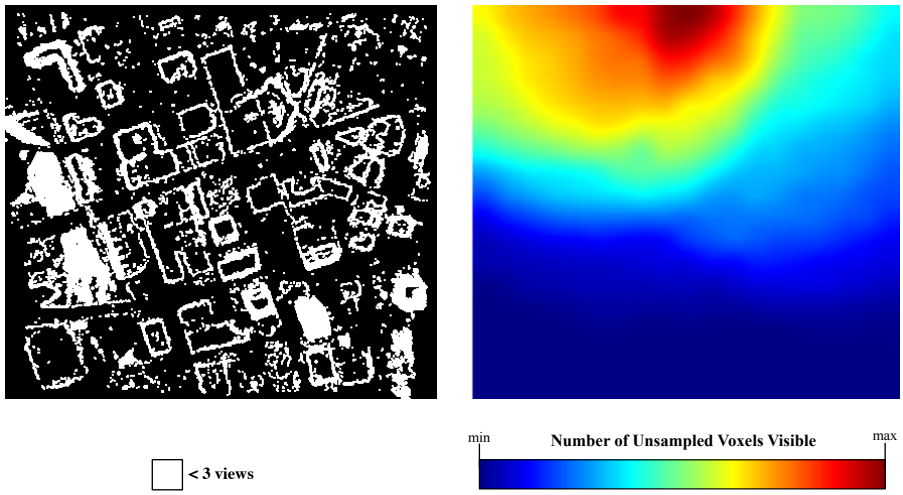



*Figure 4.41: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras. Right: A corresponding heat map of future aircraft image locations, where a southward 20 degree off-nadir pointing angle and aircraft altitude of 5,000ft have been specified.*

would be visible from the indicated location are highlighted in Figure 4.42. While it is indicated that many of the sides of buildings would be visible from this location and pointing angle, there are also numerous voxels in the river structure that are identified as visible. Since there are so many unsampled voxels surrounding the river void, it has a significant influence on the predicted location. In this case, there were a sufficient number of unsampled voxels elsewhere in the scene, and the predicted view does include many desirable unsampled surfaces.

### 4.5.2 WASP: Quarry Dataset

As part of RIT's SHARE 2012 campaign [87], the WASP sensor collected imagery over a large rock quarry near Honeoye Falls, NY. More information on the WASP sensor and the datasets is available in Appendix A. The quarry presents a unique target for 3D reconstruction because it is mostly rural, and unlike other scenes the 3D geometry is below the ground level, protruding into the earth rather than extending above it. A point cloud was derived using WASP imagery of the quarry, collected at nadir; the point cloud is shown in Figure 4.43. The central region of the quarry has some significant voids, likely due to textural difficulties in the matching process in regions covered by dense vegetation. The walls of the quarry are also not defined in the point

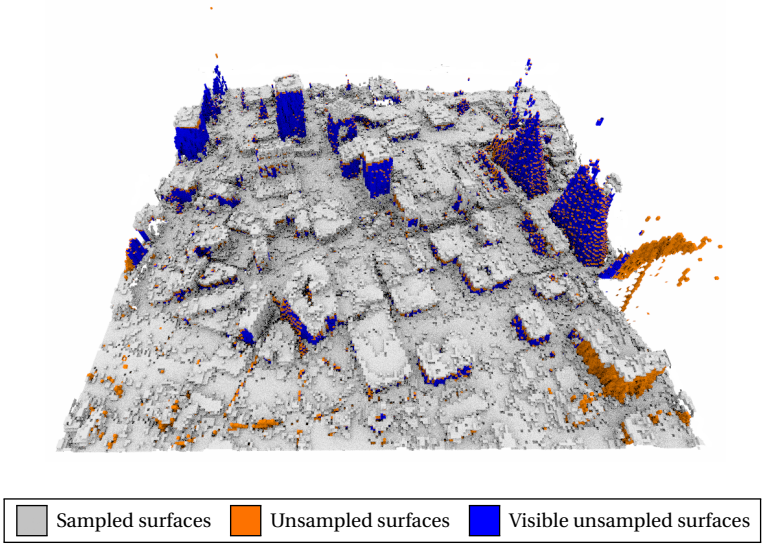



*Figure 4.42: View of the voxel space where voxels that were visible from the indicated location using a southward 20 degree off-nadir pointing angle are shown in blue. Voxels shown in orange are indicative of unsampled surfaces that were either not visible or were visible in three or more camera frames.*

cloud because they were not visible in the imagery, or were visible at a very sharp angle.

A voxel space was created using 2.25m voxels, and the result is shown in Figure 4.44. The sampled surfaces were shown alone to give some context for the shape of the unsampled surfaces. The sampled surfaces of the quarry accurately represent the structure that was derived using the point cloud. The biggest issue with the voxel space shown here is the unsampled surfaces, which create tall wall-like shapes around the voids. This effect is also evident around the actual wall of the quarry, where the unsampled surfaces in the void regions extend from the lower ground plane on the floor of the quarry to far above the upper ground plane. These effects are the result of the nadir angle of the cameras used to generate the scene. The ray-tracing to the points does not carve out the surrounding free space, leaving vertical columns of unsampled voxels. These regions cannot be eliminated with the texture analysis because the vertical surfaces are not indicated as being visible in any of the images due to the nadir pointing angle, as was seen with the previous nadir-derived dataset. Because these surfaces far outnumber the actual unsampled surfaces of interest on the quarry wall, they dominate the future image identification stage.

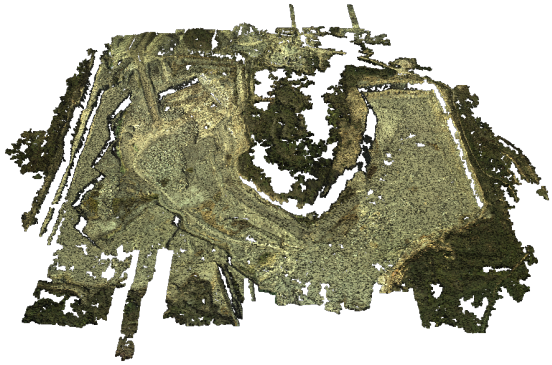
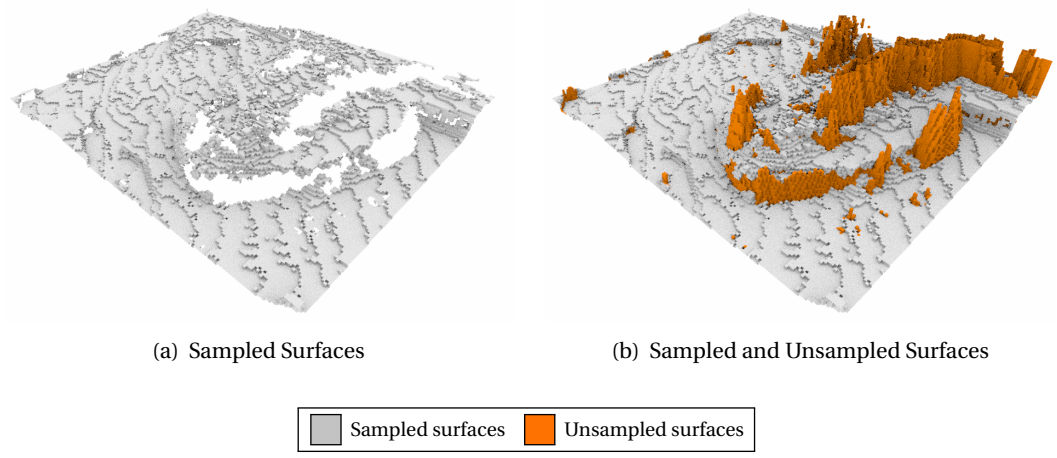


*Figure 4.43: Point cloud of the quarry, generated with WASP imagery that was collected as part of the SHARE 2012 campaign [87].*




(a) Sampled Surfaces (b) Sampled and Unsampled Surfaces

*Figure 4.44: Quarry voxel space, created with a set of WASP images and 2.25m voxels.*

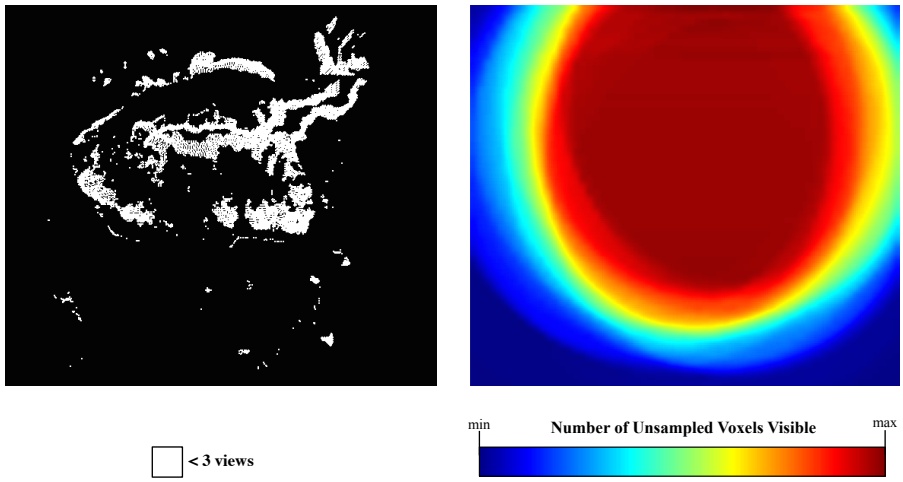



*Figure 4.45: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras. Right: A corresponding heat map of future aircraft image locations, where a southward 40 degree off-nadir pointing angle and aircraft altitude of 5,000ft have been specified.*

Results from the future image locations for a southward pointing sensor 40 degrees off-nadir that would capture the walls below ground in the quarry, are shown in Figure 4.45. Again, only unsampled voxels that were seen by less than three cameras were used in the computation. The unsampled voxels were dominated by the false surfaces surrounding voids in the point clouds that were a result of textureless regions. This is evident in the view of the voxel space, showing the voxels that would be visible from the indicated location in Figure 4.46. The quarry wall is also identified as visible, but this is likely a result of its placement relative to the other unsampled voxels. In this case, the undesired sampled surfaces dominated the image location identification stage.

It may be possible to implement a process to identify these regions in nadir point clouds so that they do not influence the future image location identification stage. Hagstrom [88], who derived voxel spaces using LIDAR data, noted that when the voxel resolution is near its limit, there are often small and random unsampled voxels in the surrounding open space. A noise removal process was implemented to deal with this effect, where unsampled voxels above the highest surface in a column were changed to be free space. If no surface was present, the closest surface level from a nearby column was used instead. This would need to be adapted to handle SfM point clouds, where much larger voids are common.

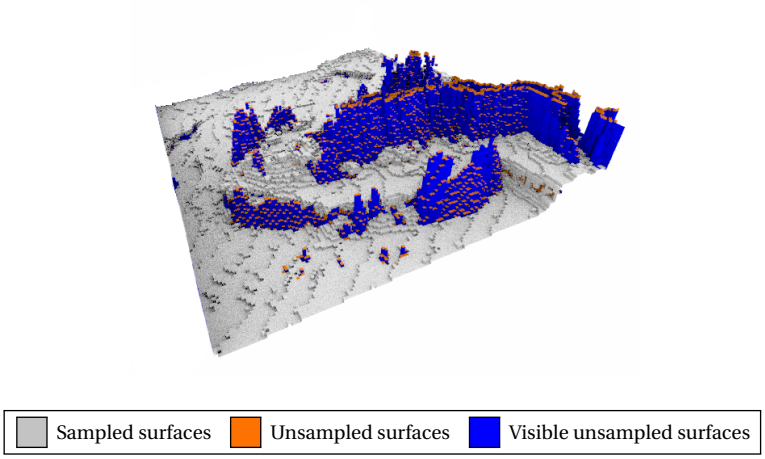



*Figure 4.46: View of the voxel space where voxels that were visible from the indicated location using a southward 40 degree off-nadir pointing angle are shown in blue. Voxels shown in orange are indicative of unsampled surfaces that were either not visible or were visible in three or more camera frames.*

While goals of the modern computing era are often to automate everything, it should be noted that manual removal of such regions would be easier than automating this, and the cost may make that worthwhile. By allowing a user to identify regions in which the large walls are present, the problematic areas could be removed from consideration and the predicted image locations would see immediate improvement. This could be implemented in such a way that only a few clicks are required for user input, and warrants investigation in future work.

