ForgeScan:

A Framework for Iterative Voxel Reconstructions and Next-Best View Selection

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Andrew Allen Schellenberg

Graduate Program in Mechanical Engineering

The Ohio State University

2023

Thesis Committee

Dr. Michael Groeber, Advisor

Dr. Ayonga Herried

Dr. Haijun Su

Dr. Andrew Gillman

Abstract

This thesis investigates methods for autonomously reconstructing digital models from measurements of a physical environment. For embodied autonomous systems, understanding the current state of the surrounding space is critical for many higher-level decisions. This work is motivated by manufacturing systems performing iterative deformation processes on workpieces. However, perception-informed decisions occur in all kinds of systems: an aerial drone mapping a warehouse, a delivery robot driving around an obstacle, or a bin-picking robot selecting the correct object.

Despite active research concerning methods to best integrate measurements into voxelized reconstructions during mapping or scanning tasks, there is no set of tools to develop and compare approaches. In response to this need, this work introduces ForgeScan, an open-source library that unifies voxel grid representations, their update methods, and next-best view selection algorithms with simulated or real depth sensors.

Rather than reworking the same development path, ForgeScan is designed to be minimal and adaptable. New voxel update rules are easily implemented as a subclass of an abstract voxel grid base class and the use of C++17 variants provides datatype flexibility.

More than a versatile voxel grid data structure, ForgeScan provides an abstract policy class to perform view selection algorithms and a lightweight depth camera simulator to generate synthetic data. User-defined policies can suggest camera poses, generate depth images of a mesh, and then add these measurements to one or more voxel grid implementations.

ForgeScan is designed to be flexible at runtime. While collecting data, users may interactively add new voxel grids or change what policy is running. At any time, the state of each grid's reconstruction may be saved and inspected with VTK.

Common voxel methodologies – space carving, truncated signed distance fields, and occupation probability – are implemented to demonstrate the library and explore the fundamental limitations of voxels. The precision and miss-rate of these methods are compared for increasing sensor uncertainty for a variety of view selection algorithms.

Additionally, simple, open-loop view selection policies are compared with novel algorithms. One of these new policies searches for voxels on the boundary of seen and unknown space and seeks to view as many of these voxels as possible. Another takes an assumed model of the scene and selects a set of views where the camera's optical axis is perpendicular to as many faces as possible while being distant from other views in the set.

Dedicated to:

Elena Akers, for her infinite support, no matter the number time of zones or distance between us;

My parents and brother, for their unrelenting encouragement courses and research monopolized my time and energy for last two years;

My colleagues and classmates, for their own stelar achievements that inspire me expand my knowledge;

My friends, for their insistence that I get up from my computer ever so often and appreciate all aspects of life.

Acknowledgments

Knowledge is not gained in a vacuum. It is not the result of a single individual working on an isolated project. And, despite how a thesis presents its contents, it is not a clear, linear progression. Research is a collaborative effort. It is a messy process where one encounters unexpected turns and dead-ends regardless of how well prepared.

Sheer knowledge is not the measure of a true researcher. Instead, it is the confidence and capacity to navigate these uncertain paths. To find the inspiration of what is possible in the confusion of what could be.

I have had the great fortune to work alongside such individuals throughout my academic experience. In this specific endeavor, there are innumerable individuals to whom I owe gratitude.

My advisor, Dr. Mike Groeber, provided support throughout the entire course of this project. His ambitious vision inspired me to face the unknown, searching for opportunities rather than seeing problems. I am grateful for his patience and confidence while I navigated between ideas and developed my approaches.

As members of my defense committee, Dr. Haijun Su and Dr. Ayonga Hereid provided valuable feedback on this project. When I was an undergraduate student, unsure of what opportunities to explore, their work in robotics influenced my nascent interest in this area.

I am grateful for the mentorship of Dr. Andrew Gillman, also a committee member, and Dr. Mathew Cherry. Their perspectives helped guide my research interests. They taught me to see how research propagates and evolves into meaningful, impactful results.

