3-2021

Comparison of Conic Ray Tracing for Occlusion Determination on 3D Point Cloud Data

Henry Cho

COMPARISON OF CONIC RAY TRACING FOR OCCLUSION DETERMINATION ON 3D POINT CLOUD DATA

THESIS

Henry Cho, Capt, USSF

AFIT-ENG-MS-21-M-021

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT-ENG-MS-21-M-021

COMPARISON OF CONIC RAY TRACING FOR OCCLUSION DETERMINATION ON 3D POINT CLOUD DATA

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Henry Cho, B.S. Aerospace Engineering, University of Michigan

Capt, USSF

March 2021

AFIT-ENG-MS-21-M-021

COMPARISON OF CONIC RAY TRACING FOR OCCLUSION DETERMINATION ON 3D POINT CLOUD DATA

Henry Cho, B.S. Aerospace Engineering, University of Michigan
Capt, USSF

Approved:

Brett J. Borghetti, PhD (Chairman)	Date
Scott L. Nykl, PhD (Member)	Date
Clark N. Taylor, PhD (Member)	Date

AFIT-ENG-MS-21-M-021

Abstract

The US Air Force has been increasing the use of automation in its weapon systems to include the remotely piloted aircraft (RPA) platforms. The RPA career field has had issues with poor pilot retention due to job stressors. For example, RPA operators spend a lot of time and attention surveilling a suspect on the ground for many hours, so adding automation to this activity could help improve pilot retention. The research problem in this thesis attempted to automate the process of observing a ground target. This thesis presents a method termed conic ray tracing for determining visibility and occlusion of a ground target from locations in the airspace represented by 3D point cloud data. This conic ray tracing method uses 3D points representing a scene to trace rays and then using a matrix formulation of the dot product to compute the angles between every ray to the points representing airspace and every ray to the points representing the ground scene. Whether the angle is inside or outside a fixed-angle cone determines occlusion or visibility respectively. The method was tested on 3D point clouds generated from Structure from Motion using real-world imagery data collected from an MQ-9 Reaper flight test. Because the truth data was not available, the results from conic ray tracing were compared to a reference created by using true graphical ray tracing on a surface reconstruction of the original point cloud scenes. When compared to the reference, the conic ray tracing method averaged 0.81 accuracy over the eight cases studied, and the computational runtime was on average 19x faster than the algorithm for computing the reference results from true ray tracing. Limitations and future work were discussed.

For Sarah

Acknowledgments

I would like to thank my research advisor Dr. Brett Borghetti and my committee members Dr. Scott Nykl and Dr. Clark Taylor for their guidance and patience throughout my research. I would also like to thank Capt Aubrey L. Olson for his assistance with my research and with collecting data. Finally, I would like to thank the United States Air Force for recognizing the value in scientific research and the Air Force Institute of Technology for providing me the opportunity in a graduate program.

Henry Cho

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

COMPARISON OF CONIC RAY TRACING FOR OCCLUSION DETERMINATION ON 3D POINT CLOUD DATA

I. Introduction

The US Air Force has been increasing the use of remotely piloted aircraft (RPA) and its autonomous capabilities for military missions. One of the tasks that RPA pilots and operators spend a lot of time performing is loitering, which is where an RPA flies in a holding pattern in order to observe a suspect on the ground. The research problem presented in this thesis uses 2D imagery to generate 3D point cloud data to represent the spatial information of the real-world environment in order to solve a occlusion determination problem. An RPA needs to maintain visibility of a ground target, and one way to determine visibility is by determining whether or not the line of sight has been occluded by an object between the target and the RPA by using ray tracing. Because rays and points are both singular primitives, occlusion determination for point cloud data needs to either trace thick rays or use finite area points [37]. The proposed solution uses thick rays in the shape of cones with a fixed vertex angle and this method is subsequently called conic ray tracing. It uses a matrix formulation of the dot product to determine the angles between the cone axis and the rays to the points in question. Visibility of the points are determined by determining whether the rays are either inside or outside the cone by comparing the angles from the dot product with the fixed vertex angle. If a ray is inside any of the cones, then that ray is blocked from view and its visibility occluded. The performance of the conic ray tracing method was measured against an approximation to the truth values called the reference. This reference occlusion point cloud was created using true graphical ray tracing, which is known to be computationally costly. The conic ray tracing method was tested using data

of six scenes collected using an MQ-9 Reaper by Capt Aubrey Olson at White Sands Test Facility in New Mexico [56]. The performance of the scenario cases was measured and compared by computing the confusion matrix between the result of the conic ray tracing method and the reference from true graphical ray tracing. This thesis also studied the effects of varying the altitude, varying the fixed cone angle, as well as computation time on the conic ray tracing method. The conic ray tracing method had an accuracy of 0.81 and performed 19x faster on average when compared to the true ray tracing.

1.1 Pilot shortages in the RPA career field

Since 2017, the Air Force has been increasing requirements for the number of all types of pilots but continued to miss those requirements, resulting in pilot shortages [47]. The Air Force had a 1500 pilot shortage in 2017 that was the result of low retention, creating a negative feedback loop by overworking its remaining pilots, driving more pilots to separate from the Air Force [72]. RAND Corporation published a detailed report on the stress and dissatisfaction among the RPA community, citing undermanning, inefficient scheduling, and overloaded workload as the most negative aspects of the career field [30]. Similar shortages continued to hover at around 90% of the required manning through year 2018 and 2019 in the pilot training pipelines, largely due to inconsistent budget shortfalls and increased force requirements in the number of pilots [47]. The pilot shortage included remotely piloted aircraft (RPA) operators. To help relieve the bottleneck in the pilot training pipeline, the Air Force has been increasing the use of training simulators and civilian instructors to train a greater number of student pilots at one time, while maintaining military pilots at bases in an operational role. The Air Force has also been increasing the use of RPA for intelligence, surveillance, and reconnaissance (ISR) as well as for air strikes. While many commercial industries have seen robotics and automation replace its manned personnel and reduce the need for human personnel, the Air Force has not yet seen the same kind of reduction in manning [57]. The Air Force continues to see increasing demand

for pilots, for both manned and unmanned aircraft, even while the adoption of automation has been increasing.

Figure 1.1: The MQ-9 Reaper has a primary set of sensors called the Multi-Spectral Targeting System (MTS) targeting pod mounted under the aircraft. The sensor system includes an infrared sensor, a color optical sensor, and a laser range finder [59, 60].

1.1.1 The RPA platform.

One area of research involves adding autonomy into the existing RPA platforms, such as the MQ-1 Predator or its current generation model the MQ-9 Reaper, shown in Fig. 1.1. People commonly misconceive the notion believing RPA operate fully autonomously or with a large degree of autonomy. In reality, RPA are remotely piloted by a human operator in the Ground Control Station (GCS), where the RPA pilot and sensor operator are housed, where many of its controls require manual input. Some of their capabilities require constant attention from a human operator, including panning the gimballed camera by providing the controls with manual input. Because loitering operations take up a lot of the time from operators, this is the research area is explored in this thesis.

One of the problems of using the camera is that the operators can lose track of a ground target due to objects blocking the visual line of sight to the sensor, resulting in an occlusion of the target. There are several conditions that can enable occlusion of the target. First, there are static objects in the scene, such as trees, walls, and buildings, that obstruct the view and create an occlusion between the target and RPA. Second, the target may be moving in and out of view, creating unreliable visibility of the target. Third, the RPA is also moving, typically flying at a cruising altitude in a holding pattern, during which certain locations of the RPA may obscure the view of the target. However, note that the ground obstructions generally cannot be controlled because trees, walls, buildings, etc are simply part of the environment, and the target's movements also cannot be controlled. Only the flight path of the RPA can be controlled in regards to visibility and occlusion of the target by choosing locations for the RPA to position in the airspace. Therefore, the flight path of the holding pattern should be automated to improve the viewing and surveillance of the target.

To improve surveillance on a ground target, the RPA should be able to observe the target for longer periods of time. In other words, the target should remain in its visual custody by maximizing visibility and minimizing occlusion. Therefore, the RPA should select a flight path that minimizes visual occlusions. One approach to solving this problem is to use a 3D point cloud representation to model the target's ground environment. A point cloud can be created generally in one of two ways: (1) by using a 3D scanning sensor such as Light Detection and Ranging (lidar) to collect a raw point cloud or (2) using a set of imagery collected from an image sensor and using a stereoscopy method called Structure from Motion (SfM) to stitch together a 3D point cloud [54, 73]. Because lidar was not available on the MQ-9 used to collect data, SfM was chosen as the method used for generating 3D point clouds to model the environment. The 3D point cloud represents

the real-world environment, which then can be exploited and manipulated using coordinate geometry and spatial relationships in 3D space to determine occlusions.

A flight test using an RPA collected 2D imagery in the form of video files of various ground scenes. The videos were then sampled into sets of images, which were then ingested into SfM. For each set of images, SfM uses keypoint matching and triangulation to produce a 3D point cloud of the scene. Each scene point cloud is then imported into MATLAB to manipulate using coordinate geometry to determine the locations where a target is visible and where it is occluded. The visibility of the target is determined by a ray tracing method that uses thick rays in the shape of cones, subsequently called conic ray tracing. Conic ray tracing produces an output termed the occlusion point cloud, which represents the airspace shaped in a circular holding pattern that defines flight path of an RPA. The occlusion point cloud contains the location in the airspace indicating where the target is occluded and where the target is visible, intuitively colored red and green respectively in the plots. Red indicates an area where the RPA should not fly and green indicates the area where the target is visible. The green area is where a recurring flight path should be created to surveil the ground target. The conic ray tracing method was tested on eight scenario cases in which the target was placed in various locations in the scenes. Because truth data was unavailable, the resulting occlusion point cloud was compared against a reference created using true graphical ray tracing on a surface reconstruction of the scene. The tests of the cases showed that the most visible location of the target is at a convex corner of a building and the most occluded locations include a concave building corner, alleyways, overhangs. The conic ray tracing method performs decently well, producing occlusion points clouds comparable to their corresponding references.

1.2 Summary

In summary, the RPA pilot retention issue can improve by automating ground target surveillance, which is known to be a mundane task for the human operators. Only the

flight path of the aircraft can be controlled to improve upon the simple circular holding pattern and to increase surveillance on the target. We cannot control which positions are visible or occluded, but we can control where the aircraft flies to maintain maximum time with visual custody. The research problem is a occlusion determination problem using 3D data. Research questions include the following. How can visibility and occlusions be determined? How is the real-world environment represented virtually?

In the remainder of this thesis, Chapter 2 reviews 2D imagery data and a few types of 3D data, especially 3D point cloud data. Chapter 2 reviews methods for generating 3D point clouds and previous methods for the computer graphics problem of occlusion determination. Chapter 3 describes the methodology used to solve the occlusion determination problem in 3D point cloud scenes by using a method termed conic ray tracing. Chapter 4 presents and discusses the results of eight test cases of a ground target in a scene implementing the conic ray tracing method and comparing the result to a reference. Conic ray tracing was tested on 3D point cloud scenes generated from SfM from videos captured on an MQ-9 flight test. Using the conic ray tracing method, the resulting output termed the occlusion point cloud is a solution to the visibility and occlusion determination problem. Chapter 5 provides the conclusion, discusses limitations, and describes future work.

II. Literature Review

Chapter 2 reviews the possible options for representing the ground environment and airspace, including 2D and 3D types of data, methods for generating 3D point clouds, and earlier approaches in previous literature to solving the computer graphics problem of occlusion determination. Concepts include 2D imagery, 3D point cloud data, RGB-D data, voxels representation, lidar technology, Structure from Motion, and ray tracing. Some of these concepts were chosen to be used in the methodology described further in Chapter 3.

2.1 2D Imagery Data

The most common type of 2D imagery data are color images. A color image is represented by a two-dimensional arrays of pixels, where each pixel typically stores three values red, green, and blue, commonly called the RGB color system. The reason RGB is used is because it is the most effective additive color model, which can be added to form magenta, yellow, and cyan [4, 33, 50]. The RGB color wheel contrasts with the RYB color wheel, where the primary colors are red, yellow, and blue (RYB), shown in Fig. 2.1. The RYB color model is subtractive whereas the RGB color model is additive. By adding the light from different amounts red, green, and blue are each represented by a value from 0 to 255 (also called 24-bit color or true color), a computer can represent over 16 million (256ˆ3 or 16,777,216) colors. It makes sense that data structure of image data is stored as a 2D array of pixels and it is also displayed on a 2D grid of little emitting light sources of a computer monitor using the additive RGB color model. A special type of color image is called grayscale, where only the intensity of the light is represented in the pixels, meaning each pixel stores only *one* value from 0 to 255, as opposed to *three* for RGB images. Grayscale is commonly visualized in "black and white" or scaled values of black [36]. An RGB image can be split into its three color components, where the intensity of each color

is represented individually in grayscale, such as the example shown in Fig. 2.2. Notice that the three grayscale images have different shades of gray when compared among each other, which is not immediately noticeable, because the grayscale values in the three components each represent intensity of their respective RGB colors despite being visually gray to the human eye.

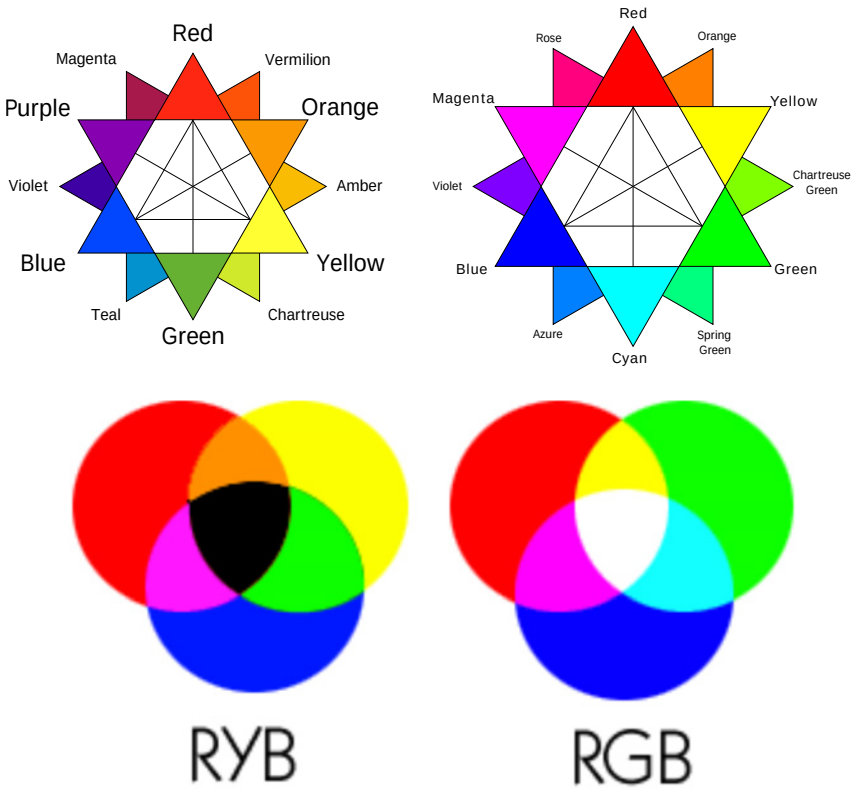

Figure 2.1: The left images show the RYB color model in contrast to the RGB color model on the right. RYB is subtractive and RGB is additive. The RGB color palette can make 16,777,216 colors using 24-bit color depth [4, 15, 40, 49].

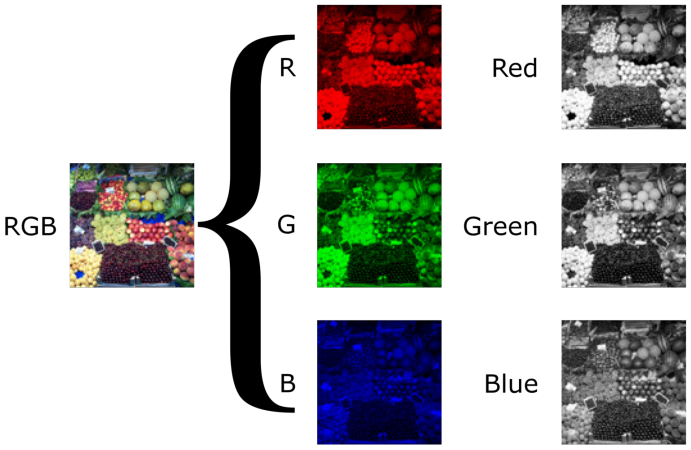

Figure 2.2: An example of an RGB color image composed of their red, green, and blue components, and each color component can be visualized by only their intensity in grayscale [16].

2.2 3D Data

There are several types of 3D data, including 3D point clouds RGB-D depth images, voxels, and CAD models.

2.2.1 3D Point Clouds.

A 3D data type called point clouds have existed since the invention of the laser in 1960 [48]. Recently 3D point clouds have become popularized due to the commercial availability of lidar sensors and their use in autonomous driving research applications. Point clouds store geometric or spatial information in the form of *xyz* points in three-dimensional space within a specified coordinate system, and for this reason 3D point clouds are known as a type of spatial data. In other words, point clouds store spatial information as coordinate values representing distance from an origin. Point clouds are intuitive when visualized in a 3D scatter plot, where a set of points can collectively appear in the shape of identifiable objects. By viewing a 3D plot of a point cloud, it can be seen that the points are a sample of an object's surface features. The two main categories of point clouds are objects and scenes. Examples are shown in Fig. 2.3 Point cloud objects are typically simpler than

scenes. Point cloud scenes typically contain objects and also more spatial features, such as an urban environment containing buildings, cars, pedestrians, etc. Scenes are almost always represented as incomplete information, such as a road incompletely represented. To contrast with images, the advantage of point clouds is that it provides spatial information, which can be used to determine relative positioning between points in space and ultimately to determine light of sight and visibility.

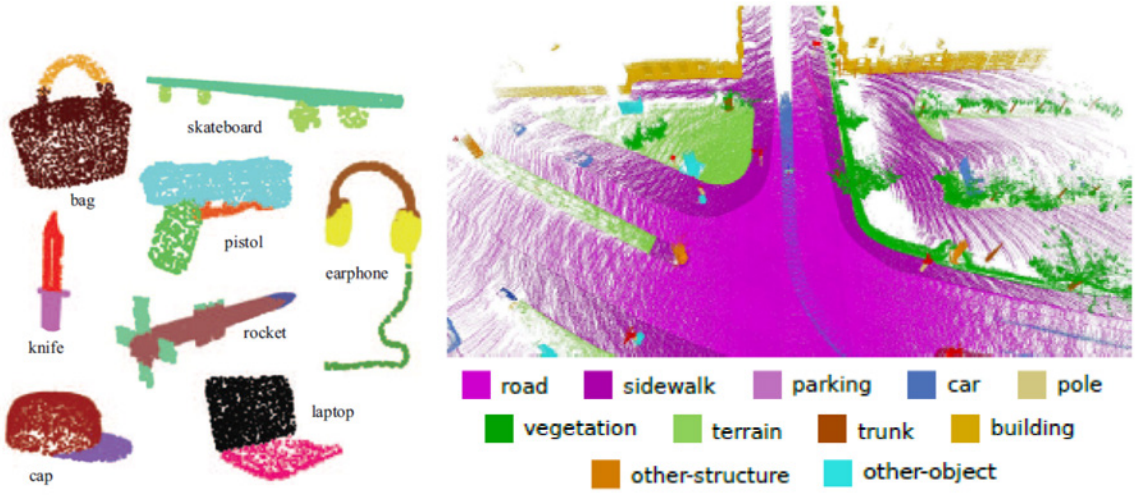

Figure 2.3: Point clouds of complete objects (left) and a point cloud of an urban scene (right) [5, 10, 61].