### 4.5.3 CorvusEye: Downtown Dataset

The final set of imagery that was tested was another downtown Rochester, NY dataset, collected with Exelis' CorvusEye sensor. The CorvusEye sensor is a newer sensor developed by Exelis that is another wide area motion imagery sensor, similar to the original WAMI sensor, but it provides RGB imagery instead of panchromatic. More information and sample images from the CorvusEye sensor can be found in Appendix A. The dataset that was used was collected at 10,000ft with a fixed stare point, flying in a circular motion in a persistent surveillance mode. A set of 10 images were used to develop a point cloud, covering less than a quarter rotation of the circular flight path. The resulting point cloud is shown in Figure 4.47.

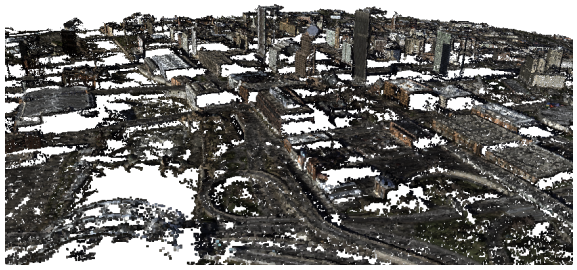
The images that were used in the reconstruction were collected with an eastward pointing

*Figure 4.47: Point cloud of downtown Rochester, NY, generated with 10 images from the CorvusEye dataset.*

angle, so when viewing the point cloud from the east the buildings are well sampled and the results look very dense. However, other vantage points were missing and this left significant voids in the point cloud which are evident when looking from other angles. Side views of the point cloud are shown in Figure 4.48. As can be seen, rooftops and westward facing sides of buildings were reconstructed, but the remaining three sides of the buildings were not reconstructed in most cases. Though similar to the WAMI point cloud, this presented a slightly different test case for the voxel-based workflow.

A voxel space was created using a central subset of the scene using 2m voxel resolution. The result is shown in Figure 4.49, where the voxel space is shown from the side with a north looking view. This was to highlight the unsampled surfaces surrounding the river as well as the unsampled shadows that surrounded the buildings. The shadows on the buildings were expected, as they were seen in both of the previous WAMI point clouds derived from oblique imagery, and their presence was a result of the ray tracing as shown in Figure 3.12. The structure in the river however is different than that seen previously. In the WAMI point clouds and resulting voxel spaces, the large homogeneous regions of the river had pyramid-like structures of unsampled surfaces in the voxel space, as explained by Figure 4.10. In this case, because the views used

(a) East Facing View



(b) North Facing View

*Figure 4.48: Point cloud computed using the 3D workflow, where the input imagery was from the CorvusEye sensor, but was limited to approximately a quarter revolution so that major vantage points necessary to complete a surface model were guaranteed to be missing.*

to reconstruct the CorvusEye point cloud were all from similar directions, there were no rays coming from different vantage points to sample the free space. This resulted in a large region of free space extending from the river to the top of the voxel space in the direction of the cameras. These large regions of unsampled voxels are similar to those that were seen in the nadir WASP datasets, both in the downtown and quarry regions. If not properly removed, they will significantly impact the results due to the large number of unsampled surfaces that compose the regions.

The unsampled surfaces that were seen by more than three cameras were suppressed from consideration in the future image identification stage. This process did remove some of the surfaces in those river regions, but did not remove all of them which would be ideal. In the nadir derived voxel spaces from the WASP data, the unsampled surfaces were not removed properly due to the grazing angle threshold. In this case, it is more likely that the surfaces were not removed because the ray-tracing encountered unsampled voxels along the path and

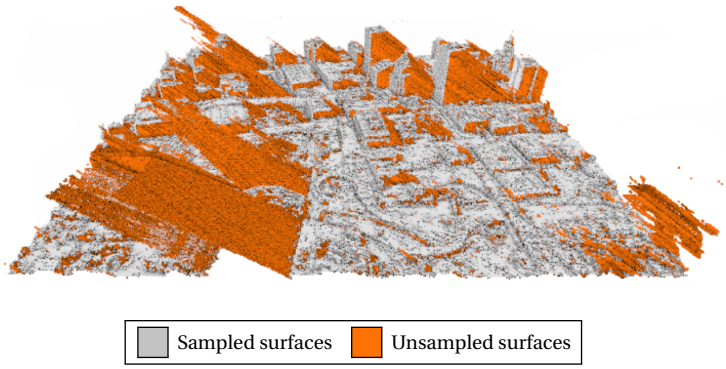



*Figure 4.49: CorvusEye Downtown voxel space, created with a set of 10 CorvusEye images and 2m voxels, where the view is looking North.*

the unsampled voxels are assumed to be opaque because it is not known whether or not they contain surfaces. Future image location maps were computed for a nadir pointing angle and a westward pointing angle, 40 degrees off-nadir; the resulting maps are shown in Figures 4.50 and 4.51 respectively.

In each case, the predicted location is centrally located such that the entire footprint of the sensor would have fallen on the voxel area, and the number of unsampled voxels visible falls off toward the edges of the map. From the maps themselves, it is difficult to determine if the predicted views are correct. The scene is dominated by unsampled voxels that compose building shadows, and these regions will be visible from the predicted locations, as shown in Figure 4.52.

Because there were so many unsampled voxels in the scene that were true unsampled surfaces, the predicted image locations using either pointing angle produce views in which a large number of these unsampled voxel surfaces would be visible. Had the scene been composed of more false unsampled surfaces, as was the case with the Quarry dataset, the result may have been different. The results from the CorvusEye point cloud, in addition to those generated with the WASP point clouds, indicate that more testing of the voxel-based workflow is necessary in future work.

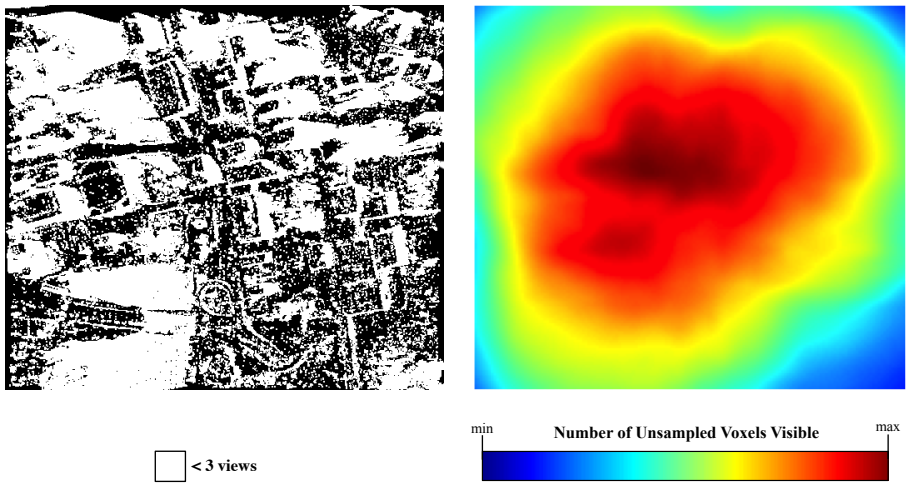



*Figure 4.50: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras. Right: A corresponding heat map of future aircraft image locations, where a nadir pointing angle and aircraft altitude of 10,000ft have been specified.*

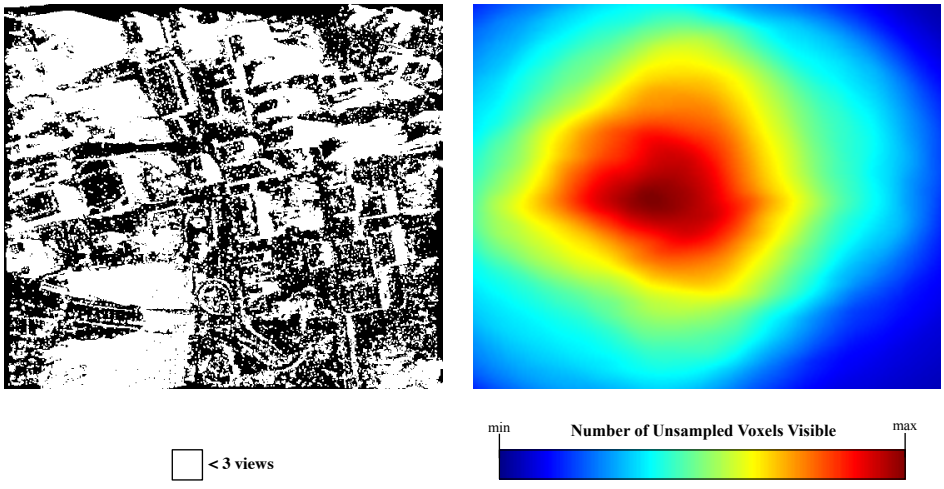
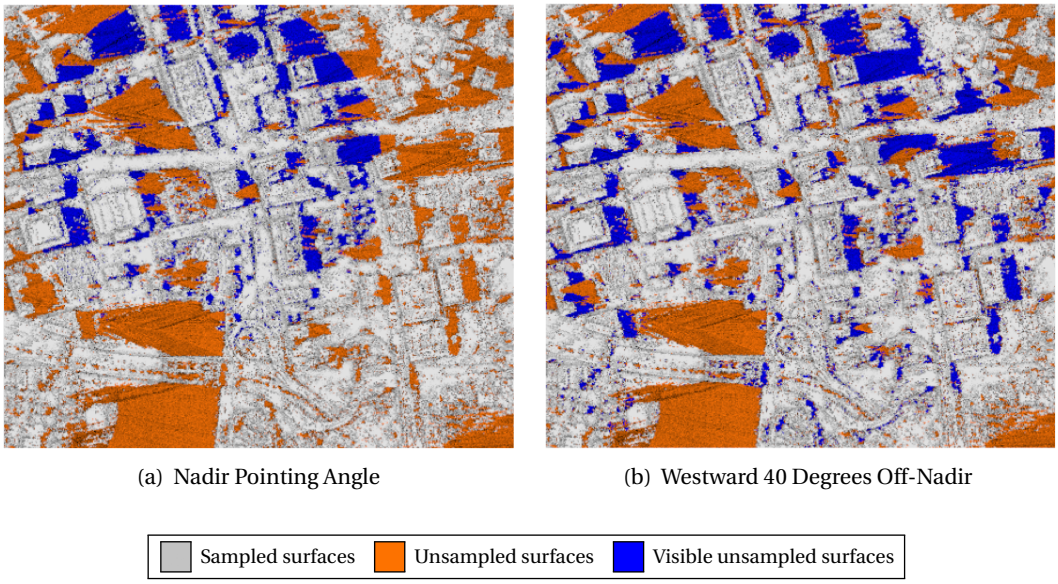



*Figure 4.51: Left: Unsampled voxels at free-unsampled boundaries that were seen by less than three cameras. Right: A corresponding heat map of future aircraft image locations, where a westward 40 degree off-nadir pointing angle and aircraft altitude of 10,000ft have been specified.*




(a) Nadir Pointing Angle

(b) Westward 40 Degrees Off-Nadir

*Figure 4.52: View of the voxel space where voxels that were visible from the indicated location using a nadir pointing angle (a) and using a westward 40 degree off-nadir pointing angle(b) are shown in blue. Voxels shown in orange are indicative of unsampled surfaces that were either not visible or were visible in three or more camera frames.*

❄ ❄ ❄

The results and analysis presented here confirms that the voxel-based workflow can successfully predict locations from which unsampled surfaces are visible. However, tests with the additional datasets indicate that future work may be necessary to fine-tune the workflow such that it is capable of handling other types of data. It is possible that large unsampled voxel regions that are obvious errors due to ray-tracing will need to be removed based on user input prior to predicting an image location, as this could significantly improve the results of the predicted lcoations. Despite that, the voxel-based analysis has been shown to capture the concept of free space in a way that has not been done, providing more information about the reconstruction than was previously available in the original point cloud format.

# Chapter 5

# Conclusions

Voids in point clouds derived from multiple-view imagery exist as a result of texturally difficult areas, occlusion of portions of the scene, and a lack of coverage in the input imagery. Traditional surface reconstruction and hole filling algorithms are not well suited for SfM derived point clouds of city-wide scenes, often resulting in holes in the model or poor estimations of the surfaces in the void areas. The is due to the fact that most of the voids are complex, and are composed of multiple intersecting surfaces. In cases where voids are a result of a lack of coverage, the reconstructions will benefit from the inclusion of more imagery such that more points will be reconstructed in the void areas. A voxel-based approach has been presented to partition the 3D volume, taking advantage of the idea of free space, such that voids can be easily identified. Multiple line-of sight analyses are then used to exploit the information contained in the voxel space such that voids resulting from texturally difficult areas can be distinguished and potential future image locations that have a clear view of the voids can be extracted. The biggest strength of the voxel workflow is its ability to capitalize on the idea of free space and use that to build a voxel model. The workflow excels when using point clouds generated from oblique imagery with properly sized voxels, as the free space is adequately carved out, and future image locations are easily identified using a ray tracing analysis.