To my lab mates, thank you for helping me maintain my sanity through the arduous parts of the research process. I hazard to say no other lab has collected such an intelligent, capable, and supportive group. I am proud and humbled to have worked with all of you - graduate and undergraduate students alike.

Finally, no project would be complete without funding. This work was supported by the Air Force Research Lab's Systems Technology Office through grant # FA8650-19-1-5227.

Vita

2021 …….. B.S. Mechanical Engineering, The Ohio State University

2021 to Present ……..…….. Graduate Research Associate, Artificially Intelligent Manufacturing Systems Lab, The Ohio State University

Fields of Study

Major Field: Mechanical Engineering

Table of Contents

List of Tables

List of Equations and Algorithms

List of Figures

Chapter 1. Introduction

1.1 Motivation

To decide on meaningful, informed actions requires understanding one's environment. This is as true for humans acting upon the world around them as it is for systems with any degree of autonomy. Perceiving and understanding an environment is increasingly important as decisions become complex and interdependent with the environment's state.

New autonomous systems are developing at an explosive rate. Underscoring this is the increasing availability of diverse robotic platforms. Ariel drones, mobile units, and manipulator arms are available at every price point for commercial and educational uses. Whatever the platform, robotics embody autonomous systems to interact with and navigate the world around them.

All types of sensors experience continual growth in capability and accessibility over time. But in the two decades, depth cameras have experienced unique, exponential increase in sensing power and commercial availability. They are increasingly present in both research and commercial applications. In the last decade, companies like Microsoft and Intel have led the way in low-cost depth cameras with their Kinect and RealSense product lines.

Improvements in embedded computational platforms have also contributed to the development of both robotics and sensors. It is now possible to have the on-board computing power and memory required to run computer vision processes, path-planning algorithms, or machine learning inference in real-time. Increasingly, companies like Nvidia are integrating graphics cards in embedded systems, empowering general-purpose parallel programming.

This triumvirate of robotic platforms, low-cost sensors, and powerful embedded computing is enabling companies, researchers, students, and even hobbyists to redefine the capabilities of autonomous systems.

These systems are designed to operate in an immense range of environments while performing any number of application-specific tasks. Meaningfully generalizing these precise decisions is not feasible. But one can recognize that many decisions rely on an understanding of their surrounding environment before making a particular high-level decision. Building this understanding is a fundamental sub-task for many autonomous systems and is referred to as a reconstruction task.

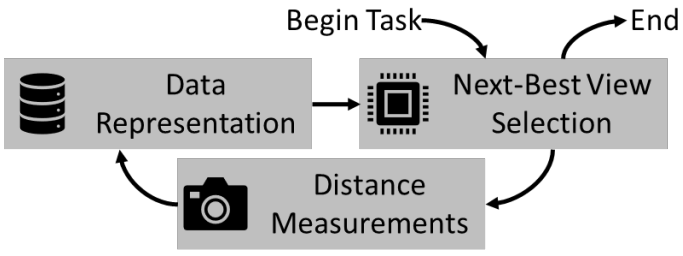


Figure 1 Key parts of reconstruction tasks.

The goal of a reconstruction task, outlined in Figure 1, is to generate a reasonable computational model of an environment, some region of the real world near the system. The task has three core elements: a unified representation of all the data collected, a decision-making algorithm to select the next best sensor location, and a sensor capable of measuring distances. This is an iterative process beginning with the view selection algorithm and ending when it determines no additional views are needed.

Whether a reconstruction is reasonable depends on several factors. One general goal is completeness – did the system explore every unknown area? Another is confidence – is there consensus between new measurements and the model? The accuracy and precision of a reconstruction are also important, though precise values are application-specific, based on the system, task, and sensors.