Point clouds have their own advantages and limitations in regard to their attributes. Point clouds are sparse, meaning most of the occupied 3D space is void space. Point clouds are incomplete, meaning there are missing spatial features. This is especially true for point clouds captured by lidar where it scans an object only on the side facing the sensor. Point clouds are discrete, meaning discrete points are representing the underlying continuous surfaces, so there are inherently unaccounted spatial information when approximating surfaces by point sampling. Point clouds are unordered, meaning the ordering of the points can permute to a different ordering but it still represents the same point cloud. Point clouds are unstructured, meaning the point cloud itself does not carry any relationships in between

the points, and because of this reason a distance metric is chosen to compute the pairwise distance between all the points, e.g. euclidean distance. A nearest neighbor algorithm would be useful for filtering out points from a large scene down to a smaller scene to compute over only the relevant points in a scene. In summary, point clouds are sparse, incomplete, discrete, unordered, and unstructured, which have the computational benefit of approximating surfaces but have their own issues inherent in the sparse and discrete nature of point data.

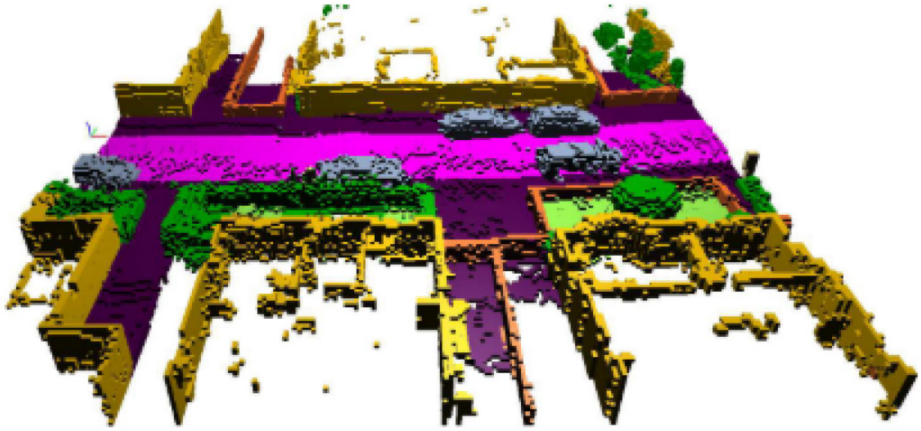

Figure 2.4: A voxelized point cloud of an urban scene of which the colors represent different object categories [5].

To avoid some of the disadvantages, point clouds are sometimes converted into a 3D data called voxels, a name derived from the idea of volumetric pixels or volume elements. One example of a voxel representation is shown in Fig. 2.4, where an urban scene was converted from points into voxels. The process of voxelization is where each point converts into a cuboid using some transformation scheme, providing the 3D point data with volume. However, this creates discretization artifacts called quantization inherent in the shape of cuboids, which corrupts the spatial information by discretizing the sampled points of a smooth surface into jagged cubes in a regular 3D grid, thus misrepresenting the underlying

surface. Using voxels is generally avoided if it is important to maintain the integrity of the geometric features. However, if the quality of the geometric features does not matter too much, then voxels should be considered because the voxel representation can typically compute faster than its original points due to the spatial regularity of a 3D array of voxels, which is the case with 3D convolution and scene voxelization [17]. However, voxels are problematic in that they are memory expensive. Analogous to "empty" black or white pixels, a computer has to store the void voxel space into memory, which scales up cubicly due to the high sparsity of 3D spatial data. There are also some examples of voxel compression techniques [14, 20, 42].

2.2.2 RGB-D Depth Image.

A depth image is a 2D array of R, G, and B values but also includes depth (D) as a fourth value, sometimes called the Z-buffer. Hence, RGB-D data is collected by capturing an RGB image but it also captures depth or range for every pixel. The process of collecting RGB-D data is called range imaging. Each unit of a depth image is still called a pixel but stores four values: R, G, B, and depth. One common way to capture RGB-D data uses two cameras in a stereovision setup with a known separation distance apart. From the two images captured on the two cameras, the range is triangulated using the difference of the two images. Another way is to combine an optical camera with a ranging laser. The advantage of RGB-D data is that the information is still represented in a regularized 2D array as in ordinary 2D imagery, but it also represents spatial information using the depth dimension. RGB-D data can be converted from its 2D array data structure to a point cloud in cartesian coordinates using the camera's properties called the intrinsic matrix [83]. Each pixel in the range image will then represent one point in the point cloud.

Examples of range images are shown in Fig. 2.5. The left image shows an ordinary RGB image of a kitchen scene. The right image shows the depth information of the same kitchen scene, where white means closer proximity and black means further away. The

depth map in this example was likely captured by a flash lidar technique, evident by the black pixels in the depth image representing light absorbing or transparent materials.

RGB-D images are good because it is 2D array information. Machine learning frameworks have also demonstrated when given a single photograph to be good at estimating its depth map [65].

Figure 2.5: Similar to point clouds, examples of RGB-D data include objects and scenes. Left: RGB image of a kitchen scene. Right: Depth map of the same kitchen scene. Notice that this depth map had issues scanning reflective surfaces such as metal [41].

2.2.3 3D CAD Models.

3D CAD models are created virtually using modeling tools in computer-aided design (CAD) software. 3D CAD models are typically created by using points, splines, planes, surfaces, etc in the 3D domain as construction tools to form a 3D model of an object or part. CAD models are used for a variety of applications but are primarily known for its use in 3D manufacturing, architecture, and video game development. A CAD model is "mathematically perfect" such that it typically does not contain the randomness or imperfections of the real world. This is in contrast to point cloud data or RGB-D images collected from reality, which are samples of the real world. CAD models are typically drawn virtually in software and then used in manufacturing to produce the model physically in the real world, such as by additive or subtractive manufacturing, which then introduces

manufacturing imperfections that were not represented in the CAD model. They can be modeled using 3D surfaces comprised of sets of faces or can also be modeled to have bulk volume. CAD models are often the ideal compared to what is produced in the real world, so it could be useful for training data for machine learning as discussed later in future work in section 5.1.2.1. In other words, CAD models are mathematically defined by vector graphics as opposed to RGB-D which is raster graphics due to its arrays of pixels. Examples of CAD models of objects or parts such as shown in 2.6.

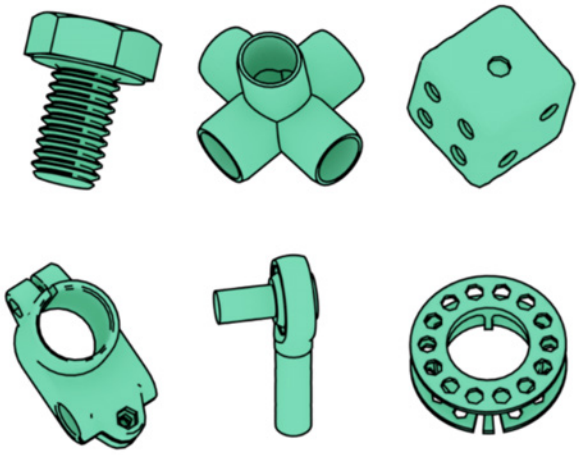

Figure 2.6: Examples of CAD model of parts from A Big CAD (ABC) Model Dataset [39].

2.3 Generating 3D Point Clouds

There are generally three ways to generate 3D point clouds: collecting point clouds using a lidar sensor, converting a RGB-D image into point cloud data, and generating them using a stereoscopy technique called Structure from Motion (SfM) [78].

2.3.1 Lidar.

Lidar is an acronym for light detection and ranging, which is similar to radar but instead of radio waves it measures range using light [63]. Lidar is very intuitive to understand. Lidar sensors use laser light, the speed of light, and the return reflection of the

laser to measure distance–a process known as ranging. First, it pulses laser light and the light travels out. The light hits an object. Some of the light scatters and some of the light reflects back due to the object's reflectance. Then the lidar sensor captures the reflected light and measures the total traveling time of the light pulse. The lidar system then divides total travel time in half and by using the speed of light to determine the distance of how far the ray of light traveled one way, thus measuring the range from the lidar to the object the light hit. This process of light ranging is illustrated in Fig. 2.7. The data collected is a point. This entire process is repeated many times quickly to collect many more points to form a point cloud. In the cases where the light does not return, such as the sky, the lidar system simply does not register a point. Some of the limitations of lidar include issues with noise and outliers, especially from scanning highly transparent or reflective objects.

There are many different kinds of lidar systems commercially available, which all vary in their degree of precision, accuracy, and scanning speed depending on their intended application. One type of lidar uses a high-RPM rotating laser and sensor mounted on top of a car to sense its surroundings. Lidar datasets commonly come from autonomous driving applications since it is a very active field of research. These datasets typically have spiral line patterns due to the rotating nature of the lidar sensor, shown in Fig. 2.8. Lidar has also been used in aerial remote sensing to measure topography and buildings, which is relevant to this research, so it would be helpful to obtain remote sensing datasets. However, the MQ-9 Reaper typically does not have a lidar module installed, and there was no lidar module for the particular Reaper that was flown, so lidar was not used in this research methodology.

2.3.2 Converting RGB-D to 3D Point Cloud.

Converting an RGB-D range image to a 3D point cloud was mentioned briefly from RGB-D data in section 2.2.2. The conversion can be achieved by using the intrinsic matrix $\boldsymbol{K}$, shown in Eq. 2.1. The intrinsic matrix contain the five intrinsic parameters of a camera: f_x, f_y, c_x, c_y, and s. The parameters f_x and f_y are the focal lengths, which typically have

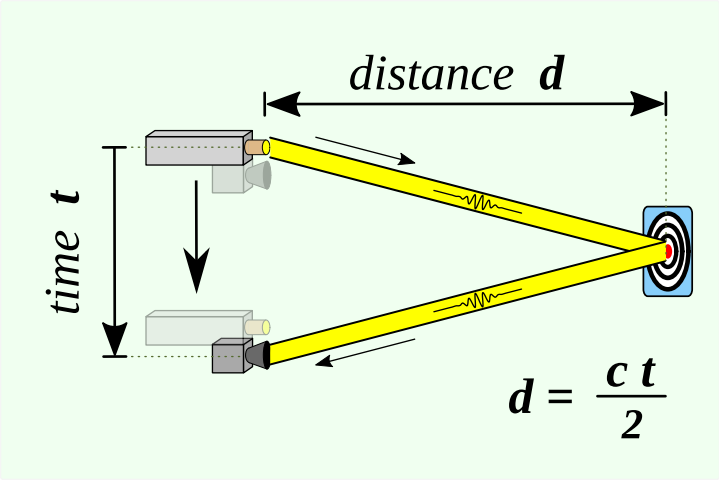

Figure 2.7: A figure that illustrates time of flight of light traveling out distance d, traveling back another distance d, and within total time of flight t. The range for traveling out one way is determined using the time of flight t and the speed of light c in the equation stated in the image [62].

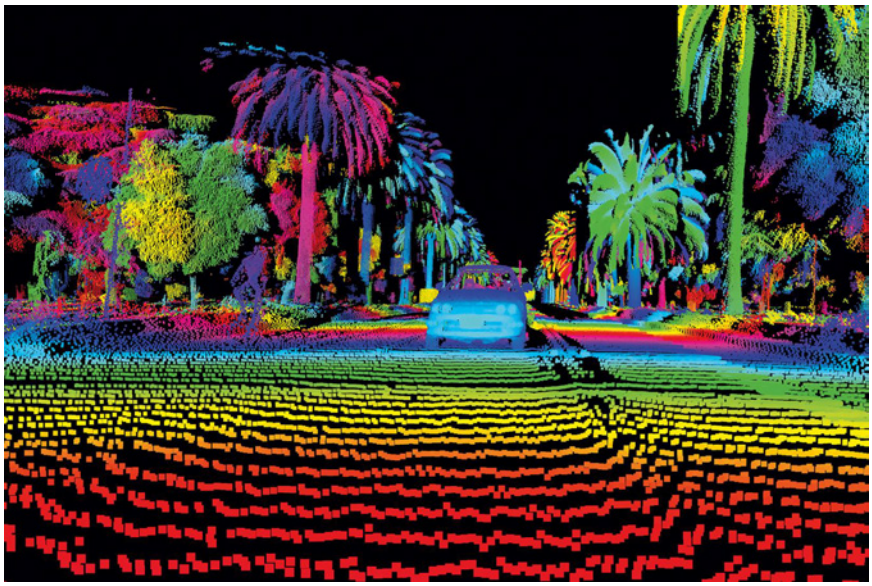

Figure 2.8: An example of a lidar data visualized with voxels where the bands of color represent relative proximity [68].

the same value. The focal lengths can be different due to flaws in the sensor or errors in calibration [69]. The coordinates $\left(c_x, c_y\right)$ is the center of the image sensor in pixel units

called the principal point. The parameter s is the skew coefficient, which is typically zero for most cases but may be nonzero for some digitization processes [51]. The intrinsic matrix is used to convert the xyz points of a point cloud to image coordinates (u, v) using Eq. 2.2, where $\boldsymbol{R}$ is the rotation matrix and $\boldsymbol{t}$ is the translation matrix. However, to convert from image coordinates to xyz points, then the inverse of the intrinsic matrix is needed, shown in Eq. 2.3. Using the parameters of the camera, a range image can convert into a point cloud and vice versa.

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z}\boldsymbol{K}\,[\boldsymbol{R} \mid \boldsymbol{t}] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

$$\boldsymbol{K}^{-1} = \begin{bmatrix} 1/f_x & -s/\left(f_x f_y\right) & \left(sc_y - c_x f_y\right)/\left(f_x f_y\right) \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

2.3.3 Structure from Motion (SfM).

Structure from Motion (SfM) is another approach to generating 3D point clouds [78]. The basis of this technique is called epipolar geometry or stereovision by using two cameras at different views. In contrast to lidar which is an instrument that collects measurements of the real world, SfM transforms existing image data. SfM is an algorithm that takes a set of 2D image data as input and produces a 3D point cloud of the scene in the set of images. The 2D image data should be photographs of multiple different views of a scene, shown in Fig 2.9, to produce a well-representative point cloud. It goes through the steps of keypoint

detection, then matching common geometry features between every pair of images [29]. The location of the common features can then be triangulated in 3D space. Triangulation uses image pairs with the highest number of keypoint matches, producing a sparse point cloud, which can be understood as the "bones" of the point cloud. If there is an outlier image in the image set, it would likely not find keypoint matches and likely does not get used in the triangulation process. Finally, densification adds more point features to the point cloud, producing a dense point cloud.

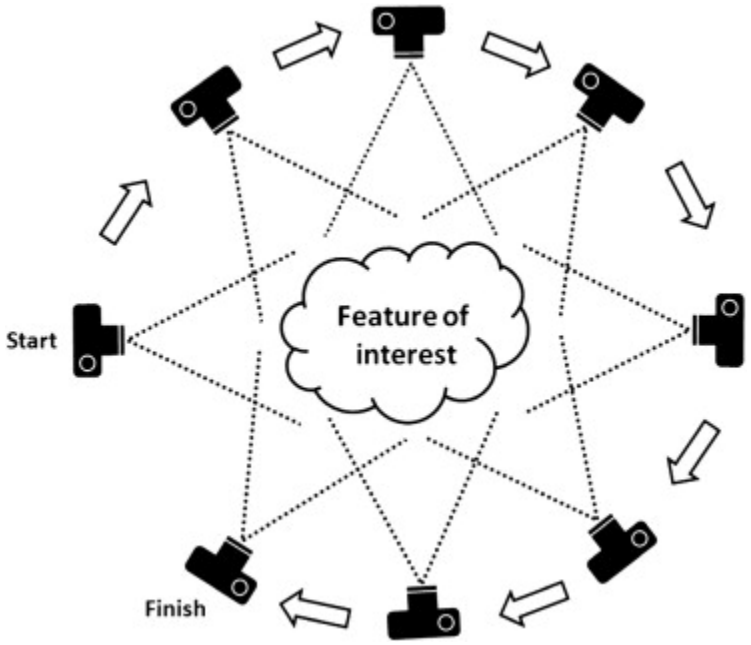

Figure 2.9: The set of images for SfM should be photographs of multiple different views of a scene [78].

2.4 Visibility Determination

Visibility determination has been a basic computer graphics problem since the early days of classical computing [3, 7, 12, 22, 26, 37, 44, 71]. The works have been grouped into mainly two sets of work: visibility between polygons and visibility for point clouds.

2.4.1 Visibility determination between polygons .

Visibility determination has traditionally been for determining whether polygons (lines, triangles, quadrilaterals, etc) are front or behind another polygon such as in 3D space. This has been important for displaying a 3D virtual environment on a 2D computer monitor. There are typically two ways to determine the visibility between polygons to render the result: rasterization and ray tracing [9, 70, 79]. Rasterization begins with the polygons and determines visibility by proximity to the virtual camera that the computer display is rendering. If a polygon is closer to the camera, then the polygon renders in pixels over other polygons that are away from the camera. The visibility is determined and the pixels store a depth value called a z-buffer. In contrast, the ray tracing technique begins from the pixels of the camera and projects rays into the scene. The rays hit the closest polygons and renders it in the pixel from which the ray originated. Ray tracing is computationally expensive because it traces many rays into a scene to sample the polygons. Rasterization is used for rendering geometry, while ray tracing is typically used for rendering reflections and shadows.

2.4.2 Visibility determination for point clouds.

Visibility determination for points has to deal with the issue of being a singular primitive, which can be addressed by either rendering thick rays or rendering the points with finite area.

2.4.2.1 Rendering thick rays.

One approach to determine visibility for point clouds is to render "thick" rays, or rays that have a dimension of surface area or volume. One method uses cylindrical rays for ray tracing 3D point geometry [66]. It renders ray traced images with global illumination using unstructured point data to avoid reconstructing the underlying surface or topology. This method allows more complex illumination models while still maintaining the simplicity of the point primitive. This method detects intersections with the points by tracing a

cylindrical ray through a scene and samples the local point density to determine if the number of points is above a predefined threshold. All the points within the cylinder are used to interpolate the position and normal of the intersection. The method demonstrated results for shadows and global illumination.

2.4.2.2 Rendering finite-area points.

Instead of rendering thick rays, the other approach to resolve the singular primitive issue of 3D point data is to render finite area on to the zero-dimension points. Rendering points into finite area is a technique generally known as splatting [84]. One technique called surface splatting renders point clouds into surfaces using a special formulation of the Elliptical Weighted Average (EWA) filter [84]. Surface splatting is a point cloud surface reconstruction technique that can also handle textures. In Fig. 2.10, 3D points are being transformed into circular splats to form a surface of the points.