## 5.1 Voxel-Based Visibility Analysis

While identifying voids in a 3D point cloud can be challenging, converting the point cloud to a voxel space provides a simple method to locate voids as they are manifested in the voxel space as unsampled voxels. The voxel space generation algorithm was tested on both real and synthetic datasets. The ground truth DIRSIG dataset was used to verify the results of the voxel workflow to confirm it was performing as expected. The synthetic truth data made it possible to generate voxel spaces from a single camera vantage point (something that would not be possible using a traditional 3D workflow). The truth data made it possible to test the voxel space on a more intuitive basis, resulting in confidence in the process used to generate the voxel spaces.

A voxel space was then generated from a 3D reconstruction using real-world imagery from the WAMI dataset, and the process behaved as expected. Using the WAMI data, the effect of resolution on the voxel space was examined by generating multiple voxel spaces using the same input point cloud at different voxel resolutions. The results of this indicated that smaller is not better in regards to voxel size, as there is some minimum resolution that the point cloud can support. As the voxel size is decreased past this resolution, spurious voxels begin to appear and the model is lost. Additionally, as the voxels grow in size, some detail information on the surfaces of the reconstruction is lost. The voxel size is crucial to subsequent steps in the voxel workflow, as using a voxel size too small will result in undersampling the voxel space and an influx of unsampled surfaces will be present. Conversely, a voxel size that is too large will eliminate fine details in the scene, but this may not be as critical depending on the application. At this time, it is up to the user to define the proper voxel size as there is no straightforward method to automate the process.

A similar analysis was performed to test the effect of the probability of free space threshold, used to classify the voxels into the ternary system: occupied, free, and unsampled. Unsampled voxels in the space are not affected by the probability of free space threshold, because they contain neither points nor rays. The majority of the voxels are either unsampled or only contain ray interactions. This is due to the fact that voxels with point interactions define a single surface, and the majority of voxels lie underneath this surface (unsampled) or above this surface (free). As the threshold used for the probability of free space decreases, the number of unsampled surfaces in the voxel space increases. The effect of this was studied visually and it was noted

that many of the sampled surfaces that are lost as a result of the reduction in threshold are converted to unsampled surfaces without much effect on the shape of the model. The increase in unsampled surfaces would have a negative impact on the future image location identification, and therefore it was determined that all voxels with point interactions would be classified as occupied for the purposes of this work. It is possible that this may be a limitation in point clouds with excessive noise, but the effect could be mitigated by using a noise reduction filter as part of pre-processing before creating the voxel space.

In areas with large voids, such as the river or buildings in which only one side was reconstructed, shadowing effects were noted in the unsampled voxels. Due to the lack of points, the free space in these regions was not sufficiently carved out, and the oblique nature of the rays associated with the WAMI dataset resulted in a pyramid like structure in the river, and an apparent shadow on buildings. This effect was also present in the nadir imagery from the WASP sensor that was tested, however the representation in voxel space was different due to the angles and nature of the ray tracing. In the nadir imagery, holes resulted in taller pyramid like structures for small holes, and large holes resulted in more of a wall effect. This became problematic in both the void identification and identification of future image locations calculated on some of the nadir datasets.

## 5.2 Void Identification

It was noted that the boundary voxels in the voxel space contain the most pertinent information, and those voxels that lie on the free-unsampled boundary are representative of the unsampled surfaces in the model. By identifying these voxels, a preliminary estimate of areas of the point cloud that contain voids is obtained, and gives a basis for computation of the future image location mapping. However, many of the unsampled voxels have already been imaged by multiple cameras and thus they are not likely to benefit from the inclusion of more imagery in the 3D workflow. These types of voids exist in texturally difficult areas that failed to generate feature correspondences, and it is important to distinguish these voids from those that are a result of a lack of coverage, as including texturally difficult areas could negatively impact the prediction of future image locations.

A line-of-sight analysis was used to determine the visibility of the unsampled voxels from the cameras used in the reconstruction. This is necessary as there is no visibility information

associated with the unsampled voxels because by definition they lack points. Next, the unsampled voxels were reprojected into the original images in which they were visible so that the local image texture could be analyzed. Two texture metrics were tested: a local average image gradient and a local standard deviation. Local standard deviation was chosen as the texture metric because it identified potentially homogeneous regions and was computationally efficient. The local standard deviation metric is affected by the choice of window size and threshold, and this can change the number of unsampled voxels in the voxel space that are used to predict future image locations. Unsampled voxels that were visible in less than three views were always included, regardless of their local standard deviation metric. This is because parameters used in the 3D workflow require 3 views for reconstruction, and therefore those voxels with less than 3 views are guaranteed not to be reconstructed.

The WAMI point cloud was dominated by unsampled voxels that were a result of lack of coverage. While different window sizes and thresholds were tested, it was found that as long as the threshold removed most of the unsampled regions, the prediction of the optimal image location was unaffected. It was also determined that a threshold on the number of views with good visibility could also be used to eliminate texturally difficult regions. When looking for voids that are due to a lack of coverage, the visibility information may be sufficient, without the need to compute a texture metric. The visibility is computed for each unsampled voxel face prior to computing the texture metric, so this could potentially reduce computational efforts. Additionally, using a visibility threshold can catch voids that are a result of spatially repetitive textures, in addition to the homogeneous regions indicated by the current texture metric.

There were some cases where the void identification and suppression performed poorly, thus causing the future image location stage to fail. This was particularly evident in nadir point clouds with large voids, because the ray tracing resulted in “walls” of unsampled voxels surrounding the voids that extended to the top of the voxel space. The unsampled voxel faces that composed these walls were determined to be not visible in any of the cameras, due to the fact that the surface normals were perpendicular to the camera viewing angle and were indicated to be grazing angles. Because they were indicated as “not visible” the texture metric had no effect and did not suppress them. As a result, these unsampled voxels were included in the future image location stage, and results were heavily weighted by these areas due to the large extent. This phenomena was also seen in some of the oblique point clouds, depending on which images were included in the reconstruction. This emphasizes the importance of properly carving

out the free space surrounding the surfaces, because failure can happen in subsequent stages.

## 5.3 Future Image Location Identification

Finally, another line-of-sight analysis was used to create a map of potential future image locations. These maps indicate how many of the voxels on the free-unsampled boundary (that are not suppressed in the previous step) could potentially be imaged from each aircraft position with a particular pointing angle. Areas that could see the most unsampled voxels were identified as potential image locations, though a full map over the area was provided. Current results constrain the maps to two degrees-of-freedom, by holding the aircraft altitude constant, using a circular camera field-of-view, and by holding either the pointing angle constant or defining a fixed stare point. For each location and defined pointing angle, the number of unsampled voxels that are visible is computed, weighted by a uniform or Gaussian weighting function with defined cutoffs to exclude grazing angles; this is the cost function used in the analysis. The 25m resolution presented here was very fine, and the spacing could be increased to reduce computation time without significantly influencing the results.

The predicted future image location with the maximum number of unsampled voxel faces visible was constant for a variety of combinations of window size and threshold (parameters of the texture metric) for the downtown WAMI dataset. This is because the scene was dominated by voxels that were seen by fewer than three views, and these voxels are included regardless of the texture metric. In cases where the voids are not dominated by voxels that are due to a lack of coverage, the voxel maps will differ.

Three different sensor configurations were tested: (1) fixed pointing at nadir, (2) fixed pointing off-nadir, and (3) fixed stare-point, in which the stare point is centrally located in the scene. In addition, these were tested with both uniform and Gaussian weighting functions. Both the nadir and off-nadir fixed pointing systems behaved as expected, predicting viewpoints in which unsampled voxels were visible. In each case, it was noted that the weighting functions had a minimal effect due to the distribution of the surface normals in the voxel space and the field-of-view of the sensor. The fixed stare-point system did not perform as well. Due to the nature of the cost function trying to maximize the number of unsampled voxel faces that are visible, the fixed stare-point system always pushed for steeper angles. This is because the circular field-of-view of the camera is larger on the ground at steeper angles, and therefore has the possibility

of imaging more unsampled voxels. As such, it is recommended that the fixed stare-point be used with a fixed radius or a radial weighting function, so the comparisons between the area imaged on the ground are similar. This was the only case in which the weighting function had a significant effect, and that is a result of the changing pointing angle across the field. It was not evident from the results whether one weighting function is better than another.

Using the camera location and pointing angle with the predicted best viewpoint, the voxels that were visible from that viewpoint were computed and displayed for visual analysis. In the case with the WAMI point cloud, derived from oblique imagery, each of the predicted views captured unsampled surfaces of interest in the scene, confirming that the predicted location could potentially be used to reduce the number of voids in a 3D reconstruction. Additional datasets were tested as well, and in each case unsampled voxel faces were indicated to be visible from the predicted location. As mentioned previously, in some cases these were voxels that should have been suppressed because they surrounded regions that were void due to texturally difficult areas. In the case where such voxels dominate the scene, some sort of manual intervention may be necessary to indicate and remove these false unsampled surfaces.

While potential imaging locations were identified, no subsequent analysis was performed to determine whether inclusion of these images resulted in a superior reconstruction, due to limitations in being able to enforce inclusion of specific imagery in the 3D workflow. In identifying the future image locations, there is no consideration of past image locations (those used in the original 3D reconstruction), thus there is no guarantee that the predicted location will be easily tied to the 3D reconstruction.

## 5.4 Limitations

This workflow does have several limitations and it would be remiss not to acknoweledge them. Creation of the voxel space is dependent on the input point cloud provided, and as such, the results obtained can only be as good as the input data. A noisy point cloud will result in a similarly noisy voxel space. The user defined voxel size is crucial to the creation of a voxel space. Using a voxel size not well-suited to the input point cloud may result in an influx of unsampled surfaces due to undersampling the voxel space, which in turn would negatively impact the results of the void identification and future image mapping.

The voxel workflow is more likely to fail when using point clouds derived from nadir im-

agery, particularly those that have large voids that are enclosed in the scene. This is because the ray tracing does not adequately carve out the free space, because of the lack of information in the region, and the result is large wall-like structures that surround the void. Because these unsampled voxels are perpendicular to the nadir view, the visibility analysis identifies them as not visible in the imagery, due to grazing angles, and as a result they are not removed in the textureless feature removal. If these unsampled voxels make up a majority of the unsampled voxels in the scene, as was the case with the WASP quarry data, they will heavily influence the future image location identification, and this effect is undesirable. In scenes where these voxels compose a smaller portion of the unsampled voxels, such as the WASP downtown data, the future image location identification may still identify desired viewpoints. Manual removal of these regions based on user defined input could significantly improve the results.

Finally, the biggest limitation is that the predicted future image location from which the most voxels are visible is not guaranteed to be tied in any way to the current 3D reconstruction. From the half circle of the WAMI data, it is evident that the predicted viewpoint would be vastly different from those included in the original reconstruction. The user will have to find a way to tie the predicted location to the imagery that has already been used so that features can be reliably matched, thereby increasing the odds for an improved reconstruction. This would be done by including intermediary imagery between the previous locations and the predicted maximum location. It is possible that the provided maps could be used to do this so as to maximize the unsampled voxels that are seen on the chosen path.

# Chapter 6

# Future Work

There are many areas within this work that could be expanded and improved upon for future voxel-based analysis of point clouds. Listed here are some improvements to the processing methods that were not implemented in the current voxel workflow, as well as other potential applications.

## 6.1 Generating Point Clouds

The results of the voxel-based workflow presented here are directly related to the quality of the input point cloud(s) used to generate the voxel space. The area of 3D reconstruction from imagery is rapidly expanding, and the techniques used here to generate point clouds (Section 2.1.7) are admittedly no longer state-of-the-art. Recent innovations may be able to provide better point cloud reconstructions for input to the voxel-based workflow. There are three areas that show potential for immediate exploration: feature extraction, densification, and geo-accurate transformation.

A 3D reconstruction from imagery is dependent on the initial feature extraction to estimate the camera pose. While SIFT is a widely used feature detection and extraction scheme, there are numerous other feature detectors and descriptors that should be explored. Some possibilities include: SURF (Speeded Up Robust Features) [8], FAST (Features from Accelerated Segment Test) [9], MSER (Maximally Stable Extremal Regions) [10], GLOH (Gradient Location and Orientation Histogram) [11], DAISY [12], and ASIFT or Affine-SIFT [13].