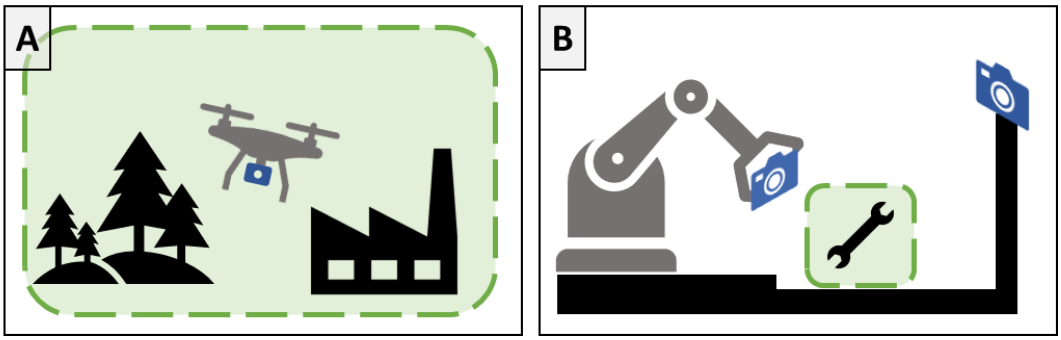


Figure 2 Reconstruction tasks of (A) mapping an environment and (B) scanning a part.

Reconstruction tasks are categorized as either mapping or scanning. Figure 2 depicts a drone mapping a large scene in A and a manipulator scanning a part in B. The relative size and positions of the system and the region of interest tend to differentiate these. Systems navigating within a large environment perform mapping while those observing a smaller

environment from the outside engage in scanning. The distinction is not rigid, but the categories imply assumptions about the specific reconstruction task.

Mapping tasks typically assume the environment is primarily empty space and sparsely populated with obstacles in unknown locations. The goal is simultaneously collecting measurements and navigating. The scene is typically assumed to be completely unknown, though some approaches may define bounds for the scanning region.

Scanning tasks typically assume a single object exists near the center of a bound scanning area. The scanning area is either fixed in space while the camera moves or moving in space while the camera is static. Scanning tasks also assume the scene to be static. Most approaches assume no knowledge beyond the bounded region. However, there are cases where prior knowledge – a CAD model, a prior scan, or an assumed geometry – could assist in the view selection algorithm.

Delivery drones must traverse new and evolving spaces before reaching a destination. Bin-picking robots must identify the objects around them and their orientations before executing a grasp. Manufacturing systems must know the current state of a workpiece and how far it is from the desired shape before deciding the next operation.

Each system has a different goal. Different data representations, view selection algorithms, and distance sensors. But each engages in mapping or scanning to accomplish its goal. Lacking this step or performing it poorly, systems crash, malfunction, or act dangerously.

For humans, this process of view selection and scene representation is intuitive: evolution and experience have honed our proprioceptive skills. But translating this understanding to autonomous systems is still an active area of research. The problems of view selection and data representation are complementary and there is neither a general solution nor widely used implementation for either.

There likely is not a one-size-fits all algorithm for view selection or data representation. Evaluating existing approaches or developing new ones for a specific task is an onerous process. Relevant algorithms and representations from literature often must be reimplemented and lack clear metrics for comparing performance.

This work presents ForgeScan as an approach for developing and evaluating solutions for reconstruction tasks in a unified manner. It unifies the three steps of a reconstruction task and specifically focuses on the modular design of next-best view algorithms and data representations.

For data representation, this work focuses on voxel grids. Voxels are discretized regions of space organized into a regular grid. While some approaches rely on point clouds [1]–[3], meshes [4], or unique combinations of data structures [5]–[8], voxel approaches remain one of the most common representations used in literature.

This work provides a means to implement view selection policies and quantify their performance on a variety of environments. This begins to answer fundamental questions about what situations various policies are most appropriate for.

1.2 Contributions

Performing reconstruction tasks is just one subset of research in autonomous systems. But it is a foundational problem which produces information required for a broad spectrum of tasks like navigation, manipulation, or inspection. Furthering research in autonomous reconstruction, this work contributes the following:

- ForgeScan, an open-source library for developing and evaluating view selection policies and data representations for reconstruction tasks.
- A comparison of space carving, truncated signed distance field (TSDF), and occupancy probability voxel representations for a variety of geometries and view selection algorithms across various levels of sensor noise.
- Two novel scanning view selection policies, one with a frontier-based approach and another using an assumed object geometry to precompute initial views.