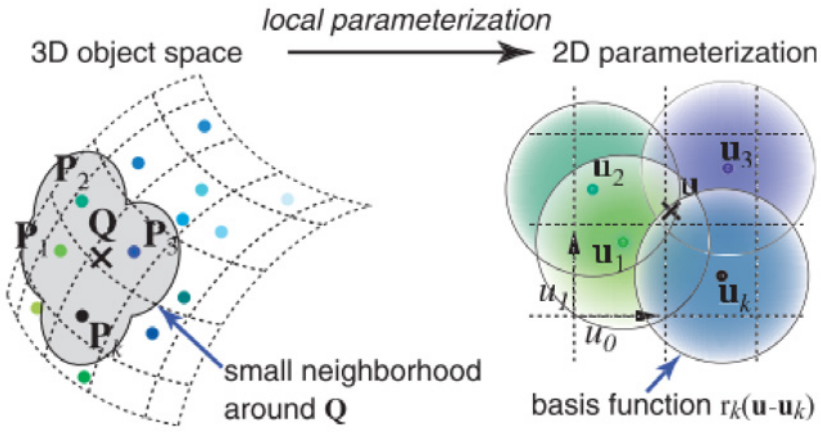

Figure 2.10: Surface splatting transforms 3D points into circular area called splats [84].

Splatting is a popular technique for constructing surfaces from a point cloud. For example, Qsplat is a system that efficiently renders displays large meshes via polygons based on point data [64]. Another research paper presented a technique that optimized surface splatting to render high-quality surface models [81]. Because computational

complexity is proportional to the number of primitives used to represent a given object, complexity reduction is especially important for point geometry. They presented a subsampling technique and global optimization scheme. Their subsampling technique converts points of a dense point cloud into splats. The global optimization scheme minimizes the number of splats that cover the entire surface while staying below a maximum error tolerance. The optimized surface splatting technique outputs a hole-free piecewise continuous surface approximation and achieves significantly lower splat counts. The technique essentially uses fewer splats while maintaining a quality surface reconstruction output. Another paper also focuses on improving the computational efficiency of splatting. This paper presents an algorithm called deferred splatting that renders surface elements (surfels) only from the points that are visible [27]. In this case, visibility is defined by a virtual camera in the virtual 3D environment that shows the 2D projection to a computer display. This method renders only the points that are visible to the camera, and therefore reduces the computational cost by avoiding the rendering of points not visible by the camera. They use a GPU accelerated point selection algorithm to efficiently render visible points into surface elements. This algorithm dramatically improves the processing performance especially for complex scenes that have a lot of geometric features.

2.5 Summary

2D image data is good for visualizing color on a computer monitor, but it does not contain the necessary spatial information to determine occlusion in 3D space. While RGB-D data does contain a depth dimension, the RGB-D sensor is not available on a MQ-9 Reaper. 3D point clouds contain spatial information in cartesian coordinates and can be generated from 2D imagery using SfM. However, points have the issue of being singular primitives, so the rays need to be thick or the points need to have finite area to perform ray tracing to determine visibility.

This thesis contributes to the research area by solving the visibility determination problem using a method termed “conic ray tracing” on point clouds generated from 2D imagery. The gap in this area is that there needed a way to determine visibility with results similar to ray tracing but not as computationally costly. The matrix multiplication of conic ray tracing is what enables its greater computational performance. The results from conic ray tracing were compared against results from the true ray tracing method.

III. Methodology

Chapter 3 presents the research problem and the methodology used to provide a solution to the research problem. The research problem is first reintroduced and further elaborates on the key research questions that this thesis seeks to answer. Then, an overview of the proposed solution is introduced by showing how the raw data is processed through each step in the pipeline from beginning to end. Every step of the proposed solution is further explained in detail with some suggestions and considerations of future work. The end of chapter 3 describes how the performance of the solution was measured by comparing the result of the proposed solution against a reference. The reference is considered the "truth" because actual truth data or an ideal performance measure was unavailable.

To summarize, the research problem involves determining the occluded and non-occluded locations of an RPA while maintaining a continuous line of sight on a ground target for surveillance by determining visibility between the target and the RPA. The proposed solution solves for the occlusion locations using point cloud data, ray tracing, the dot product, cone angles, and boolean logic. First, the workflow begins by collecting 2D videos using an image sensor on an RPA. The video is sampled into a sequence of images. Then, the images are ingested into SfM to generate a 3D point cloud of the scene. Using coordinate geometry, the target defines spatial relationships between itself and the points in its environment using a ray tracing technique that uses conic thick rays with a fixed vertex angle to determine occlusion, which is subsequently called conic ray tracing. The conic ray tracing method produces an occlusion point cloud that defines the locations in airspace where an RPA should fly and where to avoid in order to maintain visibility of the target. The occlusion point cloud is compared to a reference occlusion point cloud created using true graphical ray tracing of a surface reconstruction of the scene. The confusion matrix is computed using the occlusion point cloud produced using the proposed conic ray tracing

method against the reference to compare the performance of the methods. The accuracy and runtime were also measured and compared.

3.1 Research Problem

One task to automate in the RPA mission is ground target surveillance. An RPA operator can be tasked with observing a suspect target for many hours over many days where the suspect's actions are often uneventful. An RPA pilot typically flies circular holding patterns over a target, but sometimes the target is occluded from sight. To improve upon the holding pattern and to mitigate the issue of occlusion, an RPA should be able to fly in a holding pattern that views the target more often, specifically maintaining line of sight between the RPA and the target. The RPA can view the target more often by avoiding the locations where the target has been occluded and by flying in the locations where the RPA maintains line of sight on the target. Therefore by solving for the visibility and occlusion determination problem, an RPA can avoid occluded locations and fly in non-occluded locations so that the visual custody of the target is maintained, the surveillance of the target is improved, and the attention of the human operators is freed up for other tasks.

3.1.1 Representing the environment.

How can the occluded and non-occluded locations be determined? First, because the line of sight can be intuitively represented in 3D space with relationships between the target and RPA to model the real world, it makes sense to also represent the scene in 3D space. Then, what type of 3D data should be used? The two most common types are RGB-D and point clouds. However, RGB-D data involves a 2D image represented with RGB color information but also with depth values, so each pixel carries four values–three for color and one for depth. RGB-D data is often known as "2.5-D" data because its spatial quality is measured relative to the range from the sensor and does not objectively represent 3D spatial information in the way that point clouds do. Point clouds are an intuitive representation of a 3D environment, however, it does have its own limitations as well mentioned in section

2.2.1. Point clouds are a list of points plotted in 3D space and the set of points do not have structure. The surfaces and bulk volume of an scene are not represented, so a point cloud can be thought of as a set of points sampled from the surfaces of the objects in the scene. Point clouds are sparse–containing voids in between points, representing incomplete spatial information–and imperfect–modeling or approximating the true underlying 3D shape and form. The ground environment can be represented using a point cloud to utilize its spatial information.

3.1.2 Determining the occlusions .

If the ground target can be perfectly identified from its surroundings, how can an RPA use this information to find the occlusions? If given that the point set of the target and the point set of all its surroundings are now known, then how the surroundings occlude the target can be determined by these two separated point sets. However, because points are dimensionless, all points are viewable from any location in space. That is to say, a point cannot occlude any other points due to its dimensionless property. In the real-world, the target and its surroundings has surfaces and volume, which create occlusions by blocking the reflected light emitting from the scene from entering the sensor. However, modeling the scene using a point cloud has the issue because point geometry do not blocked rays of light as if a surface was there, but the occlusions need to be modeled somehow. One way would be to fit simplified 3D models over the point sets, creating the surfaces needed to block light. However, this would need the point cloud segmented further more point sets to find the proper fits. Another way the model the occlusions would be to create a line of sight, or ray of light, that is represented with volume. For instance, if the ground target emits light represented by a voluminous cylinder, instead of a one-dimensional ray, and the light intercepts a point representing a wall, then that light would not make it to the RPA sensor, meaning the directed volume of space represents occluded space. This is essentially trying

to cast a shadow, where the shadow is the occluded space, but the dimensionless issue needs to be address when trying to cast a shadow using a point cloud.

In summary, the research problem involves automating the flight path of an RPA to fly in a holding pattern that conducts surveillance on a ground target. The research questions include:

- How can the occlusion space be determined, i.e. the locations where the RPA cannot view the ground target due to an occlusion?
- How is the real-world ground environment represented virtually?

3.2 Proposed Solution

3.2.1 Data Pipeline (Overview).

The proposed solution can be summarized using a data pipeline shown in Fig. 3.1. Each step generally describes how the data is being manipulated. The data pipeline begins at the RPA step where the RPA flies over and captures a video of a ground target. The raw video is the source input data. The raw video is first decomposed into time-sequenced images, then using SfM the set images produces a single point cloud. The point cloud goes through some pre-processing steps, such as georectifying and de-noising to clean up the point cloud. These steps are repeated for more videos, each video captures a different scene, producing a total of six scenes. Using coordinate geometry, the point cloud was used to compute angles using the dot product and compared against cone angles. Then using logical indexing to separate the point cloud into two point sets: occluded and non-occluded. These two point sets called the occlusion point cloud were visualized on one plot. This coordinate geometry manipulation was termed “conic ray tracing” and is discussed is detail in section 3.2.3.

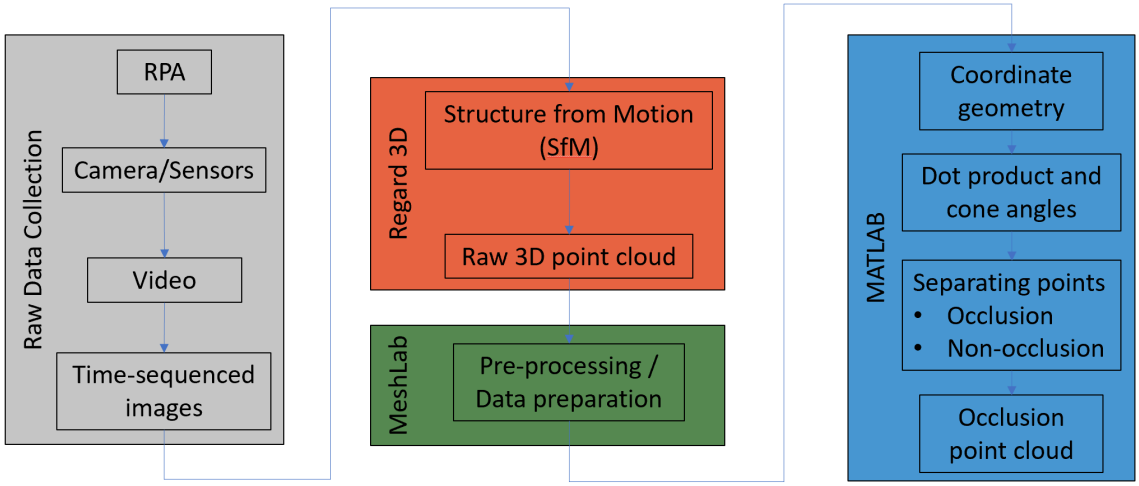

Figure 3.1: The data pipeline begins at the RPA by producing the raw video of a ground target. The raw video is sequenced into images and by using SfM it produces a point cloud of the scene. The point cloud is processed using surface reconstruction in MeshLab to create the reference. Both the raw point cloud and its reconstructed surface mesh are used in coordinate geometry to determine the occlusion point cloud.

3.2.2 Collecting, processing, and preparing data.

The raw video data was collected using optical sensors on an RPA. Then, the videos were sampled into time-sequenced images. Then, the images were processed using a SfM software package called Regard 3D, which produced dense point clouds for each of the scenes.

3.2.2.1 Video and time-sequenced images.

In order to answer the research question of how an RPA can automate its flight path pattern based on the location of the ground target, the RPA needs to have some ability to sense and gather information about its environment. An MQ-9 Reaper has an electro-optical sensor installed as part of its sniper targeting pod used for detecting and tracking ground targets. On board the RPA, the sensor captures and collects imagery data and the RPA transmits the data to a ground control station to the pilot and sensor operator. The imagery data streamed in real-time and was stored as full-motion video. These videos were

the raw source data used in the data pipeline and were collected by Capt Aubrey Olson using an MQ-9 Reaper on a flight test at White Sands Test Facility in New Mexico [56]. The main subject of the videos were typically a primary building and its surroundings, which may include cars, roads, lamp posts, and shrubbery in a desert sand environment. The videos were sampled into a set of time-sequenced images, and then the images were imported into Structure from Motion (SfM) [78]. Using an open-source implementation of SfM called Regard3D, the set of images was imported to compute for matching keypoints between every pair of images that which forms the 3D point cloud [31].

3.2.2.2 Structure from Motion (SfM).

SfM is a technique that takes an sequence of images and outputs a 3D point cloud of the scene based on the input images. While SfM can produce a point cloud from two images with an offset to provide difference vantages, one of the limitations of using SfM for this application was that it requires at least a full orbit around the ground target to produce a usable or well-defined point cloud. Because SfM constructed the point cloud by using different views of the subject and by using more multiple views, SfM is able to triangulate more spatial features in the scene. For example, if by processing only two images in SfM, the resulting point cloud would be an incomplete representation of the whole environment. If using SfM to characterize a building for example, using more images around the building can produce a point cloud with more points that describe the spatial features of a wall that otherwise could not be characterized by using only two images. The 3D points in the point cloud are based on common image features–called keypoints–from each pair of input images. Unfortunately, when processing only two images it produces a very sparse and incomplete point cloud, which is difficult to extract any uniquely defining spatial features, and therefore the resulting point cloud is underdefined and generally unusable.

One open-source implementation of SfM is a software package called Regard3D, which was developed on the Open Multiple View Geometry (OpenMVG) C++ library

[32, 52]. The software package provides a GUI interface able to import a set of images of a scene, and by computing keypoint matches, it produced a 3D point cloud of the scene. The point cloud then needs its coordinate system oriented in a standardized orientation–a process called georectifying. The *xy* plane was set as the ground plane and the *z*-axis points above the ground, positive with increasing altitude. Georectifying is important for when continually refining a point cloud using successive images or for when merging two point clouds together as the point clouds need to have a common set of coordinate axes. However, Regard3D is currently not able to continually adjust the point cloud by adding additional input, so the source code would need to be modified and developed for this capability, which is hosted on GitHub written in C++ [32].

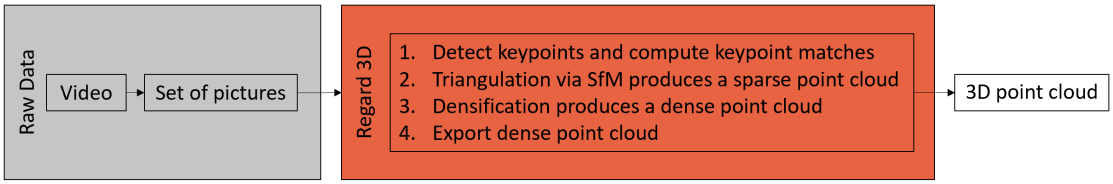

Figure 3.2: The SfM process uses a set of pictures of a scene and produces a 3D point cloud.

Using the baseline Regard 3D package, it imports a set of images and outputs a point cloud of the scene described by the images. An overview of the process is shown in Fig. 3.2. The matching algorithm used to compute keypoint matches is the Fast Library for Approximate Nearest Neighbors (FLANN) with the Classic Accelerated-KAZE (A-KAZE) detector [1, 2, 53]. Note when computing matches, random sampling is involved in the process, resulting in different keypoint matches in each execution. The keypoint sensitivity and matching ratio were manually set at their default values: 0.0007 and 0.6 respectively. After computing keypoints, the keypoints are then used for triangulation using the Incremental SfM algorithm [67]. Triangulation uses image pairs with the

highest number of keypoint matches to produce a sparse point cloud of the scene. Next, the sparse point cloud is processed by densification, which produces additional points using the Clustering views for Multi-view Stereo and Patch-based Multi-view Stereo (CMVS/PMVS) densification method with the default values: level 1, cell size 2, threshold 0.7, wsize 7, and minimum image number 3 [23, 24]. Using the sparse point cloud as a guide, it adds more points thus more spatial features to the point cloud, producing a dense point cloud as the output, shown in Fig. 3.4. This is the point cloud of the scene of the ground environment, which is then imported into 3D modeling application called MeshLab for de-noising and manual edits of the point cloud [11].

One of the limitations of this is that these are manual steps that uses GUI applications meant for a human in the loop to interpret and adjust as needed.

Figure 3.3: A single view of a building complex captured on an MQ-9 Reaper that was encircling overhead in the air. By capturing multiple views and obtaining more spatial information, Structure from Motion (SfM) can produce a point cloud with richer features.

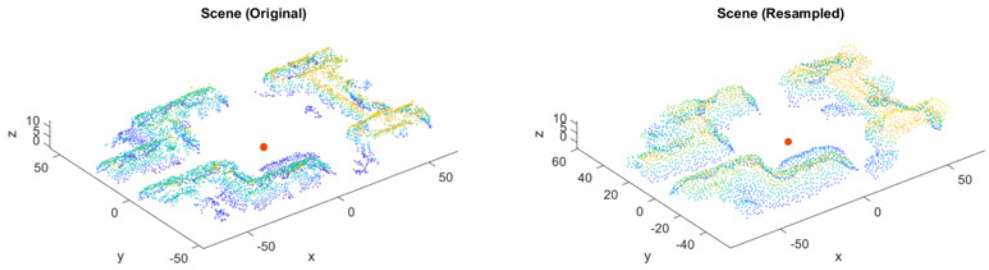

Figure 3.4: On the left, a point cloud of the building complex, generated from a sequence of images using Structure from Motion (SfM) [31]. On the right is the resampled point cloud using screened Poisson surface reconstruction and then Poisson disk sampling [11, 13, 38].

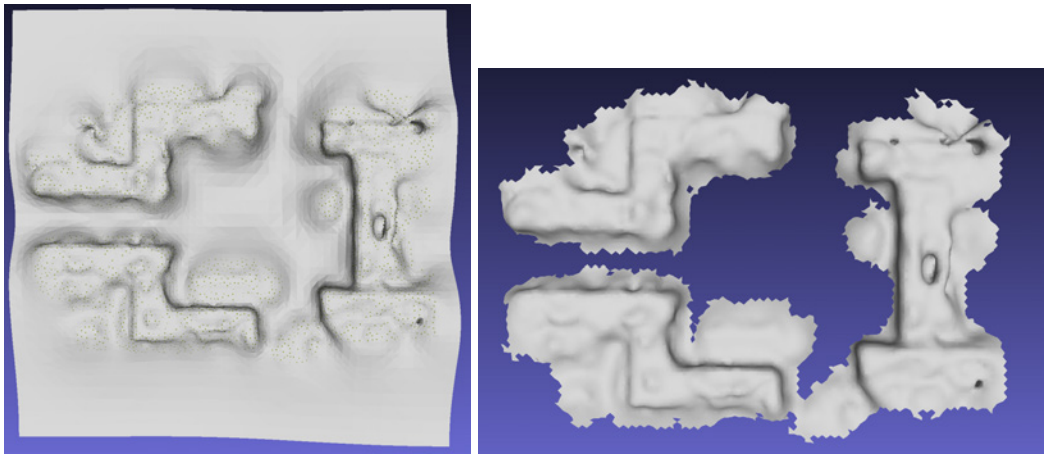

Figure 3.5: On the left, using screened Poisson surface reconstruction to produce a singular sheet surface to bridge voids in the original point cloud [38]. On the right, the faces that were not represented in the original point cloud were selected for deletion [11].