In addition to the initial feature extraction, the densification process is also something that could change. While CMVS/PMVS are widely used, probabilistic voxel modeling [22] and semi-global image matching (SGM) [23] have also shown promise for point cloud densification. The only criteria for the voxel-based workflow is that the visibility of the points (i.e. the cameras that were used to reconstruct a point) are available in addition to the points themselves. Without the visibility information, the free voxels in the voxel space cannot be identified. The sparse point clouds resulting from bundle adjustment could also be used in the voxel-based process, depending on the application since the density of the point cloud has a direct impact on the voxel resolution. Improving upon and/or changing the densification process could also eliminate some of the randomness that is associated with the current 3D workflow, and that could be beneficial in further testing and validating the voxel-based process.

Lastly, the geo-accurate transformation used in the current workflow is not the most accurate of the current transformations available, as discussed in Section 2.1.4. While a geo-accurate transform is not a requirement of the process, having the world-coordinate axes and the point cloud coordinate axes differ by a rotation can make visualization of the voxel space a challenge, and the future image location processing assumes that the $XY$ plane in the voxel space is parallel to the plane in which the future image positions are located. Additionally, if the objective is to determine optimal positions for new image collection, then the accuracy of the voxel space becomes important. Improvements to the geo-accuracy of the point cloud will be reflected in the voxel space, and could further improve or refine the predicted optimal imaging locations. Without a geo-accurate point cloud, a principle component analysis could be used to obtain the three dominant orientations to align the ground plane of the point cloud to the $X-Y$ axis in the voxel space.

## 6.2 Improving Scalability and Computational Efficiency

There are several areas in which the scalability and computational efficiency of the voxel space code could be optimized and improved upon. As written, the voxel map code requires a significant amount of memory. The memory requirements for a voxel map are directly related to its size and resolution. In some cases, the resolution of the voxel space was limited due to the large memory requirements. Additional optimizations could be implemented to improve memory management. Significant improvements could also be made to optimize the run time for voxel

map creation. Though fast voxel-traversal algorithms were implemented, the ray-tracing is still computationally intensive and contributes to long processing times. The ray-tracing process lends itself to parallelization, as the results could eventually be combined to form the completed voxel map.

Another area to consider for possible improvements is in the identification of potential future image locations. The voxel-based workflow currently considers all voxel faces on the free-unsampled boundary, provided they are not from texturally difficult regions. It may be beneficial to explore using only a subset of these unsampled voxel faces, particularly in voxel maps with a significant portion of voxels on the free-unsampled boundary. Subsets could be determined by merging clusters of faces that have identical surface normals, and associating a weight with each cluster based on the number of points it represents. This could prove to be particularly useful for real time operations.

## 6.3 Dealing with Texturally Difficult Regions

When testing additional datasets, one of the significant issues raised was the performance of the voxel workflow in texturally difficult regions. This issue was particularly evident in nadir-derived point clouds (WASP), as well as point clouds that were derived from imagery captured at similar vantage points that did not capture all regions of the scene (CorvusEye). In these caases, the ray tracing does not adequately carve out the free space in regions of the voids. In the case of large voids, this results in unsampled voxels that extend through the entire voxel space, resulting in failure in subsequent steps. In the workflow presented here, boundaries are used to identify holes, and these regions can create large unsampled boundaries within the scene. A visibility analysis is then performed in an effort to suppress voxels that were visible in multiple images and may have been a result of texturally difficult regions. This visibility analysis is likely to fail in some cases, as was shown with some of the datasets presented.

Future work should investigate methods to remove these obvious regions from the unsampled voxel faces prior to performing the future image location prediction so that they do not influence the output. Though automation of the process would be ideal, it may be easier to have the user indicate these regions through a visual analysis because they would be very obvious to a viewer.

The texture metric used in the voxel processing was implemented to suppress unsampled

voxel faces that were a result of homogeneous regions from influencing the determination of future image locations. Inclusion of more imagery in the process will never be able to eliminate the unsampled voxel faces that are a result of texturally regions, due to the way that the point clouds are generated using the particular 3D workflow that was used here. It is possible that a shape from shading method, such as the one presented by Saponaro *et. al.* [42], could be used to generate points in these regions, and thus reduce some of the unsampled voxel faces. Another option would be to use a different densification process. A method like semi-global matching, where the objective is to triangulate every pixel in the image, may provide additional points in the textureless regions where PMVS failed to generate any. It is possible that the generation of points would be sufficient such that there would not be a need to identify the textureless regions, and that step of the voxel workflow could be eliminated. If the goal is to construct a model with no unsampled surfaces, then exploration of techniques that can be used to estimate surfaces in the textureless regions is warranted.

## 6.4 End-to-End Testing of Additional Datasets and Flight Patterns

Several datasets were tested through the process of developing the voxel-based workflow, however future work should include processing more point clouds, particularly those that with different features. The workflow should be taken end-to-end with multiple datasets in order to test the impact of other flight patterns on the development of the voxel space. It is imperative to test different flight paths, removing different portions of the input imagery to determine the impact on the voxel space and predicted image locations. This will be beneficial in determining scenarios in which the workflow can recover what is missing, as well as determining limitations of the workflow.

## 6.5 Expanding Sensor Positions

The sensor positions presented in this work were limited to fixed pointing angle or fixed stare point configurations in order to constrain the problem to something manageable. As such, the open parameters were limited to $X$ and $Y$. Expanding the parameters for possible aircraft locations, within reason of what an aircraft is capable of performing, could lead to improved performance.

## 6.6 Developing Flight Lines

In the voxel workflow presented in this research, the primary objective was to determine a position where the most unsampled voxels could be seen. This work was based on the assumption that including more images into the 3D workflow of previously unseen areas would result in fewer voids. It is important to recognize that including a single image may or may not be beneficial, depending on the positions of the other cameras. If the new image is sufficiently different from the initial set, there may not be correspondences to match between the images and no way to tie it into the reconstruction. This work left the way to get to that "ideal" image location up to the end user, leaving significant areas for future work.

The next step in this process could ultimately involve using the maps developed to determine the optimal image location to develop flight lines for an aircraft, such that multiple images would be taken along the flight line and the chances for an improved reconstruction increase dramatically. An example of this is depicted in Figure 6.1, where the sum of the positions across both horizontal and vertical flight lines were computed, and the predicted flight line is indicated in each direction. It is also possible that multiple image locations will be predicted, indicated by multiple local maximums in the predicted map, and flight lines could be created to tie the locations together.

By not restricting the aircraft to traditional flight lines, it may be possible to optimize a flight plan such that more unsampled voxel faces could be imaged. This could also be used to reduce the amount of flight time necessary to collect the required imagery, thereby reducing costs.

## 6.7 Real-time Applications

As presented here, the future image locations could be used to search an image database for the corresponding image, or to design new flight plans for recollecting data. However, the desired image may not be contained in a database, and image recollection is not always a possibility, depending on the application and area of interest.

In future implementations, it would be beneficial to use some sort of reconstruction process that can build up a scene incrementally in flight. From there it may be feasible to simultaneously build the voxel space, such that it can be leveraged to predict future image locations based on the current location of the aircraft. Being able to predict optimal camera locations

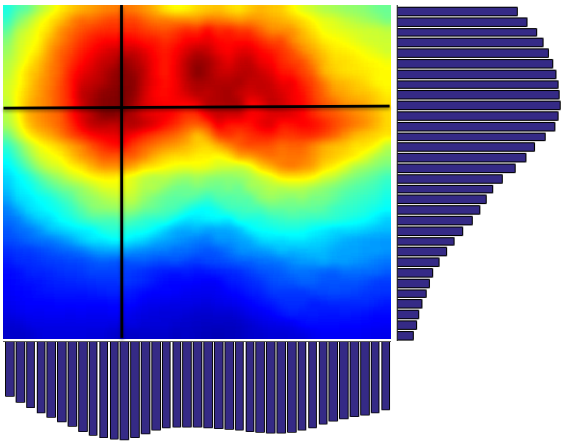


*Figure 6.1: An example of how the potential future image location map can be used to develop flight lines. The predicted horizontal and vertical flight lines are shown, where the plots show the relative sum number of unsampled voxel faces that would be visible across horizontal flight lines (right) and vertical flight lines (bottom).*

in real time opens up the possibility of redirecting an aircraft in flight to obtain imagery from those locations in an attempt to fill in the missing data in the point clouds to obtain a complete model.

## 6.8 Using the Voxel Model as a Surface Model

The voxel model was used here as the foundation to predict future image locations, but it could be leveraged for other applications as well. As was stated previously, 3D point clouds are not directly usable in modeling applications, but rather the point clouds are converted to surface-based models. Accurate surface-based models derived from imagery could be used to automatically generate scenes in DIRSIG and used for more realistic synthetic image generation and modeling, as was explored by Nilosek [89].

Using the voxel faces themselves to create a surface model, as was done for visualizations in this work, introduces a stair-step artifact. Techniques to extract isosurfaces from scalar fields, such as marching cubes (Section 2.3.5) or dual contouring (Section 2.3.6), may be used to mitigate the stair-step effect and achieve a smoother surface. It may also be beneficial in some cases, when the voxel resolution is near its limit, to perform noise removal such that the small

and random unsampled voxels in open space can be removed.

Assuming a nearly complete model with minimal unsampled surfaces, the next logical step would be to assign material attributes to the surface, such as color. The color associated with the point cloud is lost in the voxelization process, and thus the final voxel surfaces have no associated color. Imagery can be used to color the voxel surfaces, as was explored by Hagstrom [88], or hyperspectral imagery can be used to give more accurate material properties to the surface for use in DIRSIG, as was explored by Nilosek [89].

## 6.9 Rendering a Volumetric Model

The voxel space was generated with a probability of free space, or conversely a probability of occlusion. A threshold was implemented to classify the voxels using a ternary system into free, occupied, and unsampled voxels. The 3D model visualizations were then made using opaque versions of the models because it simplified the workflow and allowed for easier comparisons between models. However, this does not utilize the probability of free space parameter to its full potential. Another path to explore would be that of volumetric rendering, where the probability of free space could be equated to an estimate of transmission and voxels could then be rendered as partially transmissive. One potential issue with this is the unsampled voxels, as a decision would have to be made regarding their transmission in the model.

## 6.10 Fusing Multiple Modalities

There has been significant interest in registering point clouds from different modalities, chiefly SfM-based and LIDAR-based point clouds. The voxel-based modeling used here has been shown to work with LIDAR data, provided that the points of origin are available with the LIDAR data [88]. Assuming that an SfM point cloud and LIDAR point cloud have been properly registered, the two could be combined to form a voxel space. As LIDAR is primarily collected at NADIR angles, the points from an SfM point cloud could be used to fill in some of the missing surfaces oriented perpendicular to the LIDAR collection. Additionally, the LIDAR data could be used to improve the quality of the voxel map in texturally difficult regions that do not generate SfM points in the current 3D workflow.

# Appendices

# Appendix A

# Data

The details of the datasets presented in this work are described here. Real datasets used for testing were collected with the Exelis WAMI sensor, the Exelis CorvusEye sensor, and the RIT WASP sensor. The WAMI and CorvusEye datasets were captured in a persistent surveillance mode with fixed stare points, and the WASP dataset was collected with a fixed pointing angle at nadir. A synthetic dataset, generated with DIRSIG for testing purposes, is also presented.

## A.1 WAMI

Security challenges for large scale events have unique requirements such as the need to view large scale areas simultaneously, the need to watch continuously over large time frames with the ability to look back in time, the need to provide imagery to multiple users simultaneously, and the need to be non-intrusive are just a few. Changing needs for security and surveillance required development of a new system, which led to Exelis developing the Wide Area Motion Imagery (WAMI) system, capable of providing wide area persistent surveillance. While the WAMI acronym is being used here to refer to a specific imaging system, it should be noted that it is also used in the imaging community as a generic descriptor for wide-area imaging systems.

Exelis' WAMI system has five visible cameras, positioned to achieve a wide field-of-view, in addition to four infrared sensors with corresponding fast steering mirrors. The sensor is designed to provide 24-hour surveillance through the visible or infrared sensors. Detailed specifications for the WAMI sensor are provided in Table A.1.

*Table A.1: Specifications for the Exelis Wide Area Motion Imagery (WAMI) system. *Note that focal length provided is an average focal length.*

| **Specification** | **Visible** | **Infrared** |
|---|---|---|
| Wavelength | $0.5\mu m$ | $4\mu m$ |
| Resolution | 4872 x 3248 | 1024 x 1024 |
| Pixel Pitch | $7.4\mu m$ | $18\mu m$ |
| Focal Length* | $125mm$ | $195mm$ |

Exelis provided 453GB of image data from a WAMI collect over downtown Rochester, NY. Image collection occurred August 12-13, 2010, and includes a variety of visible and infrared data. The provided images were converted from NITF to JPEG to make them compatible with the SfM workflow used at RIT. In addition to the imagery, position and orientation information were captured at a rate of 60Hz and this information was also provided. A subset of visible images spanning approximately a four minute time frame was carved out for processing. This subset was captured in an orbital fashion with a seemingly fixed stare point. The flying altitude was 3000m, with a nominal declination angle of 40 degrees, and a frame rate of 2Hz. Raw image data from the 5 visible sensors is shown in Figure A.1.