1.3 Outline

- **Chapter 2. Background** introduces concepts fundamental to this work. It provides a brief overview of depth camera operation and a comparison of 3D data structures. It also discusses the process of tracing a ray through a regular grid.
- **Chapter 3. Related Work** shares relevant work on mapping and scanning problems. It introduces foundational work on view selection algorithms and discusses notable current works. It describes the motivation for and implementation of space carving, truncated signed distance field, and occupancy probability voxel representations.

- **Chapter 4. Methodologies** describes the structure of the ForgeScan library (code available at [9]) and notable features about its implementation. It also describes the two view selection policies introduced in this work.
- **Chapter 5. Results** compares how different voxel reconstructions are performed on various objects with different view selection policies and sensor noise. The performance of this work's novel view selection policies is also shared.
- **Chapter 6. Conclusions and Future Work** expands on the prior chapters to summarize the contributions and provide insight on what scenarios different view selection policies and representations operate best for. It identifies the remaining research gaps and suggests directions for continuing work.

Chapter 2. Background

Before discussing existing literature in this area or exploring the methodologies and results of this work, a primer on important background topics is required. Reconstruction tasks are achieved by uniting concepts from robotics, depth imaging, and computational geometry.

The details of the first field, robotics, tend to be implementation-specific and is not discussed in detail. Mobile platforms performing mapping may focus on computational efficiency and data compressibility to best use their limited hardware in real-time. A manipulator-based system may be concerned with reachability and self-occlusion of the scan object. All systems have some amount of pose uncertainty that a reconstruction must contend with when collecting measurements. Beyond these general statements, concerns can vary widely, so many reconstruction algorithms are platform-agnostic.

Depth imaging and computational geometry are intertwined with reconstruction at a more fundamental level. Details about depth camera operation are important in understanding the nature of measurement errors. The trade-off of different geometric data structures is also relevant. Updating, searching, or ray casting with at least one of these representations is key to all reconstruction algorithms.

2.1 Depth Camera Fundamentals

Depth cameras are an interesting class of sensors that simultaneously collect many distance measurements between the camera and the objects within its view. These sensors collect a great deal of information at one time and store it in an image format.

Some of these cameras include traditional color cameras too. With calibration, the color image and depth measurements can be combined into one. This combination of visual and spatial perception is unique and not replicable with other measurement techniques.

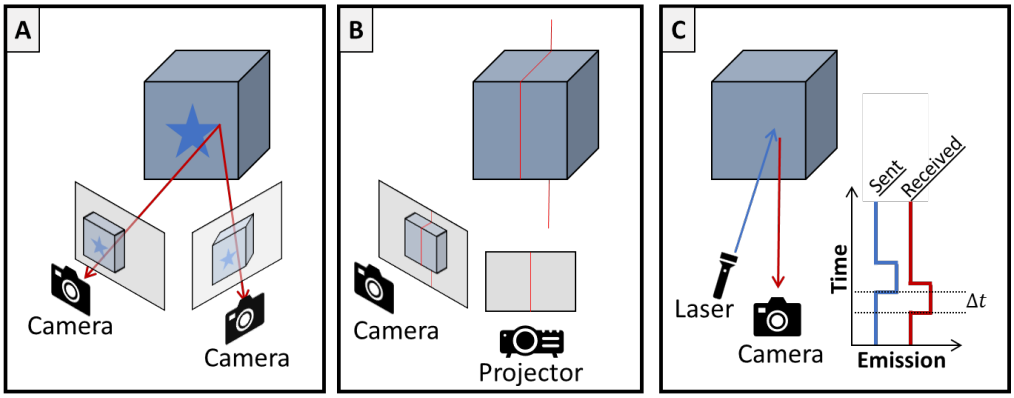


Figure 3 Depth camera methods (A) stereo vision, (B) SL, and (C) ToF.

Most depth cameras utilize one of three operating principles: stereo vision, structured light (SL), and time of flight (ToF). Each has different operating ranges, sources of uncertainty, and cost. Like the view selection policy and data representation, identifying the correct sensor is always an important system design choice for reconstruction tasks. A simplified diagram for each of these methods is shown in Figure 3.