3.2.3 Coordinate geometry to manipulate the point sets (Conic ray tracing).

Now that SfM has produced a point cloud of the scene, a ground target was synthetically represented as a single point in the virtual environment. Coordinate geometry was used to determine whether or not the target was occluded from view by an RPA flying in a holding pattern in the sky. For ease of computation, the target was represented by a single point, which was meant to be the centroid of the target located at $z = 1$ meter. For

a unweighted raw point cloud, the centroid is the same as computing the arithmetic mean. Using coordinate geometry, the target centroid point $T = (x_T, y_T, z_T)$ is used with each non-target point $N_i = (x_N, y_N, z_N)_i$, for $i = 1$ to I total number of *non-target* points, to create a set of vectors $\vec{TN}_i$. This vector can be understood as the axis of a cone, having the target centroid as the cone vertex. The angle made between the cone axis at the vertex will be small, such as 1 to 5 degrees, representing a "cone of occlusion" that is used to determine whether or not a non-target point causes an occlusion between the target and RPA. A set of uniform random points $R_j = (x_R, y_R, z_R)_j$, for $j = 1$ to J total number of flight path points, are generated on a 2D plane at a constant altitude z_R for all j, representing possible viewing positions of the RPA, and which will ultimately be used to help determine the flight path of the RPA. The target point T also uses each point R_j to make a set of vectors $\vec{TR}_j$. The vectors $\vec{TN}_i$ and $\vec{TR}_j$ are each unitized by dividing the vector components by its L2 norm, resulting in $\hat{TN}_i$ and $\hat{TR}_j$ as shown in Eq. 3.1 and 3.2. Fig. 3.6 shows the two sets of vectors before normalizing to unit vectors; note that this diagram does not draw every vector. The angle between the two vectors on the plane made by the vectors can be found by using the matrix definition of the dot product as shown in Eq. 3.3. Matrix multiplication is used to take advantage of computational advantages. Multiplying the two vector arrays results in a $I \times J$ matrix of cosine of the angles θ_{ij} between every vector pair, and then simply compute the arc cosine to obtain the matrix of all the angles $\boldsymbol{\theta}$. This matrix contains all of the angles of every vector pair, so then the angles can be compared to a constant cone angle θ_{thresh}, such as 5° or $\pi/36$ rad. If an angle θ_{ij} is greater than θ_{thresh}, then the vector $\hat{TR}_j$ is located outside of the cone and the target T is still visible by R_j. However, if the angle θ_{ij} is less than or equal to θ_{thresh}, then the vector $\hat{TR}_j$ is located inside of the cone and the target T has been occluded by N_i, and is considered no longer visible by R_j.

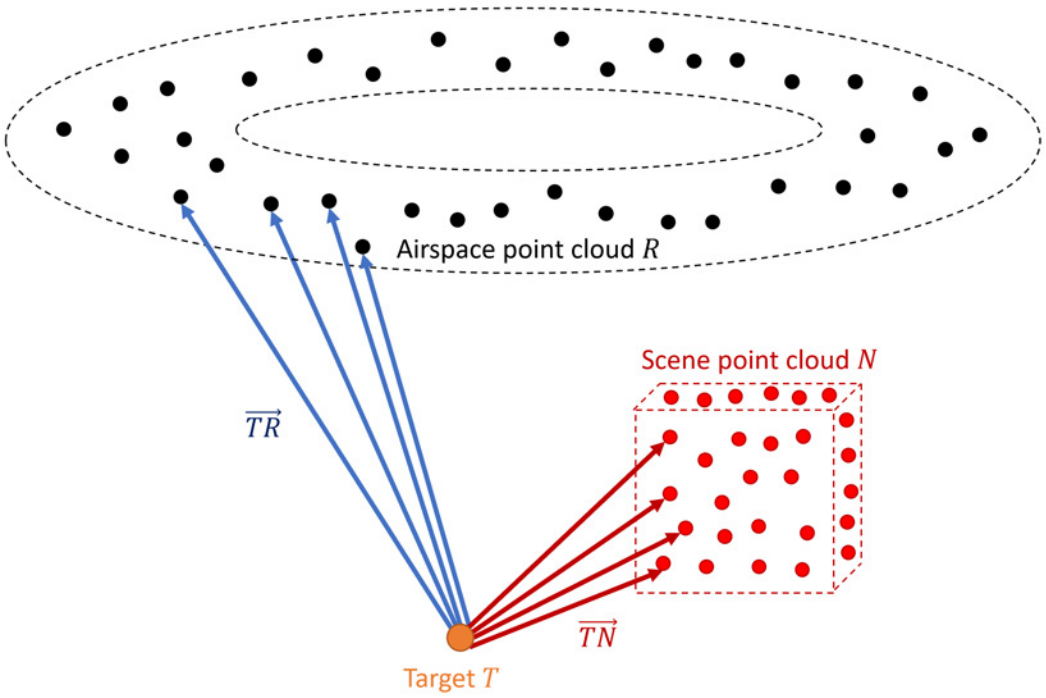

Figure 3.6: A diagram that shows how the vector sets are created between the target and the two point clouds. Note that for the purpose of simplifying the visualization not all rays are drawn.

In order to find the set of $\hat{TR}_j$ vectors that are not occluded, all of the angles for a particular point R_j need to be greater than the threshold. Therefore, the *all* function is used to compute the boolean logical values for every R_j where *false* represents occluded and *true* represents non-occluded. After computing all $(\boldsymbol{\theta} > \theta_{\text{thresh}})$, resulting in a boolean vector of size $J \times 1$, this vector can now be used to group or separate the set of flight path points R_j into either one of two point sets, *occluded* or *non-occluded*, by using logical indexing. After grouping the flight path points into two sets, it is sometimes referred as the *occlusion point cloud* in its entirety. The *occluded* points represent the positions where the RPA does not have line of sight, whereas the *non-occluded* points have visual or line of sight to the target. Therefore, for an RPA to track and surveil a target, the RPA should fly in a holding pattern determined by the set of non-occluded points.

$$\hat{TN_i} = \frac{N_i - T}{||N_i - T||_2} \tag{3.1}$$

$$\hat{TR_j} = \frac{R_j - T}{\left\|R_j - T\right\|_2} \tag{3.2}$$

$$\cos(\boldsymbol{\theta}) = \hat{\boldsymbol{TN}} \cdot \hat{\boldsymbol{TR}}^{\mathrm{T}} \tag{3.3}$$

3.2.4 Surface reconstruction and point resampling .

Due to some limitations of SfM, the output point cloud contained artifacts such as noise, outliers, and the most problematic issue being missing data. Missing data indicates that a surface, object, or geometric feature did not become well-represented by points in the point cloud. Therefore, these surface reconstruction and point resampling steps were added to mitigate the missing data issue. The issue was especially apparent in the point clouds produced by the MQ-9 imagery data where there are missing points, for example such as some of the points representing where a rooftop should be present. An SfM point cloud typically does not obtain the fine geometric features that a lidar system could collect. However, the point cloud should at least represent the general surfaces, but SfM failed to do that adequately as evidently shown in Fig. 3.3. Unfortunately, point upsampling techniques typically do not resolve this problem because such techniques usually generate new points by varying the existing points, which do not have local awareness of where the voids of missing points are located. Fortunately, surface reconstruction was used to mitigate the issue of missing data by fitting a 2D surface over the point cloud, thereby bridging the voids of missing points [6]. The 2D surfaces were represented by a set of vertices and faces. Then the surface was point sampled, generating a new point cloud based on the surface that bridged the voids. However, note that this process is also very manual, requiring a human in the loop to make interpretations on the quality of the representation of the surface to

the point cloud. Surface reconstruction also introduced artifacts discussed later in section 4.2.6.

The specific surface reconstruction technique is called Screened Poisson Surface Reconstruction [38]. The default parameters were used: reconstruction depth 8, adaptive octree depth 5, conjugate gradients depth 0, scale factor 1.1, minimum number of samples 1.5, interpolation weight 4, and Gauss-Seidel relaxations 8. However, note that this produced a large sheet that bridges the point cloud, forming faces that represent a ground that were not represented by points in the point cloud. For example, the left image in Fig. 3.5 shows primarily a singular sheet reconstructed over three separate buildings. In this example, the formation of the surface was acceptable, but there are cases where the surface reconstruction was an inaccurate fit of the true geometric features. The faces that were not representative of geometric features in the original point cloud were selected for deletion using an edge length threshold. Then, the surface was further cleaned by removing isolated faces, resulting in the surface that later becomes point sampled to convert the information back into the point cloud domain.

After the surface has been constructed, the surface mesh was point sampled in order to obtain a resampled scene point cloud. The specific sampling technique used is called Poisson-disk sampling with Monte Carlo oversampling of 20 and best sample pool size of 10, and radius variance of 1 [13]. This resampled scene point cloud contained the points from sampling the surface that includes the bridges that were formed to repair the void space on rooftops. The resampled point cloud was exported to be mathematical manipulated using the conic ray tracing method. The reconstruction and resampling process is shown in Fig. 3.7.

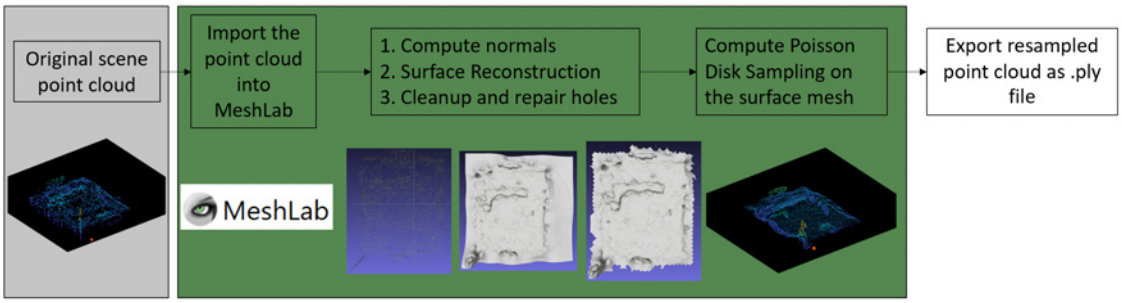

Figure 3.7: This process created the resampled point clouds by using surface reconstruction to bridge voids in rooftops and point sampling to return the surfaces into points.

3.3 Performance Metrics

3.3.1 Reference occlusion point cloud.

In order to measure the quality of the solution to the occlusion problem, there needs a performance standard to measure against. One way of assessing the correctness of the occlusion point cloud was to compare it to the truth occlusion point cloud. However, because the truth data was not collected for the scenes collected at White Sands, a reference occlusion point cloud was created to compare against the result from the conic ray tracing method. The reference was created using true graphical ray tracing on a surface reconstruction of the original point cloud scene with a process shown in Fig. 3.8. First, the original point cloud scene was imported into MeshLab, the normals were estimated by computing every point using ten nearest neighbors, and the Screen Poisson Surface Reconstruction was computed using the same default parameters described in section 3.2.4. The surface mesh was cleaned up by manually deleting the faces of which were not representative of the original scene, and then the mesh was exported in STL (stereolithography) format.

To perform true ray tracing, the surface mesh was imported into MATLAB. Using the *raytrace* function, it took the inputs transmitter sites, receiver sites, propagation model, and the map. The "transmitter" for this purpose is the location of the ground target in cartesian coordinates, and the "receivers" are the many (10,000) flight path points shaped in the

circular holding pattern in the airspace. The propagation model was set to "raytracing-image-method" where it was important to set the maximum number of reflections to zero in order to produce only the rays that have direct line of sight. The map was the surface mesh of the scene that was exported from MeshLab in STL format. The *raytrace* function performed ray tracing from the transmitter to all receivers, i.e. the ground target to all flight path points in the airspace, and then produced a set rays that have direct line of sight. Using boolean operations, the set of rays was converted into logical indices in which 1 represents occluded and 0 represents visible–the same convention as in the conic ray tracing method. Then using the flight path points with logical indexing, the points were separated into occluded and visible point sets, which ultimately form the reference occlusion point cloud. This reference occlusion point cloud was computed using true ray tracing, in which its rays are casted and blocked by a surface, was considered generally more truthful than conic rays that approximate light or line of sight in the shape of cones. The reference occlusion point cloud was meant to be an approximation to the truth data since the truth data is not available.

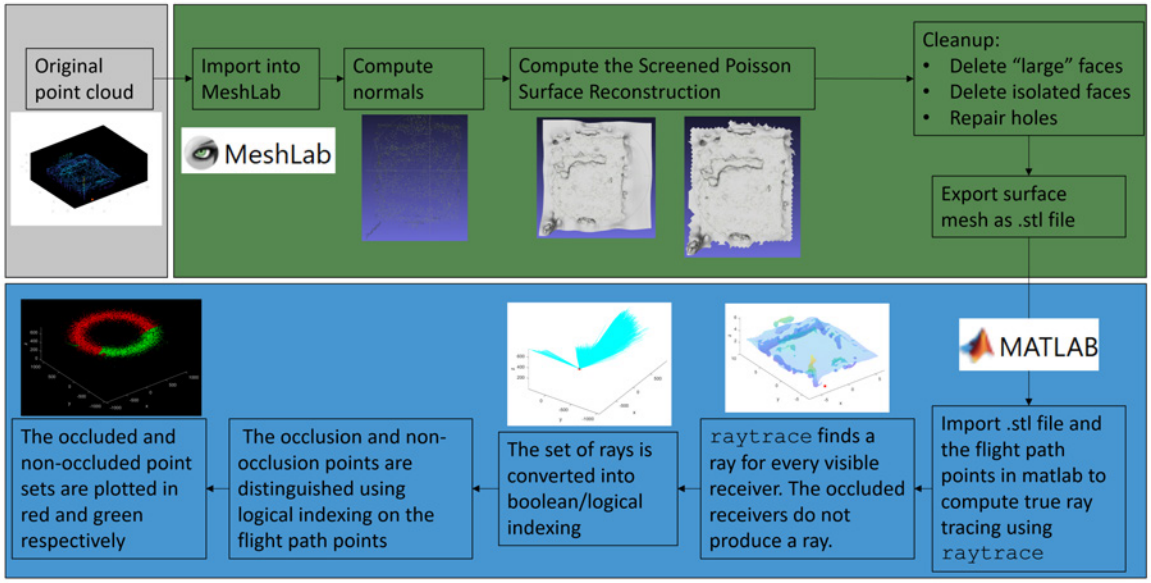

Figure 3.8: The process to create the reference occlusion point cloud using true graphical ray tracing on a surface reconstruction of the scene.

3.3.2 Confusion matrix.

The performance of the occlusion point cloud from the conic ray tracing method was then measured by computing the confusion matrix using the reference. An overview of the process is shown in Fig. 3.9, beginning from the original scene and ending with a confusion matrix. The confusion matrix compares the predicted result with the truth, but in this case the reference was considered the truth. The convention was positive means the target was occluded and negative means the target was visible described in Table 3.1. The confusion matrix measured the false positives and false negatives (type I and type II error respectively), which were measures of incorrect predictions. The conic ray tracing method was expected to get many more type I errors than type II errors due to the expanding nature of a cone at farther distances away from the vertex, which has a larger circular cross-section that selects a larger footprint of flight path points. Cones are an approximation, which selects more points to be considered occluded but when compared to the reference actually are not occluded.

The accuracy of the prediction was also computed from values from the confusion matrix using the standard definition of accuracy, which is the sum of the true positives and the true negatives then divided by the total sample size. The sample size was the number of points in the occlusion point cloud, which was comprised of 10,000 points. The accuracy was a simple measure to compare how well conic ray tracing predicted correctly.

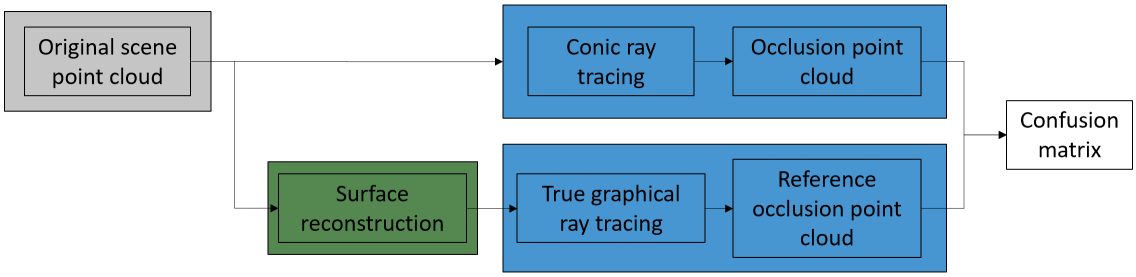

Figure 3.9: The process for evaluating the performance of the conic ray tracing method by using the reference point cloud to compute the confusion matrix.

	Predicted visible (-)	Predicted occluded (+)
Truly visible (-)	True negative (TN)	False positive (FP)
Truly occluded (+)	False negative (FN)	True positive (TP)

Table 3.1: The convention for the confusion matrix where positive means the target is occluded and negative means the target is visible. True means the point was correctly identified and false means incorrectly identified when compared to the reference point cloud. False positive and false negative are also known as the Type I and Type II error respectively.

3.3.3 Computational runtime.

The computation time of the conic ray tracing method was also measured and compared to that of the true ray tracing using MATLAB's built-in stopwatch timer *tic* and *toc*. For the conic ray tracing method, the runtime was measured from computing the

points into the vectors, to solving for the angles using the dot product, and to the boolean operations that solved for the logical indices for the occlusion points. For generating the reference point cloud using the true ray tracing, the runtime was measured on the *raytrace* function and its subsequent boolean operations that computed its occlusion logical indices. The idea was to measure the runtime for only the lines of code that computed the logical indices that is later used by data slicing to select the occluded points, so the set of logical indices were effectively the solution to this occlusion determination problem. Any runtime allotted for reading inputs, the actual data slicing to group the point sets, and any plotting were not included as part of the runtime measurement. The runtime results are discussed in section 4.4.

IV. Results and Discussion

Chapter 4 implemented the methodology to solve the point cloud occlusion determination problem by using the conic ray tracing method described in Chapter 3. The conic ray tracing method identified the occluded and non-occluded locations of the circular holding pattern in the airspace, resulting in the occlusion point cloud. The method was tested on eight scenario cases in which the ground target was located in various situations in a scene and their resulting occlusion point clouds were discussed. The resulting occlusion point clouds were compared against their corresponding reference occlusion point clouds created by using true graphical ray tracing on a surface reconstruction of the scene. The performance was measured by computing the confusion matrix between the result and the reference. The computational runtimes were also measured and compared between the conic method and the true ray tracing algorithm.