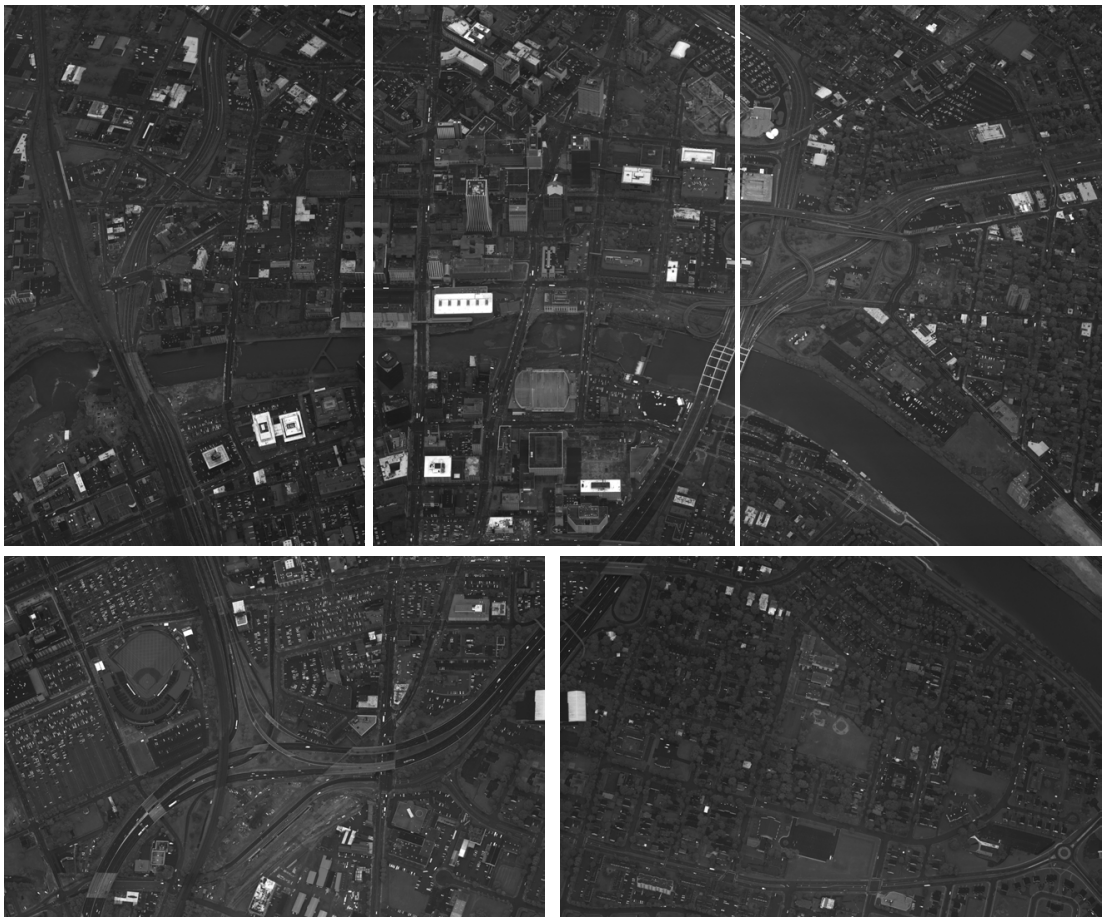


*Figure A.1: Raw image data from the 5 visible sensors on the Exelis WAMI sensor.*

## A.2 CorvusEye 1500C

Exelis introduced the CorvusEye 1500 series in 2014. CorvusEye is another wide-area airborne surveillance system. The system was designed primarily for law enforcement, military, border patrol, and first responders, but could easily be applied to other areas. CorvusEye uses a commercial off-the-shelf approach to reduce costs and is smaller and lighter than previous systems (15 inches, 83lbs), providing an affordable, capable option for domestic and international markets. The system can be used on both manned and unmanned airborne platforms, and is capable of monitoring 3 kilometer region.

The CorvusEye 1500C has a visible color sensor for daytime applications, while the CorvusEye 1500CM has an integrated visible color and mid-wave infrared sensor for both daytime and nighttime applications. For the purposes of this research, the visible color imagery is the focus. There are four cameras, each $6600 \times 4400$ pixels, which can be combined to form a 116 megapixel mosaic. There are 120 pixels of overlap between the sensors. Images are collected at a frame rate of 2Hz. GPS/INS information is available in the image metadata.

Exelis provided 1.78TB of image data from multiple CorvusEye collections. The image collections included the I390-I490 interchange flown at 10,000ft, downtown Rochester, NY flown at 10,000ft and 5,000ft, and the RIT campus flown at 15,000ft, 10,000ft, and 5,000ft. Sample imagery from the downtown collection, flown at 10,000ft, is shown in Figure A.2.

For more information about the Exelis CorvusEye sensor, visit `http://www.exelisinc.com/solutions/corvuseye1500/Pages/default.aspx`.

| | |
|---|---|
| 2 | 0 |
| 3 | 1 |

*Figure A.2: Raw imagery from the 4 sensors of the CorvusEye sensor flown at 10,000ft over downtown Rochester, NY. There is 120 pixels of overlap between the cameras. The diagram below indicates the camera number in the configuration. Cameras 1 and 3 are mounted upside down, which is indicated in the metadata.*

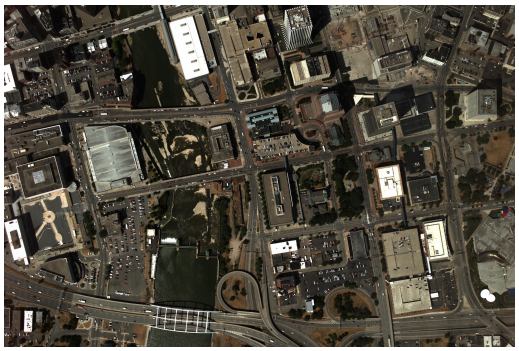


*Figure A.3: Sample WASP image from the downtown Rochester, NY dataset.*

## A.3 WASP

The RIT Wildfire Airborne Sensor Program (WASP) imaging platform, developed in the Digital Imaging and Remote Sensing Laboratory in the Chester F. Carlson Center for Imaging Science at RIT, was originally developed to detect and monitor wildfires. However in recent years its purpose has expanded into a variety of other scientific applications, including 3D reconstruction. The WASP platform is composed of four sensors: visible/near infrared (VNIR), short-wave infrared (SWIR), mid-wave infrared (MWIR), and long-wave infrared (LWIR) [90]. For the purposes of this research, only the VNIR imagery was used. The VNIR imagery has a resolution of 4000x2672 pixels.

The WASP sensor was flown over downtown Rochester, NY in manner so as to capture approximately 80% forward overlap and 90% side overlap in imagery, with the intention of use in SfM processing algorithms for the purposes of 3D reconstruction [91]. This provided a dense collection of nadir imagery over the center of Rochester, with a GSD of approximately 0.3m. A sample WASP image from the downtown collect is shown in Figure A.3.

As part of the SpecTIR Hyperspectral Airborne Experiment (SHARE) 2012 data collection campaign [87], imagery of a single large quarry near Honeoye Falls, NY was captured with the WASP sensor. The site was selected as an opportunity target due to its size and shape, as well as the unique challenge it presented to 3D reconstruction. In most scenes, the structure of the scene is above ground, but the quarry presented an interesting below ground target. The extent of the quarry is shown in Figure A.4, and sample WASP images of the quarry are shown in A.5.

*Figure A.4: Image of the extent of the single large quarry near Honeoye Falls, NY, courtesy of Google Maps.*

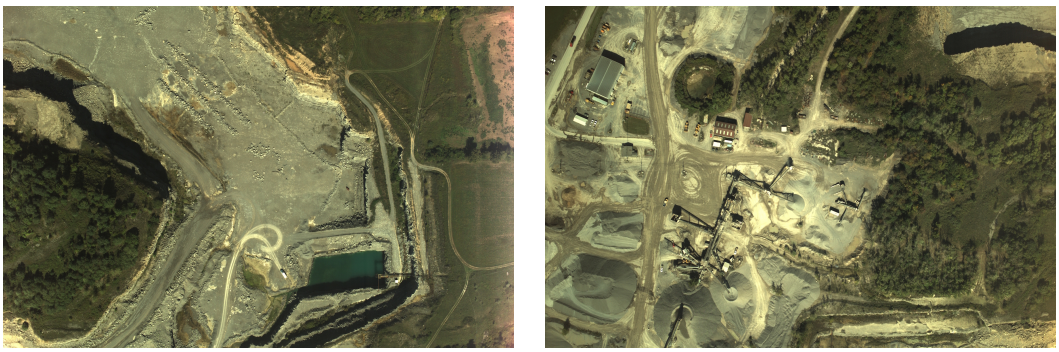


*Figure A.5: Sample WASP images from the single large quarry near Honeoye Falls, NY captured as part of the SHARE 2012 campaign.*

## A.4 DIRSIG

It is difficult to quantitatively compare the performance and output of structure from motion algorithms. The multi-view Middlebury dataset [29] provides calibrated imagery and corresponding mesh models designed specifically for evaluating the performance of multi-view stereo algorithms, but the simple figurines used do not present the same challenges that might be encountered using aerial imagery. Because aerial application of Structure-from-Motion algorithms is becoming more commonplace, it was determined that the community could benefit from an aerial dataset complete with accompanying ground truth.

The Digital Imaging and Remote Sensing Image Generation (DIRSIG) tool provides a unique opportunity to build such a dataset. DIRSIG is a synthetic image generation application designed to produce broad-band, multi-spectral, and hyper-spectral simulated imagery in the visible through thermal infrared regions of the spectrum, with additional capability to produce polarimetric, RADAR, and LiDAR imagery [92]. It is a ray tracing model based on first-principles physics sub-models, such as Bi-directional Reflectance Distribution Function (BRDF) predictions, sensor models, and atmospheric models. The modeled components are combined and integrated radiance images can be produced for an arbitrary number of band passes.

The data set was simulated with a capture date and time of August 12, 2010, 18:00.00 GMT. Megascene 1, a synthetic DIRSIG scene modeled after an area north east of Rochester, was used [93]. A multi-story, building was added to the scene to provide additional height variation. The imaging system was modeled as a simple multi-band framing array featuring an RGB focal plane, with each band featuring a Gaussian spectral radiance response, and a panchromatic band with the spectral radiance response of WorldView-2. The focal length was 125.09mm. The detector featured a $1200 \times 800$ array, with each pixel being square and $32\mu$m in size. The simulated flight path of the sensor was circular with a fixed stare point in the central portion of the scene at an altitude of approximately 800m above ground and a nominal declination angle of 40 degrees. The sampling rate for the data set was 2Hz and images were captured for 210s, resulting in a 420 image data set. The design parameters for the simulation were designed to achieve a similar ground resolution to the WAMI sensor and mimic some of its properties, however the simulation is not intended to be a synthetic replica of the WAMI sensor. Sample images are shown in Figure A.6.

In addition, each radiance image is accompanied by a truth image produced by DIRSIG.

(a) RGB (b) Panchromatic

*Figure A.6: Sample images from the DIRSIG synthetic dataset.*

Adaptive sampling was used to ensure radiometric fidelity, however it was determined that an average position (X,Y,Z) over the rays was not advantageous for this application. Consider casting rays for a pixel that falls on the edge of a structure as is shown in Figure A.7; it is possible that the pixel does not line up exactly with the edge of the structure and therefore some rays may hit the structure and some may just pass it. The average of these rays will result in a point that does not lie on either surface that was intersected by the rays, therefore adding invalid surface points to the truth point cloud. To mitigate this, a minimum and maximum range value is reported for every pixel. This ensures that the points in the point cloud and their respective normals correspond to actual points in the scene, rather than an average point.

For this particular data set, information such as the minimum range, maximum range, (X,Y,Z) hit coordinates for the minimum and maximum range, (X,Y,Z) normal components for the minimum and maximum range and material IDs for the minimum and maximum range are available. Note that the minimum and maximum range will be the same for rigid solid objects such as buildings, but will differ on the edges of buildings and in areas with dense vegetation due to the adaptive sampling methods employed in DIRSIG. Sample images of the hit coordinates are shown in Figure A.8, and a sample 3D point cloud made from the hit coordinates is shown in Figure A.9.