Stereo depth cameras utilize two calibrated cameras at known positions to estimate depth. Feature matching is used to find the same location in both camera's images and triangulation is used to estimate its spatial position.

A common set up has co-planar cameras with a horizontal offset called the baseline distance. This simplifies the pattern matching algorithm and makes the sensors depth range proportional to its baseline distance. An overview of the sensor setup and requisite calculations is provided in [10].

This process can use color images, but regularly repeated surface features – or the lack of any distinct features – can lead to ambiguity in the feature matching step. Some cameras, like those in the Intel RealSense D4XX product line [11], utilize active infrared (IR) stereo. This technique includes an IR projector to add a pseudo-random pattern to the scene while collecting images with IR cameras. The added pattern improves the reliability and speed of the feature matching step [12].

Structured light uses the same coplanar setup as stereo but replaces one camera with a projector. It broadcasts specific, known patterns from the projector and the camera observes how these deform. The patterns add identifiable features to the scene. Knowledge of the original pattern[1] and the relative positions of the camera and projector turns allows depth to be estimated based on triangulation. This method is covered in detail by [13].

[1] While SL and active stereo both project patterns onto a scene precise, prior knowledge of the pattern and its direct use in triangulation is what makes SL distinct from active stereo.

ToF cameras are a style of light detection and ranging (LIDAR) sensors. They emit light from a source then sense how that signal is reflected by the scene in front of the camera. [14] discusses the development history of this style of camera and the approaches to estimate depth by varying how the signal is emitted or measured by the sensor.

Individual sensors have a wide array of operation specifications. Operating ranges can vary from close ranges, like Intel's D405 camera at 0.07 m - 0.5 m [11], to far ones, like Carnegie Robotics' MultiSense S30 at 1.5 m - 30 m [15].

All three operating principles are prone to unique sources of error. However, poor lighting conditions and certain surface finishes impede the operation of all depth cameras methods. The active stereo and SL both project artificial features into the scene to improve the feature matching. However, strong ambient light in the same spectrum the camera uses can disrupt this. Infrared absorption and object transparency are all detrimental to the accuracy of active stereo, SL, and ToF.

All methods tend to measure less accurately and with more uncertainty as surfaces move further from a sensor's minimum-rated distance. This is observed for both triangulation [10], [16], [17] and ToF [14], [18], [19] based cameras.

For triangulation-based cameras, features either converge to similar pixel location in both images or are only visible in one image. In ToF cameras the intensity of the recorded signal decays with distance and the phase delay may lead to ambiguities like depth-folding errors.

Sensor selection is always an important system design decision. For reconstruction tasks especially, the range, accuracy, and type of camera must be selected to match the task and operational environment. This can be a limiting factor in the resolution and uncertainty present in the final reconstruction model.

2.2 Computation Representations for 3D Geometries

Data representation is an important consideration in many problems. Different insights are available depending on how information is structured. Sometimes, the relationships clearly expressed in one method are computationally inefficient or impossible to solve in another.

For spatial data there exists a wide array of representations. Each of these has tradeoffs between information, size, and flexibility. Fortunately, there are many ways to convert between these formats. Information from a sensor or other data source may be transformed into whatever format is relevant for solving a problem.

Describing a volume in space is the typical goal of 3D data. To achieve this, points, lines, surfaces, and volumes may be used to express 0D, 1D, 2D, and 3D information, respectively, inside of a 3D space.

It is important to briefly introduce common data structures before discussing how view selection policies leverage one or more of them to make decisions. A brief overview is provided for each. General details about the structure and what kind of queries or algorithms it is best suited for are shared. The following sections cover depth images, point clouds, polygon meshes, voxels, and octrees.