4.1 Data Pipeline

4.1.1 Video and time-sequenced images.

Stated previously in Chapter 3, Capt Aubrey Olson conducted a flight test and collected data using an MQ-9 Reaper at White Sands Test Facility in New Mexico and used the sensor on the sniper pod to capture raw video data of various ground scenes [56]. Fig.4.1 shows images of the videos of the six scenes captured and were respectively named *Complex1*, *Complex2*, *Intermediate1*, *Intermediate2*, *Simple1*, and *Simple2* according to relative complexity of geometric features of the primary building. The subjects of the videos were mainly buildings, cars, parking lots, and roads that were located in desert terrain environment. Most of the video footage was in color; some video footage was in infrared imaging, where black is cold and white is hot. The videos were recorded at 30 frames per second (FPS) and were stored in 24-bit RGB color in 720 by 480 resolution.

The videos were then parsed into a sequence of individual image files at a recording ratio of 60, meaning it saved one image out of every 60 frames. For a 30 FPS video, this meant it saved an image every two seconds. Sampling the video at this rate produced around 120 images for an approximately four-minute orbit flight around each building. Each set of images were then imported into Regard 3D to each produce a point cloud.

Figure 4.1: Samples from the six videos captured and collected by a MQ-9 at White Sands Test Facility, named *Complex1*, *Complex2*, *Intermediate1*, *Intermediate2*, *Simple1*, and *Simple2* respectively.

4.1.2 Structure from Motion (SfM).

The point clouds were generated using SfM on raw imagery data collected on MQ-9 flight tests. Each set comprised of approximately 120 pictures representing one scene were imported into the Regard 3D software one set at a time. The software produced raw point clouds using A-KAZE for keypoint detection, FLANN for keypoint matching, incremental SfM for triangulation, and CMVS/PMVS for point densification as described in section 3.2.2.2. Then the point clouds were processed by downsampling to 25% of the initial number of points, georectifying by rotating the points using direction cosine matrices with manual angle inputs, and filtering out points by bounding to a smaller domain using cutoff values [56]. This processing step simplified the point clouds for the purposes of this research. After processing, the outputs are shown on the left column in Fig. 4.2, where the color of the points signify relative altitude, i.e. a point with a low z-value was represented by the color dark blue and a high z-value was represented as yellow. The color in the point cloud was meant only for visualization purposes. However, note that the orange point representing the ground target was synthetically placed into the scene and was not part of the SfM output. It is further discussed later in section 4.1.3. There were observations of six building complexes, resulting in six point clouds.

On the right column of Fig. 4.2, these point clouds were the resampled versions of all six scenes using the process previously described in section 3.2.4 and were displayed beside their original point clouds for comparison. Generally, the resampled point clouds had points with a smoothing effect and appeared more uniformly spaced than that of their original points, due to Poisson-disk point sampling technique used on the surface construction of the original points. However, this process of producing a resampled scene can delete existing features, evident in the *Intermediate1* scene where a few of the parking lot lamps were not retained in the resampling. The resampling process also created unwanted artifacts in all of the resampled point clouds by extending edge features, and this is especially apparent

for case 6 discussed in section 4.2.6 where both the resampled result and the reference point cloud greatly differed from the result that used the original point cloud. And the differences were ultimately caused by the surface construction and point sampling process to misshapen edge features. The original intent of the resampled point cloud was to bridge voids in the points, particularly to repair the "holes" in rooftops, but in doing so introduced the edge feature artifacts.

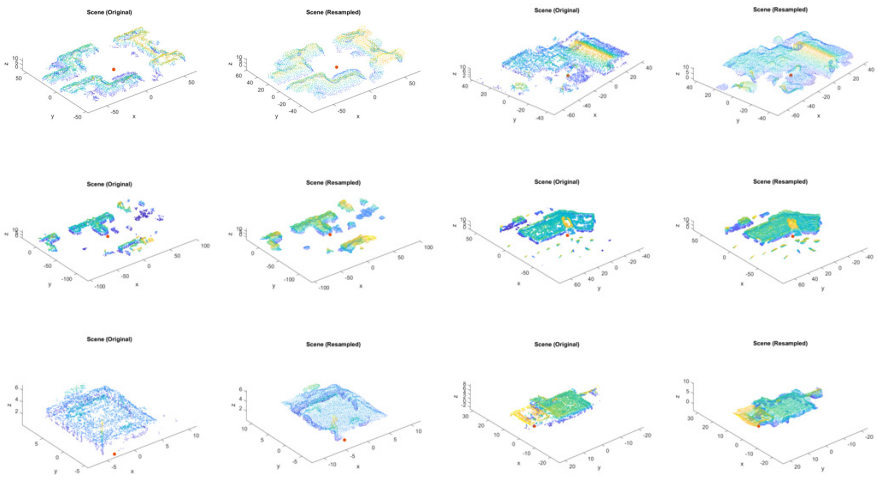

Figure 4.2: The original scene point clouds (left) are compared against their corresponding resampled point cloud (right).

4.1.3 Conic Ray Tracing.

After using SfM to produce the point clouds, the ground scene was now represented in 3D, and in this 3D representation the spatial information was used to define a relationship between a ground target and its surroundings to ultimately determine the viewing occlusion to an RPA flying overhead. The point cloud provides a new piece of information–spatial information, which is more than what the 2D imagery from the videos provide on their own. In these six scenes, neither a human actor nor a mannequin were present to represent the ground target. However, videos were collected later on that included human actors but

were not used in these experiments. The videos that include human actors are described in future work in section 5.1.5. Since the original imagery did not contain a target, the ground target was synthetically inserted into the scene as a point representing its centroid. For these experiments, the target was represented by a single point to simplify the occlusion detection computation. According to data on real-world body mass, the centroid of a human adult male is approximately located near the abdomen or the belly button [25]. According to the CDC, the average height of a human adult male is 175 cm [21]. Therefore, the centroid of an average human adult male body was assumed to have a height of 100 cm. Assuming the ground plane is located at $z = 0$, then the target was simply placed at a height of 100 cm, or a z-value of 1 meter in respect to the scale of the point cloud data. The target was placed on the xy plane in positions according to locations with spatial features of the scene that deemed interesting sources of occlusions, such as against a wall, within a corner, or under an overhang.

4.2 Test Cases

The target was placed in different locations in the different scenes to test how the conic ray tracing method would perform. The first case was the base case where the target was placed in an open area meant to be fully visible by the RPA. The second and third cases had the target located against a short wall and in another case adjacent to a long wall along a building. For the fourth and fifth cases, the target was placed in an inward two-walled concave corner and an exposed convex corner, respectively. For the sixth case, the target was enclosed by three walls in a double corner. For the seventh case, the target was placed under an outdoor overhang (an exterior roof eaves or soffit). For the eighth case, the target was placed under a different overhang (a gas station canopy). A table of these cases is shown in Table 4.1, which also describe their expected visibility or occlusion. These cases were tested using conic ray tracing on the original point clouds, then on the resampled point

clouds, and both results were measured against the reference point cloud. The confusion matrix and accuracy were reported in Table 4.2.

Case	Location of target	Expected difficulty of maintaining visual custody
1	Open area	Easy–visible in most or all of the holding area
2	Against a short wall	Easy–visible in more than 50% holding area
3	Against a long wall	Medium–visible in more than 50% holding area
4	Concaving corner	Difficult–visible in less than 50% holding area
5	Exposed convex corner	Easy–visible in more than 50% holding area
6	Double corner	Difficult–visible in less than 50% holding area
7	Under a roof overhang	Difficult–visible in less than 50% holding area
8	Under a gas station canopy	Difficult–occluded in most of the holding area

Table 4.1: A table describing the cases of the ground target located in interesting locations and an estimate of the expected visibility or occlusion of the target.

The following figures show the results that demonstrated the eight test cases where a target was placed in one of the six scenes, located in positions that produced interesting occlusions. Each figure contained six images. On the top row, the first two images are the original scene and resampled scene respectively, in which both scenes include the location of the target inside the scene, indicated by a larger orange point marker. The third image on the top row is a photograph of the scene for the reader to see a natural view of the scene. The resampled point cloud was produced using a surface reconstruction and point sampling using the process explained in section 3.2.4. And this resampling process was meant to produce a new point cloud to mitigate missing data by the point cloud generation process such as nonexistent "holes" in the rooftops because the rooftops were not adequately represented in some of the original scenes. On the bottom row, there are three images

of occlusion point clouds, in which their viewing angle azimuth were adjusted to match the scenes above, hence the angled coordinate frame. The left and middle are the occlusion point clouds determined by the conic ray tracing method that correspond to the original and resampled scenes respectively. An example of an occlusion point cloud viewed in a 3D view is shown in Fig. 4.3. The reference (a.k.a. "truth") occlusion point cloud shown on the right was created by computing true graphical ray tracing on the surface reconstruction of the original point cloud. The reference was used to measure the performance of the results produced by the conic ray tracing method because the corresponding truth data for the occlusion point clouds was not available. The ground target was located at the same coordinate location in the three scenarios (original, resampled, and reference).

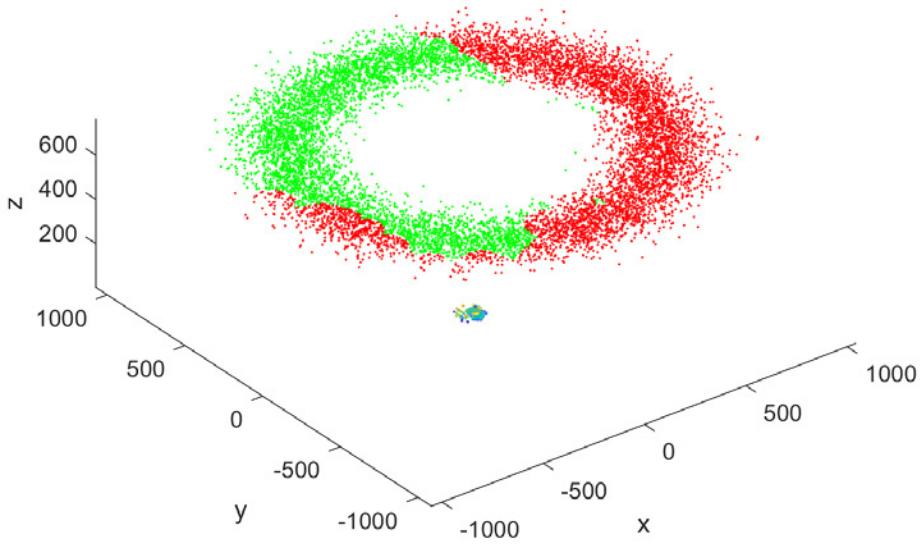

Figure 4.3: An example of an occlusion point cloud in the same figure as the ground scene shown in 3D. The occlusion point clouds later are shown in 2D from bird's eye view to flatten the skew caused by the 3D view.

4.2.1 Case 1: Base Case.

The first case was the base case, where the target was placed in an open area and was expected to be fully visible by the RPA. The occlusion point cloud was expected to show that the target was fully visible from all the flight path locations, indicated by the color green, and indeed it did, shown in green by the point cloud in the shape of a ring for the circular holding pattern in Fig. 4.4. Using the *Complex1* scene, the target was placed at $(0, 0, 0.1)$. The occlusion point cloud on the left used the conic ray tracing method described in section 3.2.3 on the original point cloud, and the middle image used the same conic ray tracing but on the resampled point cloud. The image on the right used ray tracing on the "cleaned" surface reconstruction of the point cloud scene, which is considered the truth values, and indeed all three results match with an accuracy of 1.

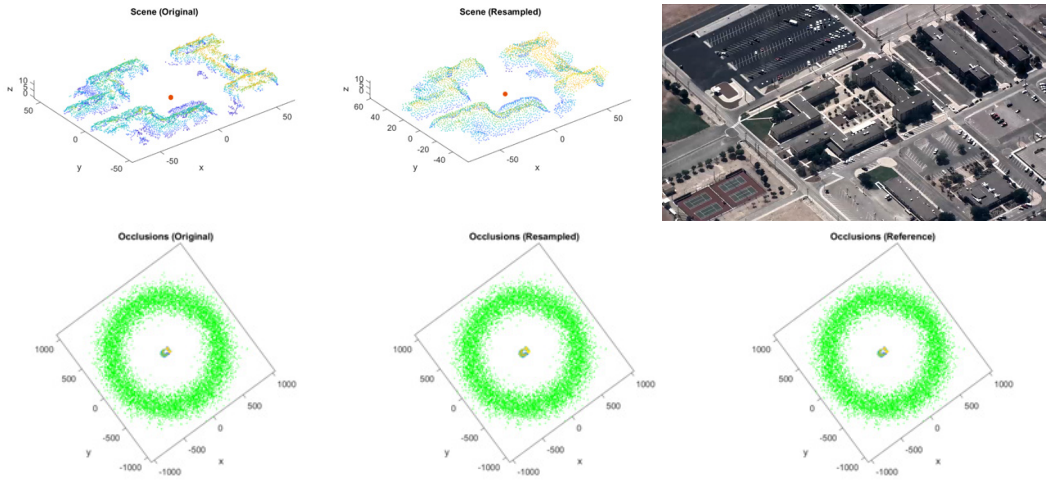

Figure 4.4: This figure shows the base case of the algorithm using the *Complex1* scene, the target was placed in an open area at $(0, 0, 0.1)$. The occlusion point cloud in the shape of a ring located above the scene represents the flight path of an RPA, and the color green indicates that the target is visible by the RPA at the point's location. The left image shows the coordinate geometry method used on the original point cloud scene.The middle shows the original geometry method used on the resampled point cloud. The image on the right shows the truth occlusion point cloud using ray tracing a surface reconstruction of the point cloud scene.

4.2.2 Case 2: Short Wall.

In the second case, the target was placed against a short wall at $(0, -40, 0.1)$ in the *Intermediate2* scene, shown in Fig. 4.5. The algorithms determined that the points representing the wall created an occlusion, and the resulting flight path points were labeled occluded accordingly, indicated by the color red. The resulting occlusion point clouds show the occluded and non-occluded areas in red and green areas respectively, matching human intuition without major issues in this case. To note some subtle differences, the original scene has some visible locations mixed into the occluded area, in other words green mixed into the red. The cause of this was due to missing points needed to represent the building features, and the missing points were caused by SfM during the generation of the original point cloud, which needs higher resolution imagery from different viewing angles as input to properly characterize the building features. This issue was addressed by creating a surface reconstruction and point sampling the mesh to generate a new resampled point cloud. Compared to the original scene, the resampled scene produced an occlusion point cloud more accurate to the reference. The result from the resampled scene is shown in the middle and the truth is shown on the right in Fig. 4.5. However, the resampling method has other issues discussed later in case 6. Other than the short wall, the target was very visible in an open area because the other points in the scene are not close enough proximity to the target to affect the line of sight from at the standard altitude of 7,315 m. The original and resampled results reported an accuracy of 0.93 and 0.95 respectively.

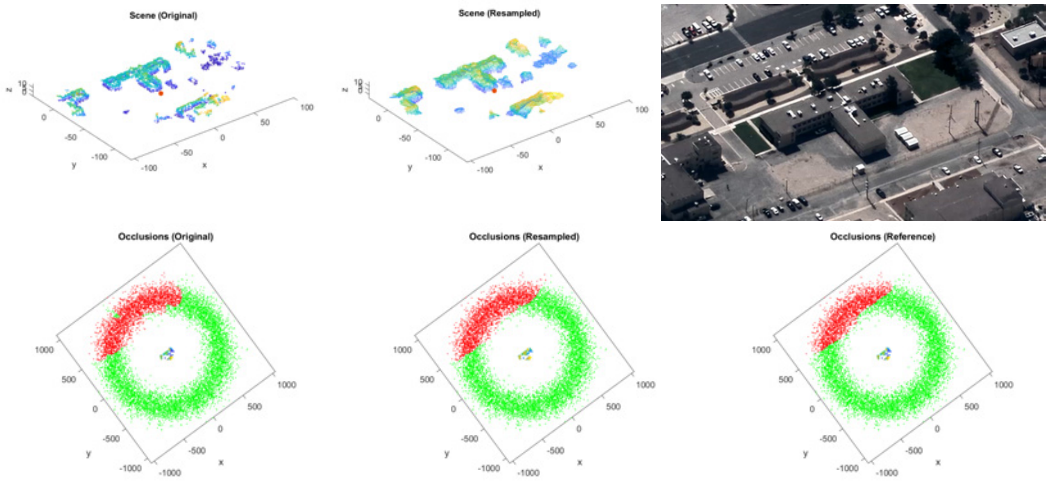

Figure 4.5: Case 2 placed the target in the *Intermediate2* scene adjacent along a short wall. The target was occluded by the short wall but was otherwise visible from all other directions.

4.2.3 Case 3: Long Wall.

For case 3, the target was placed in a similar situation to case 2 but along a longer wall. The target was placed in the same scene *Intermediate2* but behind the 'T' shaped building at $(0, 10, 0.1)$. The long wall occluded the target in the directions and areas that are intuitively expected similar to case 2. For this case, the accuracy for the conic ray tracing method was 0.87, and using the resampled scene the accuracy increased to 0.96. Using the resampled point cloud generally increases the accuracy. And similar to case 2, the resulting occlusion point clouds have no major issues. The scene point clouds and the occlusion point clouds are shown in Fig. 4.6.

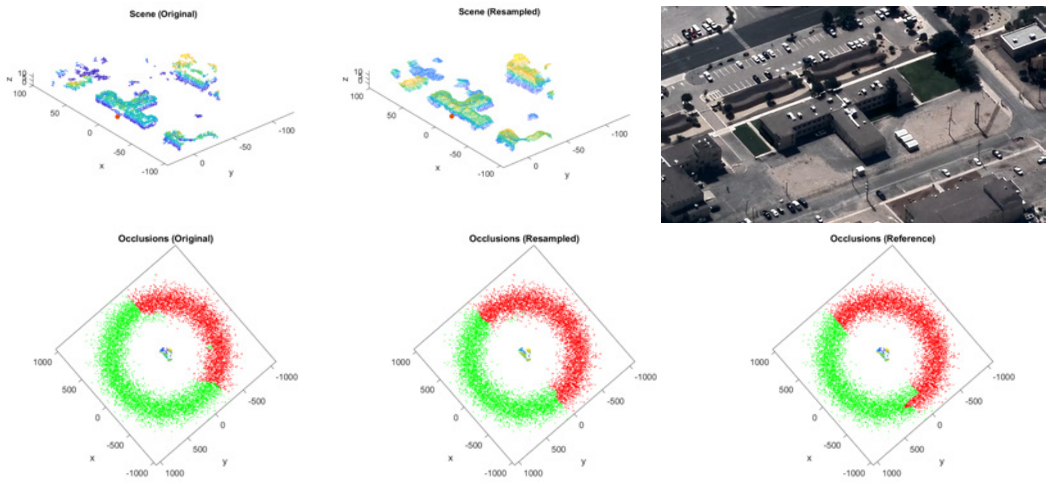

Figure 4.6: Case 3 placed the target in *Intermediate2* behind the 'T' shaped building along the long wall, which occluded the target in a manner intuitively expected and similar to case 2, but the long wall occludes a larger portion of the occlusion point cloud compared to the short wall in case 2.