This data set is unique in that it provides a 3D location and normal vector for every pixel in every image and camera location and pointing information is known. The data set was presented at the SPIE Optics and Photonics conference [94] and the full data set was released to the

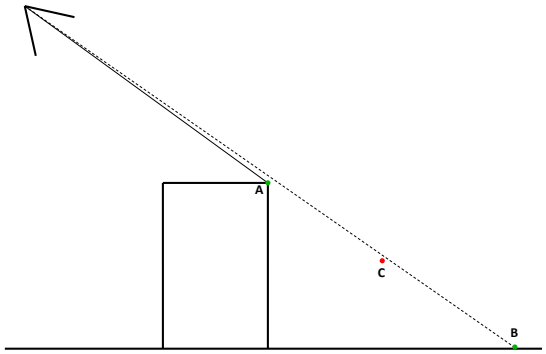



*Figure A.7: Illustration of the casting of rays for a single pixel. One ray intersects the building at point A and one ray intersects the ground at point B. The average of the two rays, point C, is not on any structure in the scene and therefore is not valid truth data. Note that adaptive sampling in DIRSIG uses more rays, the two ray case was just drawn for illustrative purposes.*

(a) X hit (min range) (b) Y hit (min range) (c) Z hit (min range)

*Figure A.8: DIRSIG truth data showing the (X,Y,Z) hit coordinates for the minimum range in a single scene.*

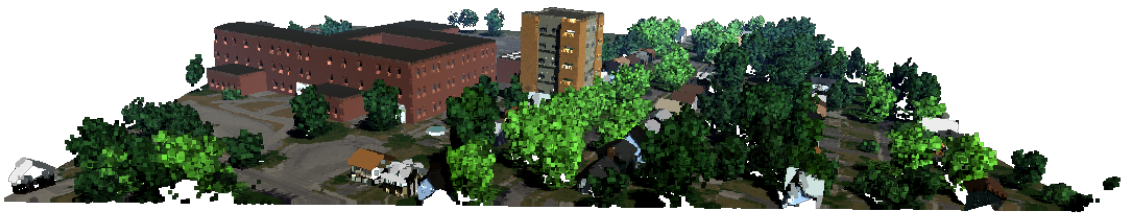


*Figure A.9: Sample point cloud made using the (X,Y,Z) hit coordinates from a single image frame.*

community and is available for download at `dirsapps.cis.rit.edu/3d-dirsig-truth/`.

# Appendix B

# Code

This appendix presents the software written in support of this thesis to develop the voxel based workflow. The code is written in C++, though there are several other helper scripts written in various languages. It should be noted that memory requirements for this software are dependent on the voxel size chosen in combination with the extent of the point clouds. This software also makes use of the OpenCV library [95], which is widely used in the community and available for a variety of platforms.

## B.1 SfM Workflow

The voxel workflow was developed using the results of the 3D SfM workflow used at RIT, discussed in Section 2.1.7, as the input results. A detailed tutorial on the Structure from Motion workflow is available in [89] or online at `http://dirsapps.cis.rit.edu/3d-workflow/?q=3d-workflow`. A set of images should be run through the workflow using the `RunProcess.sh` script with the *a*, *g*, and *k* flags set. The *a* flag runs SIFT/Bundler/CMVS/PMVS, and the *g* flag runs Gtransform. Gtransform is used in this case to rectify the SfM point cloud such that the XY plane in the local ENU (East-North-Up) coordinate system is parallel to the coordinate system of the point cloud. Check to make sure this is the case before running the voxel workflow. Note that the *k* flag prevents running the cleanup process, keeping all the output of SiftGPU, Bundler, CMVS/PMVS, and Gtransform; some of this output is required for the voxel workflow.

After running the SfM workflow, the following directory structure is created:

```
/
├── bundle
├── logs
├── pmvs
│   ├── models
│   ├── pos
│   ├── trans
│   │   ├── models
│   │   └── txt
│   ├── txt
│   └── visualize
└── results
```

The voxel workflow uses the PMVS options file (`~/pmvs/option-0000`), the PMVS patch file (`~/pmvs/models/option-0000.patch`), the Gtransform point cloud (`~/pmvs/trans/models/option-0000`) and the Gtransform camera matrices (found in `~/pmvs/trans/txt/`). Prior to using the voxel workflow, the output Gtransform point cloud must be converted from binary to ascii, such that the new file is labeled as `~/pmvs/trans/models/option-0000-trans-ascii.ply`. Additionally, the camera matrices in the `txt` files need to be converted to the PMVS format using the `cleanTrans.sh` script. Lastly, a voxels folder needs to be created in the working directory.

Note that functions have also been developed to use PMVS or DIRSIG point clouds as input to the voxel workflow. In the case of PMVS input, the indicated voxel size will not be in standard meters, but rather measured against the relative scale of the point cloud. Using a DIRSIG point cloud requires a separate file with the $(X, Y, Z)$ positions of the camera separated by spaces. In the current state, the voxel-workflow is not designed to handle PMVS clusters that result in multiple point clouds, but modification to do so should be relatively straightforward.

## B.2 Creating the Voxel Space

A binary, `voxelMap`, has been made to run voxel space creation. Generating the voxel space requires 3 input parameters: the path to the top level directory in the workspace, the element size of a voxel, and the threshold for the probability of free space. The path to the top level directory is the path to the SFM workspace that was discussed previously in B.1.

The voxel size will impact the memory requirements and processing time. It is recom-

mended that the user start with a larger voxel size and work to smaller sizes to determine the best resolution for the dataset. Voxels are cubic in nature. In the case of a GTRANS point cloud, the units are more intuitive in meters (assuming units of meters were used in GTRANS). If using a PMVS point cloud, determining the proper voxel size is less intuitive and may take a few attempts.

The probability of free space threshold is the threshold that determines the occupied and free classifications. A value of 1.0 is recommended such that all voxels with points are classified as occupied voxels. Noisy point clouds may require an adjustment of this parameter.

The C++ code is run from the build directory with 2m voxels and a 1.0 probability as follows:

```
$ ./voxelMap /full/path/to/SFM/directory/ 2.0 1.0
```

The code will generate the following output:

- `~/voxels/voxels.txt`
- `~/voxels/voxels_free.hdr`
- `~/voxels/voxels_free.img`
- `~/voxels/voxels_occupied.hdr`
- `~/voxels/voxels_occupied.img`
- `~/voxels/voxels_transmission.hdr`
- `~/voxels/voxels_transmission.img`
- `~/voxels/voxels_structure.ply`
- `~/voxels/voxels_missing.ply`
- `~/voxels/voxels_centers.ply`

Detailed descriptions of the output files can be found in Appendix C.

## B.3 Selecting a Subspace

To eliminate the walls that likely appear around the extent of the point cloud, selection of a subspace is recommended. A binary, `subspace`, has been created to save the subspace, requiring the following 8 parameters:

1. Voxel input file (full path recommended).
2. Basename of the output file (full path recommended).
3. Element size of a voxel.
4. Threshold for probability of free space (1.0 recommended).
5. Minimum $X$-coordinate.
6. Maximum $X$-coordinate.
7. Minimum $Y$-coordinate.
8. Maximum $Y$-coordinate.

The file paths are self-explanatory, and the voxel size and probability of free space threshold were described previously. The new input requirements here are the minimum and maximum coordinate values to use that will define the bounding box of the voxel subspace. An easy method to chose the location values is to open the free space `img` file in ENVI, and view the bands, where increasing band number is increasing in $Z$ space. A band from the middle of the voxel space, where building structures can be easily recognized is best suited for this purpose. Note that the origin is in the top left. The cursor location/value feature can be used to pick out a minimum point (top left) and a maximum point (bottom right). The $x$ and $y$ coordinates will define the inputs.

The C++ code is run from the build directory with 2m voxels and a 1.0 probability as follows:

```
$ ./subspace /full/path/to/voxels /full/path/to/subspace/basename
2.0 1.0 minX maxX minY maxY
```

The voxel subspace will save the same files as the original voxel space creation, but with the basename defined in the function input.

## B.4 Predicting Future Image Locations

The last step in the process is to use the voxel space to predict image locations from which voxels will be visible. The two inputs that are required are the path to the current folder in the workspace where the PMVS files are contained, and the full path to the voxel subspace file. Because the number of parameters associated with the future image location prediction was so large, they were not made to be inputs, rather they are located at the beginning of the main function where they are clearly identified. Changing the parameters requires the user to recompile the code. The parameters for the image location predictions are as follows:

1. **Texture parameters:**

   **Window Size** - window size for standard deviation computation in image texture metric (single parameter, window is square).

   **Standard Deviation Threshold** - average standard deviations computed in the local window less than the specified threshold are considered texturally difficult and not used

   **Number of Cameras Threshold** - voxels present in more views than the threshold are considered texturally difficult and not used.

2. **Camera and Angle Parameters:**

   **Camera angle** - defines the circular field of view of the camera, computed as the inverse tangent of half the number of pixels on the sensor in the smallest dimension by the number of focal pixels [radians].

   **Grazing Angle** - angles greater than the specified threshold are considered grazing angles [radians].

3. **Plane Map Parameters:**

   **Block Size** - linear spacing between potential locations [meters].

   **Altitude** - flying altitude [meters].

   **Pointing** - unit vector that specifies the pointing direction of the sensor.

   **Stare** - boolean variable indicating if a fixed stare point is to be used; if true, the pointing variable is the location of the stare point in the scene.

Uniform or Gaussian weighting functions can be used in the plane mapping, and that can be changed in the code as well. The output, `/voxels/planeMap.txt`, is a space delimited text file, where the dimensions correspond to the number of potential locations in the $X$ and $Y$ dimensions and the values are the number of unsampled voxels visible in that location, scaled by the weighting function. The $XYZ$ plane locations are saved as a PLY file ( `/voxels/positions.ply`), where the normals are indicative of the pointing vector. Also output are PLY files that can be used to visualize the voxels that were visible from the predicted maximum location: `/voxels/missing_voxels_visible.ply` and `/voxels/missing_voxels_not_visi`

## B.5 Summary

1. Run the SfM workflow using `sh RunProcess.sh -agk`
2. Convert the Gtransform ply from binary to ascii, where the ascii file is ~/pmvs/trans/models/option-0000-trans-ascii.ply
3. Convert the camera matrices to the proper format using `sh cleanTrans.sh` in the ~/pmvs/trans/txt folder
4. Make the voxels folder ~/voxels
5. Generate the voxel space using the `voxelMap` binary.
6. Select a subset of the voxel space using the `subspace` binary.
7. Generate future image location maps using the `planeMap` binary.
8. Remove duplicate vertices from the PLY files using Meshlab to reduce file size.
9. Generate visualizations in Blender if desired.

# Appendix C

# Data Formats and Handling

The following presents a detailed description of the file formats used throughout the voxel workflow.

## C.1 Voxel Space File Format

The voxel space is saved as several output files. Given a general *basename* for the voxel space, the following files are created:

- *basename*.txt - text file containing properties of the voxel space
- *basename*_free.hdr - ENVI header file
- *basename*_free.img - Binary image data for the free voxels
- *basename*_occupied.hdr - ENVI header file
- *basename*_occupied.img - Binary image data for the occupied voxels
- *basename*_transmission.hdr - ENVI header file
- *basename*_transmission.img - Binary image data for the probability of free space

The properties of the voxel space stored in the *basename*.txt file are the element size (`double`), origin (`double, double, double`), and the number of elements (`int, int, int`). An example file is shown for reference:

```
ElementSize: 0.02
Origin: -4 -3 -5
Numel: 301 351 51
```

The free and occupied data in the voxel space are stored as a flat-binary raster file (.img extension) with an accompanying ASCII header file (.hdr extension), using ENVI conventions. More information on the ENVI header format can be found at `http://www.exelisvis.com/docs/ENVIHeaderFiles.html`. The data is 16-bit unsigned integers, stored as a binary stream of bytes in band sequential (BSQ) order. The probability of free space computation is also written out in a similar manner, but the data is 64-bit double-precision floating-point values. The data was written out this way so that it could be easily read back into the voxel workflow, thereby eliminating the need to recompute the voxel space, and also so that ENVI could be used as a simple visualization tool such that each XY plane in the voxel space could be viewed as an image band.

## C.2 PLY Output Files

In addition to the binary data files that are created, PLY files are also used for visualization of the voxel space. More information on the polygon (PLY) file format can be found at `http://paulbourke.net/dataformats/ply/`. Two files are created to represent the known and unknown surfaces in the scene, called *basename*_structure.ply and *basename*_missing.ply respectively. Each of the files contains the points that make up the faces in addition to the faces themselves. For computational efficiency, four points and one face were written out for each boundary face. The resulting faces are unique, but there is redundancy in the points themselves as the boundary faces share points. The duplicate vertices can be removed using a filter in Meshlab ($Filters > CleaningandRepairing > RemoveDuplicatedVertex$) to reduce file size [50]. A tutorial is provided in Appendix D detailing how to use the two surface PLY files to make figures as they appear in this document.