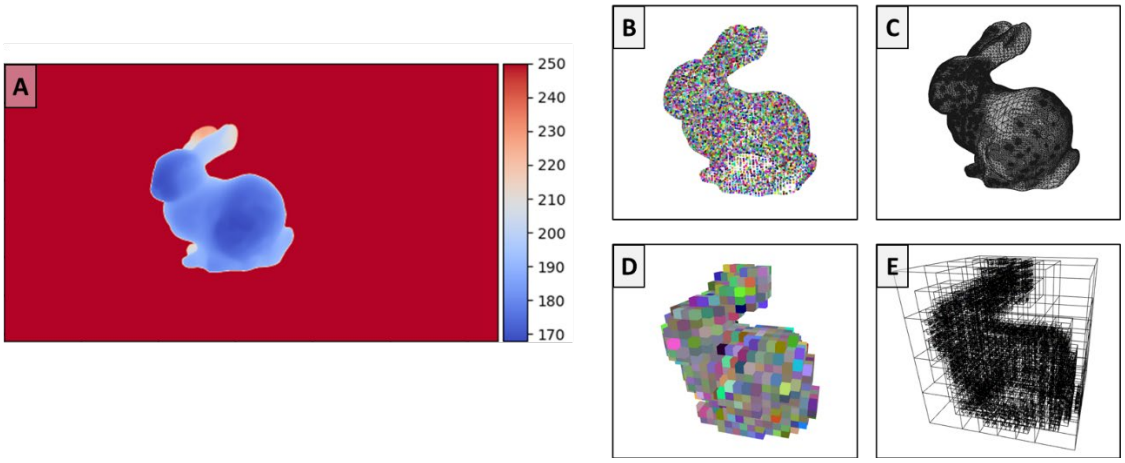


Figure 4 Representation of 3D geometries with (A) depth image, (B) point cloud, (C) triangle mesh, (D) voxels, and (E) octree of the Stanford bunny **[20]**.

2.2.1 Depth Images

Depth images represent a collection of measurements taken be a depth camera or a laser scanner. They resemble color images in structure: a 2D matrix of pixels. Rather than the intensity of light recorded by color images, each pixel corresponds to a distance measurement. An example of a depth image is shown in Figure 4A.

The image structure makes it easy to perform many useful operations on depth images. The organization into pixels places neighboring measurements next to each other and allows computer vision techniques for edge detection, object segmentation, and more to operate directly on depth images. This format may be compressed and transmitted using the same fundamental techniques as color images.

Depth images are a compact representation of point measurements. Only one value, the depth, is stored for each measurement. With knowledge of the sensor's intrinsic parameters, the depth may be projected to recover the full Cartesian coordinates.

For cameras, the depth value represents the distance of a point in the direction of the camera's optical axis. Using the pinhole camera model, a camera's focal length and sensor size are used to find the projection of a pixel in world coordinates. For color images, this projection is under-constrained and only identifies a ray – the path that light must have traveled to reach the sensor [21]. Knowing one Cartesian position, stored in each pixel of a depth image, solves for the precise point on this ray.

For some laser scanners, each measurement is from a ray projected by the sensor in a specific configuration as it sweeps across the vertical and horizontal limits. Recovering the point measurements uses spherical to Cartesian conversion. The depth image stores the radius measurements from each ray while the azimuth and elevation are implicitly defined in the sensors field of view and number of measurements about both directions.

The organization of depth images makes them a convenient initial representation for many applications. They are a common output from cameras and laser scanners. Image processing techniques can quickly process or filter data and additional information like color or segmentation labels may be appended as data channels in the image.

Converting other formats to depth images is generally not possible. But intrinsic knowledge of the sensor that measured a depth image makes it easy to convert to a point cloud after performing any image-based processing.

2.2.2 Point Clouds

In its most basic form, a point cloud is simply a set of points defined by their cartesian distance from a common reference frame. Like depth images, point clouds are a common output from many kinds of measurement devices. Most sensors that produce depth images can optionally provide the measurements directly as point clouds. Figure 4B shows a point cloud of the Stanford bunny with each point assigned a random color.

This format is simple. Sets may be combined as new measurements are taken or divided with filtering or down-sampling. Many simple operations process points independently and can be efficiently parallelized.

In addition to location, each point can hold data like color, temperature, labels, or direction. The collection of point locations and associated data may be treated generally as a set of observations on which to perform statistical processes.

Clustering and statistical filtering are common processing operations on point clouds. Comparing point clouds and locating features within them are active areas of research.