4.2.4 Case 4: Concave Corner.

For case 4, the target was placed in the *Intermediate1* scene near the building entrance in an openly obtuse concave corner at $(-20, 0, 0.1)$ with a building entrance canopy overhanging above. The corner was formed by two intersecting walls of the single building due to its architecture, forming a concaving obtuse approximately 120 degree angle as shown in Fig. 4.7. One interesting thing about this case was that the canopy at the building entrance was not fully characterized in the point cloud, which is apparent in its resulting occlusion point cloud on the left image in figure. When comparing with the reference, using conic ray tracing on the resampled point cloud seemed to overcompensate for the occlusion caused by the canopy. However, it is important to note in this case that the truth values are also suspect because the surface reconstruction did not fully characterize the missing piece of the canopy. To address this issue, a more accurate point cloud scan or a computer-aided drafting (CAD) model of the building is needed. The conic ray tracing method worked

reasonably well at finding occlusions but when provided a poor point cloud representation the algorithm by itself does not account for the missing geometric features.

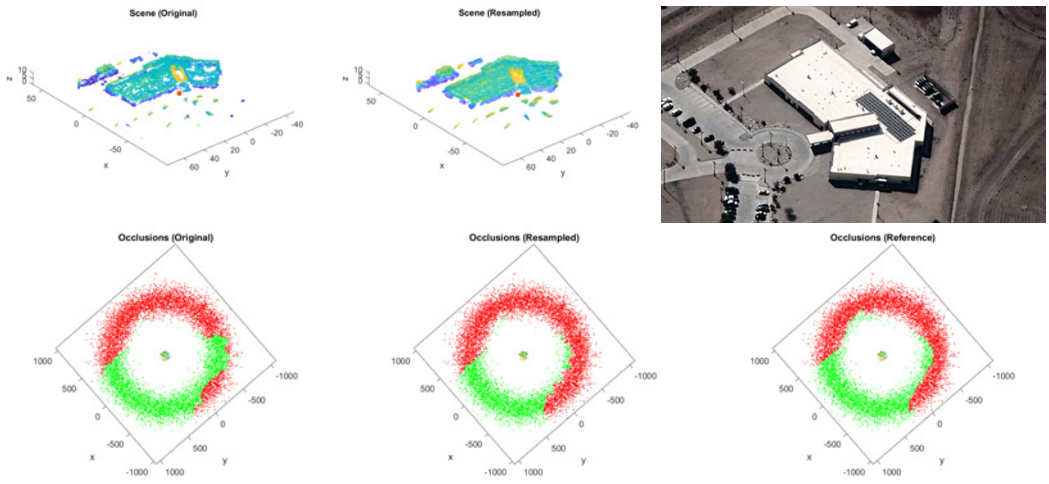

Figure 4.7: Case 4 placed the target in the *Intermediate1* scene to simulate a situation where the target is occluded within a concaving corner. The original scene point cloud has an issue with the point cloud poorly representing the real-world geometric features due to missing points. Resampling the point cloud can somewhat address the issue but also can add undesired artifacts.

4.2.5 Case 5: Convex Corner.

Case 5 is another case that placed the target in a corner, but a convex corner for this case, using the same building from the *Intermediate1* scene. The target is located at $(-17, 42, 0.1)$, which is near a corner of the building and is shown in orange in the point clouds in Fig. 4.8. The target was placed where it is visible along both of the walls intersecting at the corner. The resulting occlusion point clouds determined that the area above the building is where the target was occluded by the building. Because the target was located at a convex corner, an RPA would have much greater visibility on the target than compared to the concaving corner test case. The accuracy of the occlusion point cloud for the original and resampled scenes were 0.98 and 0.97 respectively. Note that the accuracy

for the resampled scene is worse than that of the original scene, which goes against the trend of the other cases that the result from the resampled scene typically performed better than the original result. There is not a good explanation for this inconsistency because the resampled point cloud scene matches closely with the surface reconstruction, so the resampled point cloud should produce a result similar to the reference point cloud. One possible reason is that the resampling resulted in larger point structures and the conic ray tracing method had ray traced the larger point structures, resulting in more points selected for occlusion, and thus resulting in an occlusion area larger than when the conic ray tracing was performed on the original point cloud. An RPA would have the greatest amount of visibility of the target when the target is located at one of the convex corners than any other position around a building's perimeter.

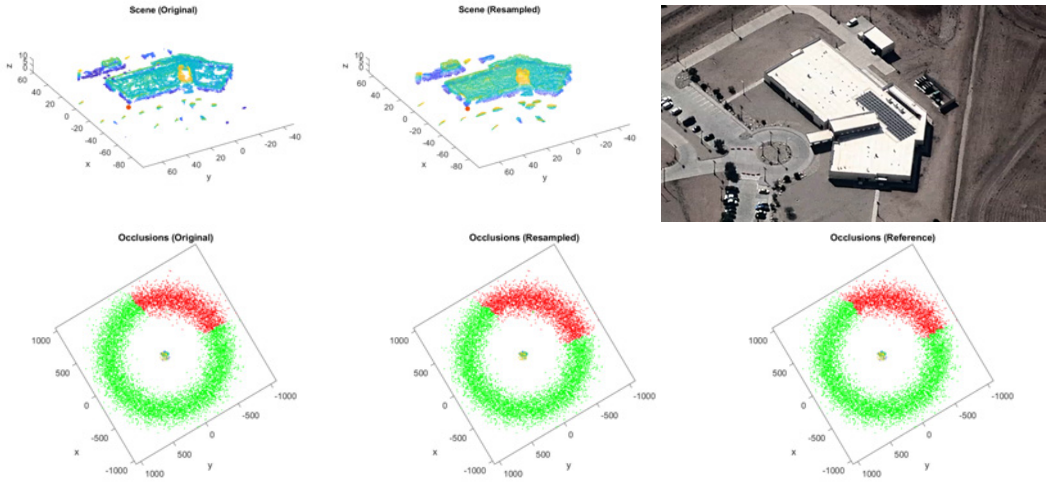

Figure 4.8: Case 5 placed the target in the *Intermediate1* scene, in contrast to case 4, to be located at a convex corner. The occluded area was determined above the building relative to the target located at the convex corner. The RPA has the greatest amount of visibility of the target when the target is located at one of the convex corners than any other position around a building's perimeter.

4.2.6 Case 6: Double Corner.

Case 6 was an attempt at testing the "double corner" formed by the intersection of three walls, such as at a dead-end alley. The target was placed in the *Complex2* scene at $(-15, -15, 0.1)$, which from the video appears to be at a side entrance or gated loading dock of the building. From looking at only the point cloud, it was not clear if a target can stand in that area because SfM did not define it well, but from the video it was apparent the area is capable of and generally intended for people to be able to access and walk in that area. The area was enclosed by effectively three walls with two intersections, forming two corners. The resulting occlusion point clouds were from using the original point cloud, the resampled point cloud, and the reference occlusion point cloud is shown from left to right respectively in Fig. 4.9. This was a case where all three results each have their own issues. First, the reference created from using a surface reconstruction process that tends to extend edge features by adding faces that are not representative of the point features in the original point cloud and are not be representative of the true real-world geometry either. This means that this reference occlusion point cloud was invalid at least for this specific placement of the target. The extended edge features on the surface mesh caused this occlusion can be seen in the appendix.

The second issue, the occlusion point cloud from the resampled scene has an apparent "polka-dot" effect caused by multiple factors: the surface reconstruction extended the edge features, then the Poisson disk sampling sampled those surfaces resulting in a larger footprint, and conic ray tracing determined the sparse points in that local area as occlusions indicated in red. It has a "polka-dot" pattern because of the uniformly sampling by Poisson disk sampling. The intersections between cones and a plane are ellipse shapes, creating the polka-dot effect seen in the middle image in Fig. 4.9.

The third issue, the original point cloud was produced by SfM, which has inherent inaccuracies in the form of missing points due to using low quality of the raw video

imagery in SfM. The result from using the original scene was likely the most valid out of the three results, but this is only speculation from intuition. While it may be difficult to judge the validity of this test case, it exposed limitations of the methods. One of the fundamental issues is the poor quality of the original point cloud characterize the true geometric features of the real-world scene. Surface reconstruction and resampling was used to bridge "holes" in the point cloud using a surface mesh and then point sampling the mesh to recover geometric information. But this process also introduced error by reconstructing surface fits that are improper since the point cloud was missing points to begin with. There can be multiple approaches for future work to address the issues, including using a higher resolution sensor to capture video, using a better point cloud generator algorithm, and developing an algorithm for finding and localizing holes to repair in a point cloud.

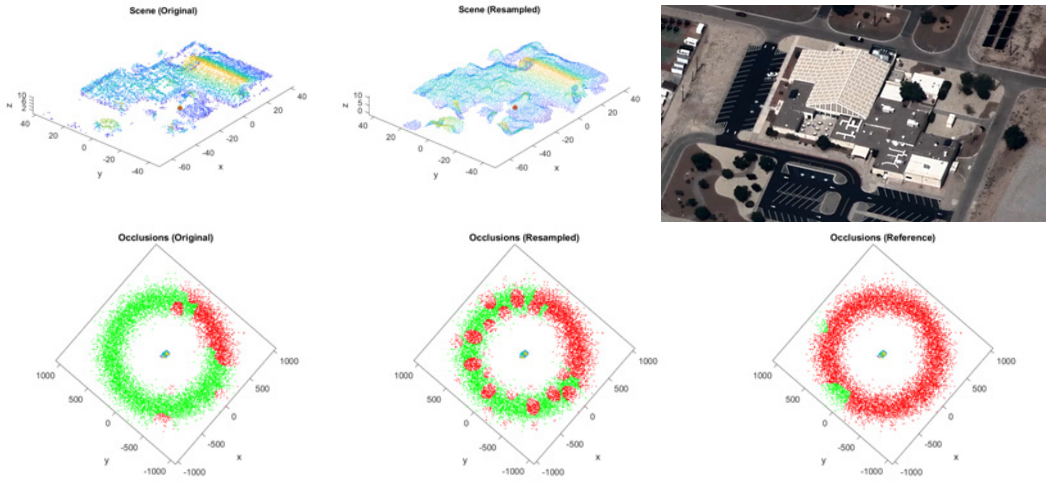

Figure 4.9: Case 6 placed the target in the *Complex2* scene located at a double corner area formed by three intersecting walls. The results from the original scene, resampled scene, and reference have occlusion point clouds that are very different from one another (on the left, middle, and right respectively). The issues are caused by different factors, such as sampling error inherent to the SfM point cloud, improper fitting in the surface reconstruction, and artifacts created from sampling a surface mesh.

4.2.7 Case 7: Roof Overhang.

For case 7, the target was placed at $(-3, -5, 0.1)$ in the *Simple1* scene under an overhanging roof edge called an eaves or soffit. This scenario was meant to test how well the conic ray tracing method can determine the occlusions caused by overhanging obstructions, where in this case the target was still visible on one side. In Fig. 4.10, conic ray tracing was computed over the original point cloud (first column) and resampled point cloud (second column), then compared to the reference point cloud created using true ray tracing (third column). On the left, the result from computing the original scene had a few points that formed corresponding patches of occlusion. In the middle, the resampled point cloud had removed the spurious points, creating a smoother and more uniformly spaced point cloud scene, resulting in contiguous pieces of occlusions in the airspace. Note that this is in contrast to case 6, where the number of spurious points increased in the resampling process due to the surface formed to represent the spurious points, and Poisson disk sampling resampled the surface which generates a point set with a spatial occupancy typically larger than that of original set of points. In this case, the reconstructed surface was large enough to warrant its deletion in the surface clean up process from Fig. 3.7. Thus, the surface local to the spurious points was not present to be resampled and the spurious points were not retained in the resampled point cloud. However, it is possible that the spurious points represents an actual object present in the real-world but the SfM process generating the point cloud did not adequately capture its spatial features. Checking with the video, it was not clear whether there was an actual object there because it was obscured by the shadow. Though in many cases it is more desirable to determine a contiguous flight path that is free of patches of occlusions caused by spurious points. The resulting occlusion point clouds contained an occlusion point set that is greater than half the point cloud due to the roofing eaves blocking the view from the sides of the building, extending the two ends of the point set into a "horseshoe-shaped" occlusion set, which was intuitively expected.

Overall, the occlusions in the original and resampled scenes were in general agreement when compared with the reference with an accuracy of 0.89 and 0.92 respectively.

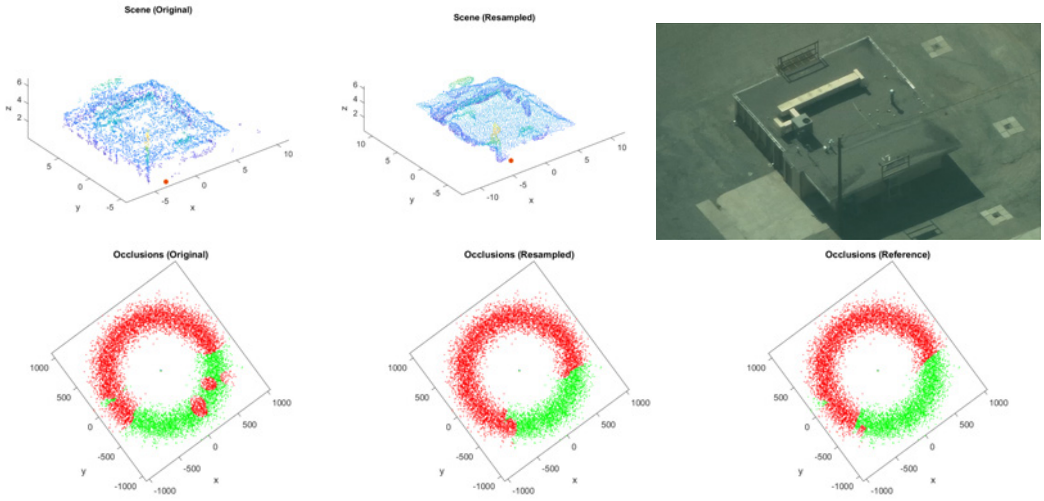

Figure 4.10: Case 7 placed the target in the *Simple1* scene to test for the occlusions caused by an overhanging roof edge called an eaves. On the left is the original scene, in the middle is the resampled scene, and on the right is the reference point cloud. The occlusions are generally in agreement with the reference point cloud.

4.2.8 Case 8: Gas Station Canopy.

For case 8, the target was placed at $(0, 15, 0.1)$ in the *Simple2* scene under a gas station canopy, where the target may be visible from the horizontal views but was expected to be fully occluded from the sky view due to the canopy. Note that the original point cloud has a "hole" in the roof due to missing points that should be there to represent the structure of the roof. This hole was apparent in the resulting occlusion point cloud where the points are colored green closer to the center of the ring, but instinctively this should be wrong because the target would be more occluded from view when located closer to the center of the canopy. The resampled point cloud repaired the hole and thus its resulting occlusion point cloud shows that the target has been fully occluded by the canopy. Despite these issue in the original scene, the resulting occlusion point clouds are in general agreement

of each other. The accuracy of the original and the resampled scene are 0.90 and 0.99 respectively when compared to the reference. However, the issue brought up with this test case is then how should the RPA determine a holding pattern if the target has been fully occluded. It could either lower the altitude and increase the holding radius to try to obtain a more horizontal view, or it could default to the standard circular holding pattern and wait for the target to show itself. The scene point clouds, the occlusion point cloud, and the reference is shown in Fig. 4.11.

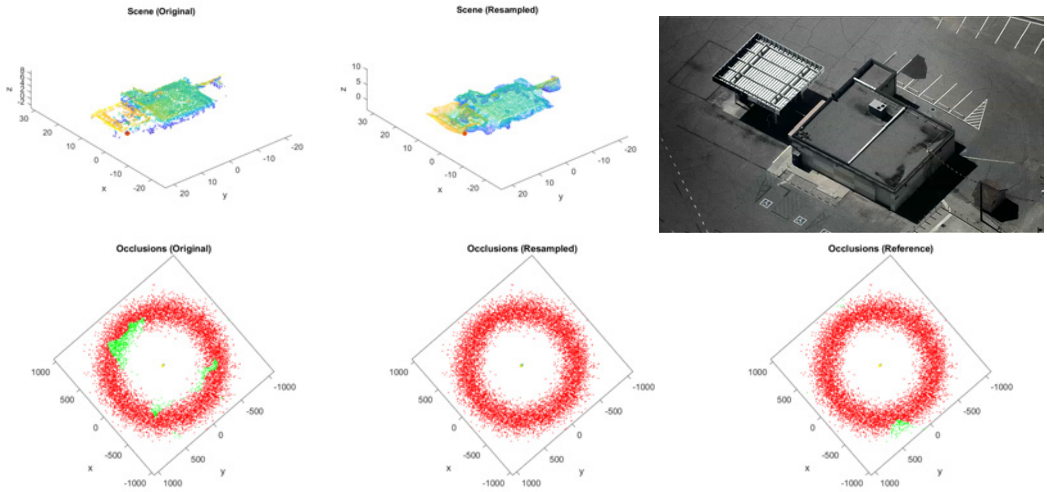

Figure 4.11: Case 8 placed the target in the *Simple2* scene under a gas station canopy, which is almost fully occluded as expected–at least at this particular altitude and radius of the holding area.

4.3 Confusion Matrix and Accuracy

The confusion matrix was computed for each of the eight test cases. A confusion matrix was computed between the result from the original scene and the reference. And then another confusion matrix was computed between the result from the resampled scene and the reference. The convention is that '1' is positive representing occluded, so true positive (TP) means a point was correctly occluded. Then true negative (TN) means truly visible, and so forth. The values from the confusion matrices are tabulated in Table 4.2 and

in Table 4.3 for the original result and resampled respectively. The counts add up to 10,000 because the airspace point cloud was generated using 10,000 points.

4.3.1 Results from the original scenes.

In Table 4.2, the eight test cases are tabulated where each case compared the conic ray tracing result with the reference by computing a confusion matrix. Positive means occluded and negative means visible. The actual positives are the sum of the TP and FN, and the actual negatives are the sum of the TN and FP. These values compare the imbalance of counts of positive versus counts of negative. The results from the original scenes generally have more type II error than type I error, which was an opposite trend in comparison to the results from the resampled scenes discussed below. This discrepancy is not explained or fully understood. Note that the values for case 6 were highly invalid due to the issues discussed in section 4.2.6.

Case	Act. pos	Act. neg	TP	TN	FP	FN	Acc
1	0	10000	0 (0%)	10000 (100%)	0 (0%)	0 (0%)	1
2	2181	7819	2141 (98%)	7138 (91%)	681 (9%)	40 (2%)	0.93
3	5467	4533	4138 (75%)	4533 (100%)	0 (0%)	1329 (25%)	0.87
4	6171	3829	5149 (83%)	3572 (93%)	257 (7%)	1022 (17%)	0.87
5	2724	7276	2518 (92%)	7269 (99%)	7 (1%)	206 (8%)	0.98
6	9546	454	2172 (23%)	454 (100%)	0 (0%)	7374 (77%)	0.26
7	6472	3528	5965 (92%)	2892 (82%)	636 (18%)	507 (8%)	0.89
8	9872	128	8949 (91%)	10 (8%)	118 (92%)	923 (9%)	0.90

Table 4.2: The confusion matrix from comparing the predicted occlusion points using conic ray tracing against the reference points created using true ray tracing. True positive means a predicted occluded point is truly occluded.