## C.3 Plane Map File Format

The plane maps are written out as space delimited text files, where the output value is the weighted number of unsampled voxel faces that were visible from each location in the plane map. The number of samples on a line corresponds to the number of samples in the plane map, and similarly the number of lines corresponds to the number of lines in the plane map. This was done for simplicity. The plane maps can be loaded into Matlab as a matrix using the load command, and displayed as a figure after scaling. Figures in this document were generated using the 'jet' colormap feature of Matlab.

Each entry in the output matrix in the text file corresponds to a specific location and pointing angle in the plane map. In the case of a fixed-pointing sensor, the pointing is the same for each location, but the pointing will change for a fixed-stare sensor. Locations of the imagery can be computed using the properties of the plane map, or they can be written out as a PLY file, where the pointing is included as the normal vector for each point.

# Appendix D

# Tutorial: Creating Voxel Visualizations with Blender

Many of the figures in this document were created using Blender [79]. The following tutorial details the specifics of how these figures were created using shortcuts available. Note that some of the commands documented here require a three-button mouse and a numeric keypad. This tutorial was created using Blender version 2.72.

## D.1 Generalized Procedure

The following is a generalized list of the steps a user goes through to visualize the voxel space and generate images in Blender. More details regarding each step are available in D.2.

1. Open a new project in Blender. Delete objects in the start-up file if necessary.
2. Change the engine to 'Cycles Render'.
3. Import the PLY files generated by the voxel workflow: *basename*_structure.ply and *basename*_missing.ply.
4. Pan and zoom if necessary to make the model visible in the viewing window.
5. Check clipping to ensure the entire model will be visible.

6. Attribute materials to each PLY object, using the 'Ambient Occlusion' material type.

   RGB triplet used for structure surfaces: (0.8, 0.8, 0.8)

   RGB triplet used for missing surfaces: (1.0, 0.25, 0.0)

7. Change the background color if desired. White was used for the models presented in this document, RGB triplet (1.0, 1.0, 1.0).

8. Render the model.

9. Add a camera if desired.

10. Render the image through the camera if desired.

The reader is advised that this tutorial is only intended to introduce users to Blender navigation and tools such that it would be possible to make figures for the voxel space. Blender has numerous capabilities that are not discussed in detail here and more information and tutorials are available at `https://www.blender.org/support/`. This tutorial also mentions some Blender hot-keys, and more information on the hot-keys is available at `http://download.blender.org/documentation/BlenderHotkeyReference.pdf`.

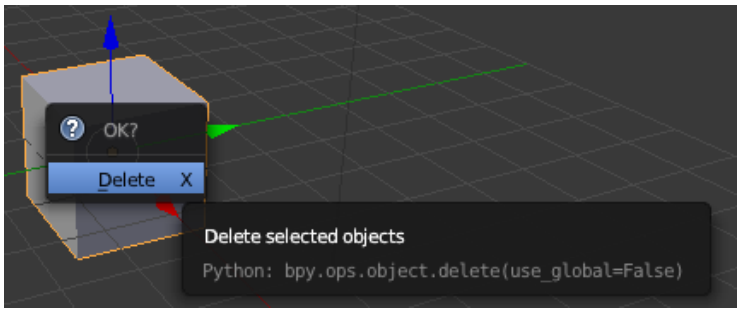



*Figure D.1: Delete an object in Blender by right clicking to select the object and then using the delete key.*

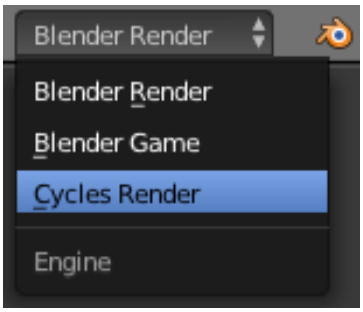



*Figure D.2: Change 'Blender Render' to 'Cycles Render'.*

## D.2 Detailed Actions

### Deleting Objects

To delete these objects from the scene, right-click on an object to select it, and use the delete key to remove it (note on a Mac, function-delete must be used). This is shown in Figure D.1, and this process can be applied to any object. Note that the start-up file can be modified so that it does not automatically load objects, and this is recommended if the user will be making multiple figures.

### Cycles Render

Change the engine from 'Blender Render' to 'Cycles Render' (top middle of the Blender interface), as shown in Figure D.2. This is an easy step to forget, and can be changed later in the process. If still in 'Blender Render' mode, some of the screens shown in this tutorial will appear differently.

### Import PLY Files

Files are imported by selecting $File > Import > StanfordPLY$ as shown in Figure D.3. This must be done for each file. As Blender does have some difficulty with larger file sizes, it is recommended that the user remove the duplicate vertices using Meshlab (or similar), as discussed in Appendix C, before loading the files into Blender.

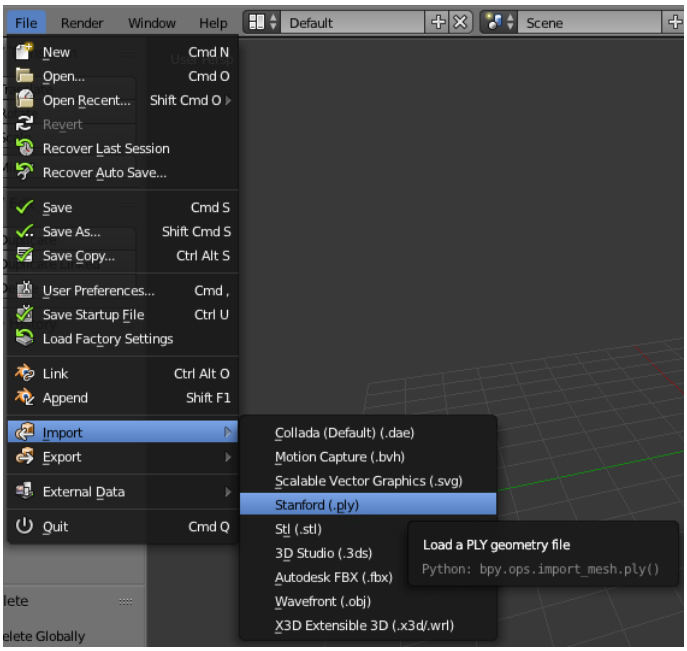



*Figure D.3: Import a PLY file in Blender.*
*Table D.1: Navigating in Blender with a three-button mouse.*

| **Task** | **Control Sequence** |
|---|---|
| Zoom | Scroll button zooms in/out. |
| Rotate | Click and hold scroll button, drag to rotate. |
| Move/Shift | While holding shift, click and hold the scroll button, drag to move. |

### Navigation in Blender

Depending on the scale of the point cloud, it may not be visible immediately after import, requiring the user to zoom out to view it. General navigational tips, including zoom, rotation, and shifting, are given in Table D.1. In addition, methods to change the viewpoint more easily using a numeric keypad are presented in Table D.2.

*Table D.2: Changing viewpoints in Blender with the numeric keypad.*

| Number | View |
|---|---|
| 0 | Camera view |
| 1 | View looking down the Y-axis |
| 3 | View looking down the X-axis |
| 5 | Orthographic view |
| 7 | View looking down the Z-axis |

### Clipping Issues

If the PLY is not visible, even after zooming out, there may be a clipping problem. When in the main window in object mode, the 'n' key is Blender hot-key that can be used to bring up the Number Panel. The number panel shows the location, rotation, and scaling of the active object; the clip end parameter can be modified to extend the viewing. The active object can be selected by right clicking, or by selecting it from the list on the right side of the screen, as shown in Figure D.4.

### Material Attribution

To attribute a material to an object, the object must be selected. Select the *basename*_structure.ply file to be the active object using the menu on the right, shown in Figure D.4. Click on the material tab to add a new material, as shown in Figure D.5. Change the surface type to Ambient Occlusion. If this looks completely different, check to make sure that the engine being used at the top is the Cycles Render engine, as this step is easy to forget. For the figures generated in this document, an RGB triplet of (0.8, 0.8, 0.8) was used for the known structure surfaces. Repeat this process with the *basename*_missing.ply file selected. The RGB triplet used for the surfaces here was (1.0, 0.25, 0.0).

### Render the Model

To view the model as rendered, change the object mode from solid to rendered as shown in Figure D.6. If desired, the background color can also be changed in the world-tab, as shown in Figure D.7. Note that this only affects the rendered view.

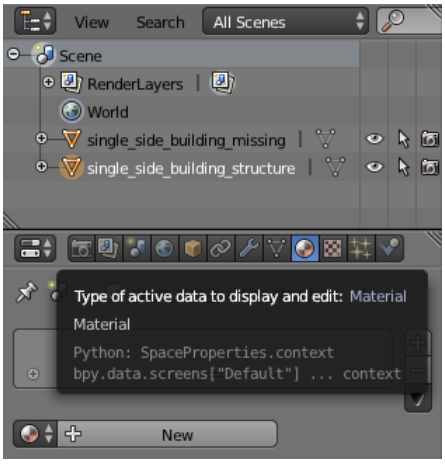



*Figure D.4: Select an active object.*

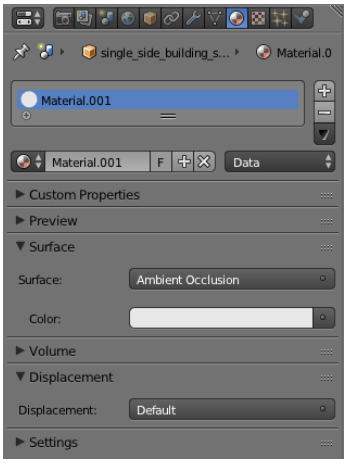



*Figure D.5: Add a material*

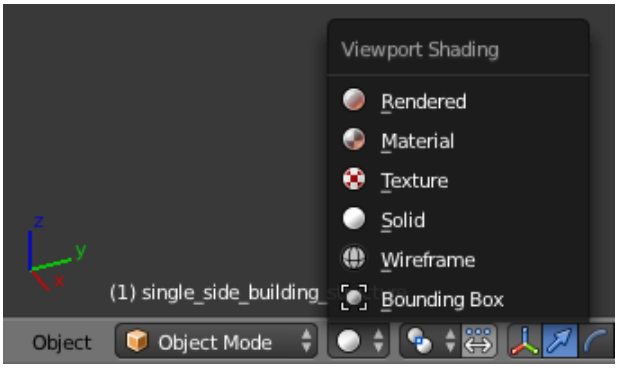



*Figure D.6: Change the object mode to rendered to view the model.*

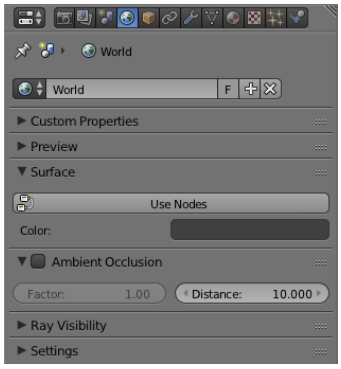



*Figure D.7: Change the world background color.*


*Figure D.8: Camera parameter menu, which can be used to set clipping and change to an orthographic or perspective camera.*

### Adding a Camera Object

A camera can also be added to the scene if desired. This is recommended if it is necessary to be able to recreate images of the voxel space from the same viewpoint, as was done to see the effect of voxel resolution and the effect of the probability of free space threshold. This is done by selecting $Add > Camera$, and the numeric panel (n-key) can be used to change the position and orientation of the camera. Note that a nadir camera is obtained with the X, Y, and Z rotations set to 0. The 5 key on the numeric keypad will change to the camera view. This view is useful to move the camera so that the model is visible in the frame. Again, clipping can be a problem and may render the object not visible from the camera view. With the camera object active, and the camera tab selected on the right menu, the clipping can be extended so that the model is visible, as shown in Figure D.8.

An orthographic camera is often easier to move and check the viewpoint relative to the model. To set the size of the orthographic camera, set the scale equal to the largest dimension of the voxel model (this can be determined in the numeric panel when the *basename*_structure.ply object is active). Finally the camera resolution and number of samples in preview/render mode can be set as shown in Figures D.9 and D.10 respectively. Higher sampling rates take longer to render, but achieve a better looking model. A value of 50 is recommended.