4.3.2 Results from the resampled scenes.

In order to mitigate the problem of having void space misrepresenting the point features of rooftops, a surface reconstruction technique was applied on the original scene point cloud to bridge the void space and then the surface was point sampled, resulting in a resampled point cloud. Conic ray tracing was performed on the resampled point clouds and then its occlusion point cloud was compared to the reference by computing the confusion matrix. For the resampled scenes, there were more type I error than type II errors, which is what was originally expected due to the expanding nature of the cones where at higher altitudes it would overselect points due to its larger footprint. It is not fully understood why the original scenes had instead resulted with higher type II error. The same conic ray tracing method was performed on the original and resampled scenes using the same 5 degree cone angle, and both scenes were compared against the same set of references. It should be expected that the conic ray tracing method overselects occlusion points in both original and the resampled, thus false positives (falsely occluded) should be higher in both scenes. But for the original scenes, this is not the case.

In Table 4.4, this table shows that the results from the resampled scenes generally perform with higher accuracy, i.e. it gets more true positive and true negatives correct than in the original scenes. However, this is also biased because the resampled scenes and the reference share similar steps in their construction, particularly the Screen Poisson Surface Reconstruction.

Case	Act. pos	Act. neg	TP	TN	FP	FN	Acc
1	0	10000	0 (0%)	10000 (100%)	0 (0%)	0 (0%)	1
2	2181	7819	2181 (100%)	7322 (94%)	497 (6%)	0 (0%)	0.95
3	5467	4533	5142 (94%)	4466 (98%)	67 (2%)	325 (6%)	0.96
4	6171	3829	6163 (99%)	3158 (82%)	671 (18%)	8 (1%)	0.93
5	2724	7276	2722 (99%)	6932 (95%)	344 (5%)	2 (1%)	0.97
6	9546	454	5085 (53%)	329 (72%)	125 (28%)	4461 (47%)	0.54
7	6472	3528	6472 (100%)	2726 (77%)	802 (23%)	0 (0%)	0.92
8	9872	128	9871 (99%)	0 (0%)	128 (100%)	1 (1%)	0.99

Table 4.3: The confusion matrices when comparing the resampled scene against the reference.

Case/Scene	Acc (original)	Acc (resampled)
1 Base case	1	1
2 Short wall	0.93	0.95
3 Long wall	0.87	0.96
4 Concave corner	0.87	0.93
5 Convex corner	0.98	0.97
6 Double corner	0.26	0.54
7 Roof overhang	0.89	0.92
8 Gas station canopy	0.90	0.99

Table 4.4: The accuracy from the original scenes were compared to the accuracy from the resampled scenes. The resampled scenes were generally improved in accuracy compared to the original scene.

4.4 Computational Runtime

The computational runtime was measured for the conic ray tracing method and for true ray tracing. Only the operations that compute the occlusion indices were measured, so any runtime used to read inputs or plot visuals were not included in the measurement. The eight cases were run through 10 iterations resulting with an average runtime of 0.47 seconds using conic ray tracing and 9.3 seconds for true ray tracing, meaning the conic ray tracing method was 19x faster than true ray tracing. However, note that true ray tracing is known to be a computationally expensive algorithm, and so this comparison is not meaningful. Instead, the computational runtime should be compared a different method such as Capt Olson's occlusion determination method or a cylindrical ray tracing.

V. Conclusions and Future Work

In conclusion, this thesis presented a method for determining visibility and occlusions of a ground target by ray tracing 3D point cloud data. The ground target, its ground scene environment, and the airspace were all represented by 3D point cloud data. The ground scene was generated using 2D imagery with a stereoscopic technique called Structure from Motion, and the airspace or flight path points were randomly generated in the shape of a ring to represent a circular holding pattern at a fixed altitude of a typical RPA. The benefit of using 3D point cloud data is that it can create spatial relationships between points by tracing rays. The occlusions are created between the ground target and objects in its environment. Using ray pairs between the ground scene and the airspace, the angle for each ray pair were computed using the dot product, and visibility and occlusions were determined by comparing the angles to a fixed cone angle. The conic ray tracing method solved for all the angles between all the scene points and all the airspace points to determine visibility or occlusion. The output from this conic ray tracing method was called the occlusion point cloud, which identified where in the airspace is the ground target visible or occluded. Because truth data was not available, the output was compared against a synthetic reference, which was created by using true graphical ray tracing on a surface reconstruction of the original scene point cloud. True ray tracing produced a different occlusion point cloud with which to compare against, i.e. a reference. The performance of the conic ray tracing method was measured by computing the confusion matrix between the two results to compare the accuracy as well as the type I error and type II error. The conic ray tracing method averaged a 0.81 accuracy and its computational runtime performed 19x faster than true ray tracing.

The advantages of this method is that it is fast because one of the primary computations for determining visibility was done by computing angles between the rays, which has a fast computational workload because of hardware acceleration of matrix multiplication on a

GPU. Some of the limitations of this process is that it still requires a human in the loop in this process as it is, such as using Regard 3D to produce a point cloud, synthetically placing a ground target into the scene, and using MeshLab to generate a surface reconstruction. These processes currently use GUIs for a human to make interpretations but should be converted into automated algorithmic processes that are either deterministic or probabilistic in order to reduce human intervention. More research in automating these processes are discussed in future work below.

5.1 Future work

5.1.1 Generate a point cloud scene from a single image.

The ideal framework should be able to represent the whole ground scene in the shortest amount of time computationally possible, such as by generating a 3D point cloud from a single image to represent the whole environment. One research article showed it is possible to generate a point cloud using a single image, however, it was able to do so only for the one primary object in the image, ignoring its foreground and background, and these objects typically had at least one axis of symmetry [19]. It may be possible to develop a framework that can generate a point cloud for an entire scene using a single image by using a machine learning approach to build a relationship between images and their corresponding point cloud model. Generating an accurate point cloud scene from a single image would be the ideal or near the ideal capability because the RPA would be able to understand the ground environment by using a single image–its most current image capture–without needing to process more images. Unfortunately, because this capability is not yet available, the 3D representation of the ground environment was achieved by capturing multiple images of the scene taken at different viewpoints and constructing the point cloud representation by SfM. It would be valuable to develop this capability because the framework could generate a scene immediately and thus would not be required to take four minutes to fully encircle a ground scene.

5.1.2 Using machine learning to segment a point cloud.

Another step for automating target detection would be to apply machine learning, specifically a deep learning network, to identify the target from its surroundings. Early machine learning methods were able to perform the machine learning task of classification, which means to classify or categorize an image using a finite set of predefined labels. Recently, machine learning approaches have extended to 3D point cloud data to categorize a point cloud into separate point sets. A point cloud machine learning model would need to be trained by labeled point cloud data by recognizing patterns in the spatial information associated with a label. Labeled data is important for the feature extractor to learn the geometric features of the shape of targets. Classification or categorizing a point cloud into point sets into classes is also known as semantic segmentation. A point cloud semantic segmentation framework could be used to segment the entire point cloud into two point sets: *target* and *non-target*. The targets are generally shaped as the bodies of human adult male represented locally in the point cloud, and the non-target points are everything else in the target's surroundings.

If it was important to identify more than one target, semantic segmentation is not sufficient to determine their individual instances from the class. Semantic segmentation is able to group all the targets into a single class, so the information contained in the data from the computer's perspective does not distinguish the *target* class into individual targets. Fortunately, another type of segmentation called instance segmentation is able to further segment multiple targets in the *target* class into instances, as the name implies, by learning another label called instance ID in addition to the class label. Recent developments (2017-2020) include instance segmentation frameworks for 3D point cloud data [28]. This type of segmentation enables a point cloud to further segment multiple objects from one class into individual instances, so to identify more than one target, instance segmentation is needed as opposed to semantic segmentation. There are several existing implementations

of point cloud instance segmentation [18, 35, 45, 46, 76, 82]. Many of these early point cloud instance segmentation frameworks use a feature extractor, a semantic segmentation network, and then a post-processing algorithm, such as clustering, to determine the instances of each class.

5.1.2.1 Labeling point clouds for machine learning.

Supervised learning typically needs many diverse training observations, i.e. labeled point clouds, to properly train and generalize a machine learning model to be able to infer unseen input. Therefore, many point clouds should be collected and labeled that have rich and diverse 3D geometric features to produce a labeled point cloud dataset. Unsupervised learning methods for 3D data are either immature or nonexistent because it is difficult to cluster point information; this is an area that could also use some exploration. The points would be labeled into one of two classes, *target* or *non-target*, and then a machine learning framework could learn to identify the target from the rest of the scene. Using semantic segmentation, this would segment the scene into only two classes. This is meant to simplify the problem, where the points representing human bodies would segment into the *target* class and everything else in the scene (buildings, walls, trees, ground, etc) will segment into the *non-target* class. However, for the case with multiple targets, to further segment multiple targets in the *target* class into individual instances, then the points also need to be labeled with an instance ID in addition to the class label. Each targets' point set need to be identifiable as a specific individual instance to train an point cloud instance segmentation network.

Because manually labeling point clouds is such a tedious task even with cuboid point selection tools available, there are two other ways to address this issue either by spreading the workload to a group of point cloud labelers or by automating the production of labeled point cloud data. Currently, dedicated groups of researchers are manually labeling point clouds that produced many of the currently available datasets. This method

is slow, tedious, but certain to produce a labeled point cloud dataset. In order to produce the large and diverse amount of training data needed, one way is to crowdsource the workload of labeling point clouds. For example, a group of computer scientists at Carnegie Mellon University developed the well-known labeling application called the **C**ompletely **A**utomatic **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (commonly known as CAPTCHA) that was originally created to produce a dataset of character and word recognition by crowdsourcing the labeling task to internet users as a security test against bots [74]. The group was later acquired by a large technology company and helped develop a crowdsourced labeling system to produce an image recognition dataset for autonomous driving applications. In theory, a similar crowdsourcing system could be developed for labeling point clouds. However, compared to labeling words or images, manually labeling point clouds as a CAPTCHA task using a cuboid selector tool would be less intuitive and more frustrating due to the 3D nature of point selection on 2D computer displays. To simplify the task for an average internet user, it should also be multiple choice similar to image recognition tasks. For example, a CAPTCHA task could provide an image of an object and then ask to select the set of points from a finite number of multiple choices that best matches the object in the image. Or it could provide a simplified 2D projected view of the 3D point data and then ask to select the points that best matches the image. The answers to such CAPTCHA tasks would then be aggregated into a dataset of labeled point clouds, which then could be used for training a point cloud machine learning framework.

The other method of producing labeled point clouds is to procedurally generate them in a virtual environment using simulated lidar. For example, the group of researchers who developed SqueezeSeg demonstrated this approach by using Grand Theft Auto V, a video game that can simulate driving in an urban environment [80]. Using a software plugin called Script Hook V, a simulated lidar was installed on top of an in-game autonomous car making it able to collect point clouds while driving [8, 34]. The lidar's simulated rays

collected four main pieces of information on every generated point: the point's coordinates, the object class, the instance ID, and the bounding box of the object. While the point clouds are being generated, it is simultaneously collecting the labels that are programmed into the metadata of in-game objects, thus creating a labeled point cloud dataset with automated ease. A future work project would be to install the simulated lidar sensor on an in-game aircraft and scan the in-game ground environment to produce scene point clouds. This would produce training data for machine learning in similar scenario cases from this research. One research question would be to ask how well a machine learning model works when trained on virtual data and then applied to point clouds from the real-world. An AFIT researcher Dr. Scott Nykl developed a computer simulation engine called AftrBurnr that has a simulated flash lidar module, which could also help in developing an automated point cloud labeler [55]. Some of Dr. Nykl's students were able to use AftrBurnr to produce virtual training data [43, 58].

5.1.3 Considerations on the ground target to improve detailed occlusion.

If it was important to find detailed occlusions, such as occlusions caused by specific parts of the body of the target, then one area of potential future work would be to utilize the entire set of points of the target, instead of being represented as a single point, because its geometric features are needed to be represented to determine the occlusions for specific body parts. Another option would be to use machine learning to segment the target into its major body parts–such as head, torso, arms, and legs–to identify by those specific body parts by object class and provided that a dataset exists to train a machine learning model to that level of granularity. This would increase the granularity of the visibility when detecting the target and therefore provide greater detailed visibility to the human operators. Another option would be to decide on choosing the head of the body to represent the target, as opposed to the centroid. There are many cases where the centroid of the target is occluded, such as an RPA viewing from even slightly horizontal non-nadir positions. The head is

usually the most visible part of the body when viewing from the sky because typically it is the first body part to become visible after being vertically eclipsed behind a building or wall. The head also includes the facial features, so it is reasonable to select the head as the focus for detection and identification. Note that the current sensor technology at the standard operating cruising altitude captures too low resolution to effectively distinguish facial features for the purposes of facial recognition.

5.1.4 Obtaining the truth point cloud.

The ideal occlusion point cloud can be created by first capturing a detailed point cloud representation of the buildings in the ground scene by using a lidar scanner on the ground to capture the fine details to include the true surface normals. This ground scene point cloud with fine details would be considered ideal, or near-ideal, in comparison to the SfM point cloud that represents only a general shape of the buildings. Using this highly accuracy scene point cloud, the ideal occlusion point cloud can then be determined. The scene point cloud is surface reconstructed using the true normals, instead of estimated normals using nearest neighbors, to create a set of 2D surfaces. Then, instead of using cones to model the line of sight, rays of light are used. The target point acts as a source of light that emit uniform rays that are occluded by the surfaces, thus casting a shadow on a surface. The surface is the flight path plane in the sky, so the shadow occludes an area on that plane, which then can be used to select the points that are within the shadow area, creating the ideal occlusion point cloud.

5.1.5 Data that includes human actors for targets.

Note that in the six scenes used to test conic ray tracing, there was no target in the initial raw videos collected, meaning there were no human actors nor any mannequins representing the target. However, Capt Olson had later collected videos of scenes that included human actors performing simple tasks such as standing and walking to represent the ground target [56]. These could be used in future work in conjunction with machine

learning to perform point cloud segmentation to identify the ground target from the rest of the point cloud.

Appendix A: Research Equipment and Instruments

The equipment used for this experiment is primarily a laptop computer running Microsoft Windows 10 with a dedicated NVIDIA GPU. The neural network will be designed mainly using Python, TensorFlow, and Keras. The hardware specifications and software version numbers are listed in Table 5.1.

Specifications:	
Computer	Lenovo ThinkPad P51
RAM	16 GB dual-channel (15.6 GB usable)
CPU	Intel Xeon E3-1535M v6 @ 3.10GHz "Kaby Lake" 14nm
GPU	NVIDIA Quadro M2200, driver version 456.38
OS	Microsoft Windows 10 Edu 64-bit, version 2004, build 19041.388
Hard drive	476 GB Samsung SSD
Python	3.7.4
TensorFlow	2.1.0
Keras	2.3.1
MATLAB	R2020b
Regard 3D	1.0.0
MeshLab	v2020.12

Table 5.1: Hardware specifications and software versions of the computer used in this thesis

Appendix B: Surface Reconstruction Results

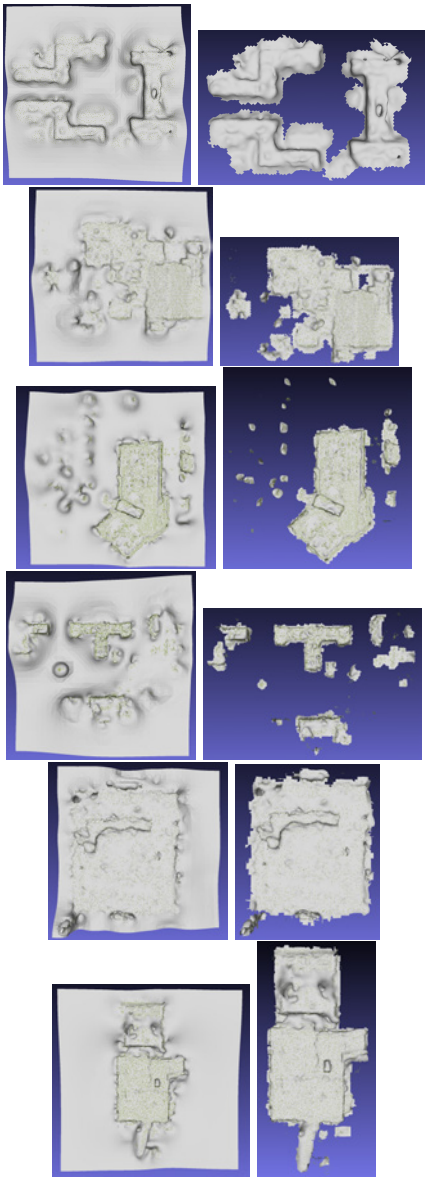

Figure 5.1: Screened Poisson surface reconstruction and the resulting surface deletion.

Appendix C: Ethics of Autonomous Warfare

One of the final topics to discuss is the ethics of autonomous warfare due to the increasing development of automation in weapons technology. {taken from chapter 1, introduces human-machine relationships} Currently, researchers have been developing autonomous systems that interact with the pilot in a human-machine collaborative relationship in which the machine component can be considered an augmentation of the human pilot. For example, human pilots currently provide an immediate situational awareness and tactical response on a level that RPAs cannot currently provide. The human-machine relationship enables the advantages of both the human and the machine by combining the experience and judgment that the human provides with the speed and precision that the machine provides in order to jointly perform more sophisticated tasks. In the long-term, it is possible that the human pilot becomes the weakest link, the most limiting component, that the human pilot would then be designed out of the weapon system because of the advances in adversarial autonomous weapon systems that would produce threats that can attack too quickly, too great in numbers, and thus overwhelming for a human operator to manually respond. It is already anticipated that the performance of the automation in RPA will one day greatly exceed human pilots in manned aircraft because of the physiological limitations of the human body. In a threat scenario such as a swarm of autonomous drones, the countermeasure must also be an autonomous system that can outperform the threat. This scenario has led into a new arms race of AI and autonomous weapons that outperform and overwhelm weapon systems that rely on humans for decision making and taking action because autonomous systems can operate quicker, reach farther, have greater accuracy and precision, and do not have the human limitations of stress and fatigue.

An autonomous weapon system would need to have the authority to decide and act based on its programming, and this has led into a policy debate on how much authority should it be given and what it is allowed to target. In 2015, an AI researcher Toby Walsh signed and released an open letter discussing ethical responsibility by calling for the ban of autonomous weapons that target humans [75]. Walsh fears that autonomous weapons in the wrong hands will certainly be used against civilians because war is asymmetric by its nature, so he believes the United Nations (UN) should enact policy to limit its production and sales by agreeing to not produce fully autonomous weapons. The term "fully autonomous" means to be given the authorization to target humans by its own internal programming directive. The UN has been holding annual conferences called the Convention on Certain Conventional Weapons (CCW) and part of its agenda is to discuss lethal autonomous weapons. As many as 97 countries have endorsed the ban of fully autonomous weapons. However, no international treaty has been met with full consensus. Currently, the US and Russia firmly reject the proposed ban, and China proposed to allow development and production of fully autonomous weapons but not the use or fielding of them [77]. Certainly, there are strong interests in the research and development of autonomous weapons, and it is likely that the technology will mature first before any heavy regulations may be agreed upon. The AI arms race is in profess.