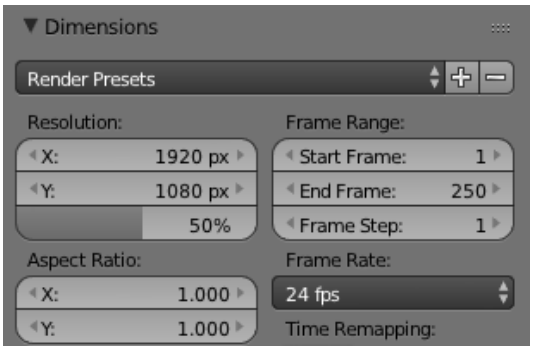



*Figure D.9: Change the camera resolution.*

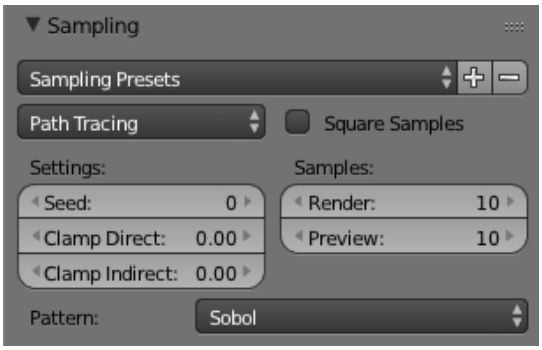



*Figure D.10: Change the sampling.*

**Render an Image**

Rendering an image through the camera view can be achieved using the F12 key, or by selecting $Render > Render Image$. The rendered image can be saved by using the F3 key or by selecting $Image > Save As Image$.

# Bibliography


[1] Ying-mei Wei, Lai Kang, Bing Yang, and Ling-da Wu. Applications of structure from motion: A survey. *Journal of Zhejiang University SCIENCE C*, 14(7):486–494, 2013.

[2] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[3] Hans P Moravec. Rover visual obstacle avoidance. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, volume 2, pages 785–790, 1981.

[4] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, volume 15, page 50, 1988.

[5] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, 1998.

[6] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 9, pages 150–153. IEEE, 1984.

[7] Krystian Mikolajczyk and Cordelia Schmid. Indexing based on scale invariant interest points. In *Proceedings of the 8th International Conference on Computer Vision*, volume 1, pages 525–531. IEEE, 2001.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[9] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1515. IEEE, 2005.

[10] Michael Donoser and Horst Bischof. Efficient maximally stable extremal region (MSER) tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 553–560. IEEE, 2006.

[11] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[12] Engin Tola, Vincent Lepetit, and Pascal Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 32(5):815–830, May 2010.

[13] Jean-Michel Morel and Guoshen Yu. ASIFT: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2):438–469, 2009.

[14] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003.

[16] Duane C. Brown. A solution to the general problem of multiple station analytical stereo triangulation. Technical report, RCA-MTP, 1958.

[17] Manolis I. A. Lourakis and Antonis A. Argyros. SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 36(1):Article 2, March 2009.

[18] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[19] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[20] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Towards internet-scale multi-view stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1434–1441. IEEE, 2010.

[21] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[22] Thomas Pollard and Joseph L Mundy. Change detection in a 3-d world. In *Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2007.

[23] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 807–814. IEEE, 2005.

[24] Derek J Walvoord, Adam J Rossi, Bradley D Paul, Bernie Brower, and Matthew F Pellechia. Geoaccurate three-dimensional reconstruction via image-based geometry. In *SPIE Defense, Security, and Sensing*, page 874706. International Society for Optics and Photonics, 2013.

[25] Noah Snavely. Bundler: Structure from motion (SfM) for unordered image collections. `http://phototour.cs.washington.edu/bundler/`, 2009.

[26] J.Chris McGlone. *Manual of Photogrammetry*. American Society for Photogrammetry and Remote Sensing, 2013.

[27] Andreas Wendel, Arnold Irschara, and Horst Bischof. Automatic alignment of 3d reconstructions using a digital surface model. In *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 29–36. IEEE, 2011.

[28] Chun-Po Wang, Kyle Wilson, and Noah Snavely. Accurate georegistration of point clouds using geographic data. In *3DTV-Conference, 2013 International Conference on*, pages 33–40. IEEE, 2013.

[29] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 519–528. IEEE, 2006.

[30] Frank Neitzel and J Klonowski. Mobile 3d mapping with a low-cost uav system. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, 38:1–6, 2011.

[31] David Crandall, Andrew Owens, Noah Snavely, and Dan Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3001–3008. IEEE, 2011.

[32] Brance P Hudzietz and Srikanth Saripalli. An experimental evaluation of 3d terrain mapping with an autonomous helicopter. In *Conference on Unmanned Aerial Vehicle in Geomatics*, 2011.

[33] Anestis Koutsoudis, Blaž Vidmar, George Ioannakis, Fotis Arnaoutoglou, George Pavlidis, and Christodoulos Chamzas. Multi-image 3d reconstruction data evaluation. *Journal of Cultural Heritage*, 15(1):73–79, 2014.

[34] David Nilosek, Derek J Walvoord, and Carl Salvaggio. Assessing geoaccuracy of structure from motion point clouds from long-range image collections. *Optical Engineering*, 53(11):113112–113112, 2014.

[35] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building rome in a day. In *Computer Vision*, pages 72–79. IEEE, 2009.

[36] Erin Ontiveros, Carl Salvaggio, Dave Nilosek, Nina Raqueño, and Jason Faulring. Evaluation of image collection requirements for 3d reconstruction using phototourism techniques on sparse overhead data. In *SPIE Defense, Security, and Sensing*, volume 8390, pages 83900K–1, 2012.

[37] Changchang Wu. SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT). `http://cs.unc.edu/~ccwu/siftgpu/`, 2007.

[38] David Nilosek. Geo-accurate dense 3D point extraction. `http://dirsapps.cis.rit.edu/3d-workflow/`, 2014. Version 2.0.

[39] Map image of Rochester, NY. Retrieved on December 3, 2012 from `Http://maps.google.com`.

[40] Rochester-NY-skyline.jpg. `http://linkagesrochester.org/drupal/node/29`.

[41] Andrea Vedaldi and Brian Fulkerson. VLFeat. `http://www.vlfeat.org/`.

[42] Philip Saponaro, Scott Sorensen, Stephen Rhein, Andrew R Mahoney, and Chandra Kambhamettu. Reconstruction of textureless regions using structure from motion and image-based interpolation. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 1847–1851. IEEE, 2014.

[43] Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.

[44] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4):349–359, 1999.

[45] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[46] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[47] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 339–346. ACM, 2002.

[48] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM, 1998.

[49] Qian-Yi Zhou and Ulrich Neumann. 2.5 d dual contouring: a robust approach to creating building models from aerial lidar point clouds. In *Computer Vision–ECCV 2010*, pages 115–128. Springer, 2010.

[50] Meshlab. `http://meshlab.sourceforge.net/`.

[51] Hong-Tzong Yau, Chuan-Chu Kuo, and Chih-Hsiung Yeh. Extension of surface reconstruction algorithm to the global stitching and repairing of stl models. *Computer-Aided Design*, 35(5):477–486, 2003.

[52] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)*, 3(4):266–286, 1984.

[53] M Gopi, Shankar Krishnan, and Cláudio T Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Computer Graphics Forum*, volume 19, pages 467–478. Wiley Online Library, 2000.

[54] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel poisson surface reconstruction. In *Advances in Visual Computing*, pages 678–689. Springer, 2009.

[55] Xiang Li, Wangen Wan, Xiao Cheng, and Bing Cui. An improved poisson surface reconstruction algorithm. In *Audio Language and Image Processing (ICALIP), 2010 International Conference on*, pages 1134–1138. IEEE, 2010.

[56] David Levin. The approximation power of moving least-squares. *Mathematics of Computation of the American Mathematical Society*, 67(224):1517–1531, 1998.

[57] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T Silva. Computing and rendering point set surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 9(1):3–15, 2003.

[58] Ravikrishna Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms (TALG)*, 4(2):18, 2008.

[59] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *ACM Transactions on Graphics (TOG)*, volume 26, page 23. ACM, 2007.

[60] A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, volume 28, pages 493–501. Wiley Online Library, 2009.

[61] Evgeni V Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. *Institute for High Energy Physics, Moscow, Russia, Report CN/95-17*, 42, 1995.

[62] Sarah FF Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Medical Image Computing and Computer-Assisted Interventation?MICCAI?98*, pages 888–898. Springer, 1998.

[63] Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205. Eurographics Association, 2003.

[64] Jianning Wang and Manuel M Oliveira. Filling holes on locally smooth surfaces reconstructed from point clouds. *Image and Vision Computing*, 25(1):103–113, 2007.

[65] James Davis, Stephen R Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 428–441. IEEE, 2002.

[66] J Verdera, V Caselles, M Bertalmio, and G Sapiro. Inpainting surface holes. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II–903. IEEE, 2003.

[67] Tao Ju. Robust repair of polygonal models. *ACM Transactions on Graphics (TOG)*, 23(3):888–895, 2004.

[68] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 878–887. ACM, 2004.

[69] M Flood. Asprs guidelines: Vertical accuracy reporting for lidar data. *Amer. Soc. Photogram. Remote Sens. LiDAR Committee. Ver*, 1, 2004.

[70] Thomas K Peucker, Robert J Fowler, James J Little, and David M Mark. The triangulated irregular network. In *Amer. Soc. Photogrammetry Proc. Digital Terrain Models Symposium*, volume 516, page 532, 1978.

[71] Sorin C Popescu and Kaiguang Zhao. A voxel-based lidar method for estimating crown base height for deciduous and pine trees. *Remote Sensing of Environment*, 112(3):767–781, 2008.

[72] Ulla Pyysalo, Juha Oksanen, and Tapani Sarjakoski. Viewshed analysis and visualization of landscape voxel models. In *24th International Cartographic Conference, Santiago, Chile*, 2009.

[73] Shaun R Levick, Gregory P Asner, Ty Kennedy-Bowdoin, and David E Knapp. The relative influence of fire and herbivory on savanna three-dimensional vegetation structure. *Biological Conservation*, 142(8):1693–1700, 2009.

[74] Muge Mutlu, Sorin C Popescu, Curt Stripling, and Tom Spencer. Mapping surface fuel models using lidar and multispectral data fusion for fire behavior. *Remote Sensing of Environment*, 112(1):274–285, 2008.

[75] Theodore C Yapo, Charles V Stewart, and Richard J Radke. A probabilistic representation of lidar range data for efficient 3d object detection. In *Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[76] Gary A Haas. Three-dimensional change detection with the use of an evidence grid. Technical report, DTIC Document, 2006.

[77] Shea Hagstrom, David Messinger, and Scott Brown. Feature extraction using voxel aggregation of focused discrete lidar data. In *SPIE Defense, Security, and Sensing*, pages 76840X–76840X. International Society for Optics and Photonics, 2010.

[78] Shea Hagstrom and David Messinger. Line-of-sight analysis using voxelized discrete lidar. In *SPIE Defense, Security, and Sensing*, pages 80370B–80370B. International Society for Optics and Photonics, 2011.

[79] Blender, version 2.68a. `http://www.blender.org/`.

[80] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[81] Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8(2):161–171, 2000.

[82] Noah Snavely. *Bundler User's Manual*.

[83] Yasutaka Furukawa. *Documentation - PMVS*.

[84] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Proceedings of EUROGRAPHICS*, volume 87, pages 3–10, 1987.

[85] Amy Williams, Steve Barrus, R Keith Morley, and Peter Shirley. An efficient and robust ray-box intersection algorithm. In *ACM SIGGRAPH 2005 Courses*, page 9. ACM, 2005.

[86] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, (6):610–621, 1973.

[87] DIRS Laboratory. SHARE 2012: SpecTIR hyperspectral airborne experiment 2012. `http://www.rit.edu/cos/share2012/`.

[88] Shea T Hagstrom. Voxel-based LIDAR analysis and applications.

[89] David R Nilosek. Analysis and exploitation of automatically generated scene structure from aerial imagery.

[90] DIRS Laboratory. Wildfire airborne sensor program (wasp). `http://lias.cis.rit.edu/projects/wasp`.

[91] David Nilosek, Erin Ontiveros, and Carl Salvaggio. 3D-rochester image and lidar dataset for point cloud reconstruction and processing algorithms. `http://dirsapps.cis.rit.edu/3d-rochester/`.

[92] Digital Imaging and Remote Sensing Laboratory, Rochester, NY. *The DIRSIG User's Manual*, 2013.

[93] Emmett J Ientilucci and Scott D Brown. Advances in wide-area hyperspectral image simulation. In *AeroSense 2003*, pages 110–121. International Society for Optics and Photonics, 2003.

[94] Katie N Salvaggio and Carl Salvaggio. Automated identification of voids in three-dimensional point clouds. In *SPIE Optical Engineering+ Applications*, pages 88660H–88660H. International Society for Optics and Photonics, 2013.

[95] OpenCV: Open source computer vision software. `http://opencv.org/`.