Bibliography

[1] Alcantarilla, Pablo F, Jesús Nuevo, and Adrien Bartoli. "Fast explicit diffusion for accelerated features in nonlinear scale spaces". *BMVC 2013 - Electronic Proceedings of the British Machine Vision Conference 2013*. 2013. URL www.robesafe.

[2] Alcantarilla, Pablo Fernández, Adrien Bartoli, and Andrew J. Davison. "KAZE features". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7577 LNCS, 214–227. 2012. ISBN 9783642337826. ISSN 03029743.

[3] Appel, Arthur. "Some techniques for shading machine renderings of solids". 37. 1968.

[4] Baird, Chirstopher S. "Why are red, yellow, and blue the primary colors in painting but computer screens use red, green, and blue?", 2015.

[5] Behley, Jens, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. "SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences". *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 9296–9306. 2019. ISBN 9781728148038. ISSN 15505499.

[6] Berger, Matthew, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. "A Survey of Surface Reconstruction from Point Clouds". *Computer Graphics Forum*, 36(1):301–329, 2017. ISSN 14678659. URL https://hal.inria.fr/hal-01348404v2.

[7] Bittner, JiÅí and Peter Wonka. "Visibility in computer graphics". *Environment and Planning B: Planning and Design*, 30(5):729–755, 2003. ISSN 02658135.

[8] Blade, Alexander. "Script Hook V", 2017. URL https://www.dev-c.com/gtav/scripthookv/http://dev-c.com/gtav/scripthookv/.

[9] Caulfield, Brian. "What's the Difference Between Ray Tracing, Rasterization?", 2018. URL https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/.

[10] Chang, Angel X., Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, Fisher Yu, Yi Li, and Fisher Yu. "ShapeNet: An Information-Rich 3D Model Repository". dec 2015. URL http://www.shapenet.orghttp://arxiv.org/abs/1512.03012.

[11] Cignoni, P., M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. "MeshLab: An open-source mesh processing tool". *6th Eurographics Italian Chapter Conference 2008 - Proceedings*, 129–136. 2008. ISBN 9783905673685.

[12] Cohen-Or, Daniel, Yiorgos L. Chrysanthou, Cláudio T. Silva, and Frédo Durand. "A survey of visibility for walkthrough applications". *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003. ISSN 10772626.

[13] Corsini, Massimiliano, Paolo Cignoni, and Roberto Scopigno. "Efficient and flexible sampling with blue noise properties of triangular meshes". *IEEE Transactions on Visualization and Computer Graphics*, 18(6):914–924, 2012. ISSN 10772626. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6143943{&}casa{_}token=FpNY5tRHvnsAAAAA:vluWAnBazmV0bpk8FSewqDxbx{_}nsqp3CgfV592sZtJG38uEpIop4ynXUutyUCkUIP1ly1fo{&}tag=1.

[14] Dado, Bas, Timothy R. Kol, Pablo Bauszat, Jean Marc Thiery, and Elmar Eisemann. “Geometry and attribute compression for voxel scenes”. *Computer Graphics Forum*, volume 35, 397–407. 2016. ISSN 14678659.

[15] DanPMK. “RGB Color Wheel”, 2008. URL https://commons.wikimedia.org/wiki/File:RBG{_}color{_}wheel.svg.

[16] Dilmen, Nevit. “RGB image split into its three RGB channels”, 2012.

[17] Eisemann, Elmar and Xavier Décoret. *Fast scene voxelization and applications*. Technical report, 2006.

[18] Engelmann, Francis, Martin Bokeloh, Alireza Fathi, Bastian Leibe, and Matthias NieBner. “3D-MPA: Multi-Proposal Aggregation for 3D Semantic Instance Segmentation”. 9028–9037, mar 2020. URL http://arxiv.org/abs/2003.13867.

[19] Fan, Haoqiang, Hao Su, and Leonidas Guibas. “A point set generation network for 3D object reconstruction from a single image”. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, 2463–2471. Institute of Electrical and Electronics Engineers Inc., nov 2017. ISBN 9781538604571.

[20] Fowler, James E. and Roni Yagel. “Lossless compression of volume data”. *Proceedings of the 1994 Symposium on Volume Visualization, VVS 1994*, 43–50. 1994. ISBN 0897917413.

[21] Fryar, Cheryl D, Deanna Kruszon-Moran, Qiuping Gu, and Cynthia L Ogden. *Mean Body Weight, Height, Waist Circumference, and Body Mass Index Among Adults: United States, 1999â2000 Through 2015â2016*. Technical report, 2018.

[22] Funkhouser, Thomas A., Carlo H. Séquin, and Seth J. Teller. "Management of large amounts of data in interactive building walkthroughs". *Proceedings of the Symposium on Interactive 3D Graphics*, volume Part F1296, 11–20. 1992. ISBN 0897914678.

[23] Furukawa, Yasutaka, Brian Curless, Steven M. Seitz, and Richard Szeliski. "Towards internet-scale multi-view stereo". *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1434–1441. 2010. ISBN 9781424469840. ISSN 10636919.

[24] Furukawa, Yasutaka and Jean Ponce. "Accurate, dense, and robust multiview stereopsis". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010. ISSN 01628828.

[25] Gambino, Stephanie, Michael Mirochnik, and Scott Schechter. "Center of Mass of a Human", 2005. URL https://hypertextbook.com/facts/2006/centerofmass.shtml.

[26] Greene, Ned, Michael Kassy, and Gavin Millery. "Hierarchical Z-buffer visibility". *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1993*, 231–237. 1993. ISBN 0897916018.

[27] Guennebaud, Gaël, Loïc Barthe, and Mathias Paulin. "Deferred splatting". *Computer Graphics Forum*, 23(3 SPEC. ISS.):653–660, sep 2004. ISSN 01677055. URL http://doi.wiley.com/10.1111/j.1467-8659.2004.00797.x.

[28] Guo, Yulan, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. "Deep learning for 3D point clouds: A survey", 2019. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9127813{&}casa{_}token=nsbOZ7o5fVEAAAAA:xcZaTZy0OITA9CgDLNOrXqwovyvm{_}M8yePCGJupVlXxUdrYJ2cQfi2V8sk6CknppSV19i7F76g{&}tag=1.

[29] Häming, Klaus and Gabriele Peters. "The structure-from-motion reconstruction pipeline - A survey with focus on short image sequences". *Kybernetika*, 46(5):926–937, 2010. ISSN 00235954.

[30] Hardison, Chaitra, Eyal Aharoni, Christopher Larson, Steven Trochlil, and Alexander Hou. *Stress and Dissatisfaction in the Air Force's Remotely Piloted Aircraft Community: Focus Group Findings*. RAND Corporation, may 2017.

[31] Hiestand, Roman. "Regard3D", 2019. URL https://www.regard3d.org/.

[32] Hiestand, Roman. "rhiestan/Regard3D: A open source structure-from-motion program based on OpenMVG.", 2019. URL https://github.com/rhiestan/Regard3D.

[33] Hirsch, Robert. *Exploring colour photography: A complete guide*. 2005. ISBN 1856694208. URL http://books.google.com/books?id=4Gx2WItWGYoC{&}pg=PA28{&}dq=maxwell+additive+color+photograph+register{&}lr={&}as{_}brr=0{&}ei=-K6BR-TrBYGmswP6jqHDCw{&}sig=KzP8phk345XPVijOyYR{_}KlffeXc{#}PPA28,M1.

[34] Hurl, Braden, Krzysztof Czarnecki, and Steven Waslander. *Precise synthetic image and LiDAR (PreSIL) dataset for autonomous vehicle perception*. Technical report, 2019. URL http://www.dev-c.com/gtav/scripthookv/.

[35] Jiang, Li, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. "PointGroup: Dual-Set Point Grouping for 3D Instance Segmentation". 4866–4875, apr 2020. URL http://arxiv.org/abs/2004.01658.

[36] Johnson, Stephen. *Stephen Johnson on Digital Photography*. 2006. ISBN 059652370X. URL http://books.google.com/books?id=0UVRXzF91gcC{&}pgis=1.

[37] Katz, Sagi, Ayellet Tal, and Ronen Basri. "Direct visibility of point sets". *ACM Transactions on Graphics*, 26(3), 2007. ISSN 07300301.

[38] Kazhdan, Michael and Hugues Hoppe. "Screened poisson surface reconstruction". *ACM Transactions on Graphics*, 32(3), 2013. ISSN 07300301. URL http://dx.doi.org/10.1145/2487228.2487237.

[39] Koch, Sebastian, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. *ABC: A big cad model dataset for geometric deep learning*. Technical report, 2019. URL https://deep-geometry.github.io/abc-dataset.

[40] Kwamikagami. "RYB Color Wheel", 2015. URL https://commons.wikimedia.org/wiki/File:Color{_}star-en{_}(tertiary{_}names).svg.

[41] Lai, Kevin, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. "A large-scale hierarchical multi-view RGB-D object dataset". *Proceedings - IEEE International Conference on Robotics and Automation*, 1817–1824. 2011. ISBN 9781612843865. ISSN 10504729.

[42] Laine, Samuli and Tero Karras. "Efficient sparse voxel octrees". *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1048–1059, 2011. ISSN 10772626.

[43] Lee, Andrew. "Object Detection with Deep Learning to Accelerate Pose Estimation for Automated Aerial Refueling". *Theses and Dissertations*, mar 2020. URL https://scholar.afit.edu/etd/3163.

[44] Leyvand, Tommer, Olga Sorkine, and Daniel Cohen-Or. "Ray space factorization for from-region visibility". *ACM Transactions on Graphics*, volume 22, 595–604. 2003. ISSN 07300301.

[45] Liu, Chen and Yasutaka Furukawa. "MASC: Multi-scale Affinity with Sparse Convolution for 3D Instance Segmentation". feb 2019. URL http://arxiv.org/abs/1902.04478.

[46] Liu, Shih-Hung, Shang-Yi Yu, Shao-Chi Wu, Hwann-Tzong Chen, and Tyng-Luh Liu. "Learning Gaussian Instance Segmentation in Point Clouds". jul 2020. URL http://arxiv.org/abs/2007.09860.

[47] Losey, Stephen. "Here's where the Air Force's pilot shortfall is the worst", 2020. URL https://www.airforcetimes.com/news/your-air-force/2020/07/07/heres-where-the-air-forces-pilot-shortfall-is-the-worst/.

[48] Maiman, T. H. "Stimulated optical radiation in Ruby". *Nature*, 187(4736):493–494, 1960. ISSN 00280836.

[49] Mongenet, Marc. "All 16,777,216 colors of 24-bit RGB palette (truecolor)", 2005. URL https://commons.wikimedia.org/wiki/File:16777216colors.png.

[50] Morton, J.L. "Basic Color Theory". *Color Matters*, 2014. URL http://www.colormatters.com/color-and-design/basic-color-theory.

[51] Morvan, Yannick. "Projective Geometry", 2018. URL http://epixea.com/research/multi-view-coding-thesisch2.html{#}x13-320002.2.1.

[52] Moulon, Pierre, Pascal Monasse, Romuald Perrot, and Renaud Marlet. *OpenMVG: Open multiple view geometry*. Technical report, 2017. URL https://pix4d.com/.

[53] Muja, Marius and David Lowe. "FLANN - Fast Library for Approximate Nearest Neighbors User Manual". *Visapp*, 331–340, 2011.

[54] National Oceanic and Atmospheric Administration. "What is LIDAR?", 2020. URL https://oceanservice.noaa.gov/facts/lidar.html.

[55] Nykl, Scott, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu. "An overview of the STEAMiE educational game engine". *2008 38th Annual Frontiers in Education Conference*, F3B–21. 2008. URL https://git.nykl.net.

[56] Olson, Aubrey. "AFFTRAC Unclassified Flight Test Data", 2020.

[57] Osoba, Osonde and William Welser. *The Risks of Artificial Intelligence to Security and the Future of Work*. Technical report, 2017.

[58] Parsons, Christopher, Zachary Paulson, Scott Nykl, William Dallman, Brian G. Woolley, and John Pecarina. "Analysis of simulated imagery for real-time vision-based automated aerial refueling". *Journal of Aerospace Information Systems*, 16(3):77–93, mar 2019. ISSN 23273097. URL https://arc.aiaa.org/doi/10.2514/1.I010658.

[59] Pratt, Leslie. "MQ-1 Predator", 2008. URL https://upload.wikimedia.org/wikipedia/commons/c/c7/MQ-1{_}Predator{%}2C{_}armed{_}with{_}AGM-114{_}Hellfire{_}missiles.jpg.

[60] Pratt, Leslie. "MQ-9 Reaper", 2008. URL https://commons.wikimedia.org/wiki/File:MQ-9{_}Reaper{_}UAV{_}(cropped).jpg.

[61] Qi, Charles R., Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep learning on point sets for 3D classification and segmentation". *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, 77–85. 2017. ISBN 9781538604571.

[62] RCraig09. "Lidar time of flight", 2020.

[63] Ring, James. "The Laser in Astronomy". *New Scientist*, 344:672–673, 1963. URL https://books.google.com/books?id=0hWpWSF7e7YC{&}pgis=1.

[64] Rusinkiewicz, S. and M. Levoy. *QSplat: A multiresolution point rendering system for large meshes*. Technical report, 2000.

[65] Saxena, Ashutosh, Sung H. Chung, and Andrew Y. Ng. "3-D depth reconstruction from a single still image". *International Journal of Computer Vision*, 76(1):53–69, 2008. ISSN 09205691.

[66] Schaufler, Gernot and Henrik Wann Jensen. "Ray Tracing Point Sampled Geometry". 319–328. 2000.

[67] Schonberger, Johannes L. and Jan Michael Frahm. "Structure-from-Motion Revisited". *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, 4104–4113. 2016. ISBN 9781467388504. ISSN 10636919.

[68] Sharma, Bhupendra. "What is LiDAR technology and how does it work?", 2020.

[69] Simek, Kyle. "Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix", 2013. URL https://ksimek.github.io/2013/08/13/intrinsic/.

[70] Stich, Martin. "Introduction to NVIDIA RTX and DirectX Ray Tracing", 2018. URL https://developer.nvidia.com/blog/introduction-nvidia-rtx-directx-ray-tracing/.

[71] Sutherland, Ivan E., Robert F. Sproull, and Robert A. Schumacker. "A Characterization of Ten Hidden-Surface Algorithms". *ACM Computing Surveys (CSUR)*, 6(1):1–55, 1974. ISSN 15577341.

[72] Switzer, Tobias. "The Air Force Pilot Retention Crisis Is Not Over", 2020. URL https://warontherocks.com/2020/10/the-air-force-pilot-retention-crisis-is-not-over/.

[73] Ullman, S. "The Interpretation of Structure from Motion". *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 203(1153):405–426, 1979. ISSN 00804649. URL https://dspace.mit.edu/bitstream/handle/1721.1/6298/AIM-476.pdf;jsessionid=380F3929BD3E6B8ED8FB93856FD9AA09?sequence=2.

[74] Von Ahn, Luis, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. "reCAPTCHA: Human-based character recognition via web security measures". *Science*, 321(5895):1465–1468, 2008. ISSN 00368075. URL http://science.sciencemag.org/.

[75] Walsh, Toby. "Open Letter on Autonomous Weapons - Future of Life Institute", 2015. URL https://futureoflife.org/open-letter-autonomous-weapons/?cn-reloaded=1.

[76] Wang, Weiyue, Ronald Yu, Qiangui Huang, and Ulrich Neumann. "SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation". *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2569–2578. 2018. ISBN 9781538664209. ISSN 10636919.

[77] Wareham, Mary. "Stopping Killer Robots– Country Positions on Banning Fully Autonomous Weapons and Retaining Human Control", 2020. URL https://www.hrw.org/report/2020/08/10/stopping-killer-robots/country-positions-banning-fully-autonomous-weapons-and{#}{_}ftn5.

[78] Westoby, M. J., J. Brasington, N. F. Glasser, M. J. Hambrey, and J. M. Reynolds. "'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications". *Geomorphology*, 179:300–314, 2012. ISSN 0169555X.

[79] Whitted, J. Turner. "Ray-Tracing Pioneer Explains How He Stumbled into Global Illumination", 2018. URL https://blogs.nvidia.com/blog/2018/08/01/ray-tracing-global-illumination-turner-whitted/.

[80] Wu, Bichen, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. "SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud". *Proceedings - IEEE International Conference on Robotics and Automation*, 1887–1893. 2018. ISBN 9781538630815. ISSN 10504729.

[81] Wu, Jianhua and Leif Kobbelt. "Optimized sub-sampling of point sets for surface splatting". *Computer Graphics Forum*, volume 23, 643–652. Blackwell Publishing Ltd, sep 2004. ISSN 01677055. URL http://doi.wiley.com/10.1111/j.1467-8659.2004.00796.x.

[82] Yang, Bo, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. "Learning Object Bounding Boxes for 3D Instance Segmentation on Point Clouds". jun 2019. URL http://arxiv.org/abs/1906.01140.

[83] Yodayoda. "From depth map to point cloud: How to convert a RGBD image to points", 2020. URL https://medium.com/yodayoda/from-depth-map-to-point-cloud-7473721d3f.

[84] Zwicker, Matthias, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. "Surface splatting". *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001*, 371–378, 2001. URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:4266873.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Oct 2019–Mar 2021

4. TITLE AND SUBTITLE

Comparison of Conic Ray Tracing for Occlusion Determination on 3D Point Cloud Data

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cho, Henry, Capt, USSF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-021

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Test Pilot School
220 Wolfe Ave.
Edwards AFB CA 93523
DSN 277-3000, COMM 661-277-3000

10. SPONSOR/MONITOR'S ACRONYM(S)

USAF TPS

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The US Air Force has been increasing the use of automation in its weapon systems to include the remotely piloted aircraft (RPA) platforms. The RPA career field has had issues with poor pilot retention due to job stressors. For example, RPA operators spend a lot of time and attention surveilling a suspect on the ground for many hours, so adding automation to this activity could help improve pilot retention. The research problem in this thesis attempted to automate the process of observing a ground target. This thesis presents a method termed conic ray tracing for determining visibility and occlusion of a ground target from locations in the airspace represented by 3D point cloud data. This conic ray tracing method uses 3D points representing a scene to trace rays and then using a matrix formulation of the dot product to compute the angles between every ray to the points representing airspace and every ray to the points representing the ground scene. Whether the angle is inside or outside a fixed-angle cone determines occlusion or visibility respectively. The method was tested on 3D point clouds generated from Structure from Motion using real-world imagery data collected from an MQ-9 Reaper flight test. Because the truth data was not available, the results from conic ray tracing were compared to a reference created by using true graphical ray tracing on a surface reconstruction of the original point cloud scenes. When compared to the reference, the conic ray tracing method averaged 0.81 accuracy over the eight cases studied, and the computational runtime was on average 19x faster than the algorithm for computing the reference results from true ray tracing. Limitations and future work were discussed.

15. SUBJECT TERMS

point cloud, visibility, occlusion, ray tracing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			(ENG)
U	U	U	UU	99	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 ext. 7469