There is no spatial organization within the data structure: points may be inserted in any order. Many useful operations like surface normal estimation or clustering require querying values at points neighboring each pixel. At each point, this requires an exhaustive search of the entire point cloud. Strategies exist to reduce this overhead or precompute reduced search regions, but nothing in the data structure implicitly defines local, geometric relationships.

Humans easily identify meaningful features in a point cloud and understand the implied surfaces and volumes. It is clear to a viewer that the points in Figure 4B depict the Stanford bunny. But higher dimensional geometric elements like edges, surfaces, or volumes are not actually represented by this format. Features like planar regions or local surface normals may be estimated but are not explicitly conveyed.

Many algorithms exist to turn a point cloud into a mesh. These search the point cloud and decide how to add edges between points to define surfaces which can explicitly define volumes. The methods employed by these algorithms vary widely and often have several hyperparameters, the tuning of which affects speed and accuracy. After mesh construction, adding new point cloud measurements requires re-running a meshing algorithm for at least a local region.

Point clouds are minimal yet versatile data structures. Each point directly corresponds to a location measurement and additional data about a point is easily appended. The geometric structure is analogous to a set of statistical observations making it easy to analyze the collected measurements. Their simple format makes no statements about edges, surfaces, and volumes in the point set. Often a reconstruction task requires this information as an output, fortunately algorithms exist to estimate this information and transform a point cloud into a mesh.

2.2.3 Meshes

A polygon mesh is a useful way to express geometry of different dimensionality. A set of points define 0D vertices. Pairs in this set of points describe 1D edges. Sets of edges express

2D faces. If there are no boundary edges, then the mesh describes a 3D volume. A triangle mesh of the Stanford bunny is shown in Figure 4C.

Polygonal meshes are a general implementation where each face may consist of any number of edges. In a triangle mesh, polygons are subdivided until each face has exactly three edges. This format is commonly used in graphics rendering – fixing the number of edges per face allows a variety of optimizations in both storage and computation.

Each point and face can have additional data appended to it. Commonly, color or surface normals are included to help with rendering processes. But other measurements or labels are easily assigned to these features.

A mesh data structure must track three related sets of information: point locations in a list of vertices; pairs of vertices in a list of edges; and a set of edges to create a polygon in a list of faces. Algorithms modifying meshes must be careful not to corrupt relationships between these lists when modifying geometries within a mesh.

Because they can express points, lines, surfaces, and volumes meshes are versatile. But they are also complex. The mathematics field of topology provides the basis for many algorithms to manipulate meshes or describe their properties.

Validating watertightness is a common task. This asks if a mesh describes a volume. Any holes – which imaginary water would pour out of – mean that a mesh does not describe a closed volume with a distinct inside and outside.

It is possible for a mesh data structure to represent geometry which would be impossible to construct in the real world. The errors that cause a mesh to be malform like this are referred to as non-manifold errors. These can result from self-intersecting faces, completely internal faces, or the surface normal of a watertight mesh pointing both internally and externally. Many algorithms prefer to operate on manifold meshes and some require it.

Programs for animation and drafting allow users to precisely create meshes and export them into mesh file format like PLY, STL, OBJ, or more. One can also estimate a mesh based on depth images or point clouds generated from sensor measurements.

Meshes are a useful format for describing geometries in a variety of dimensions. Constructing or updating a mesh is not always trivial – especially if the vertices' source is a point cloud or depth image. Operations on meshes can require the data structure to have specific mathematical properties.

It is common that the output from a reconstruction task is a mesh. This does not mean the reconstruction process must use a mesh as its data representation, only that the chosen representation be convertible to a mesh when needed.

2.2.4 Voxels

Voxels are small, cubic regions of space organized into a regular grid to describe a larger rectangular prism. Each voxel describes one part of a larger 3D volume and their organization into a regular grid means the precise location of each is implicitly encoded by

its index within the grid. An expression of the Stanford bunny in a voxelized format is shown in Figure 4D.

A voxel grid is defined by the resolution of its voxels and the number of voxel elements in each direction. The pose of a voxel grid's lower bound identifies what section of a larger space it describes. Each voxel can store one or more values like color, surface normal, or measured values.

Geometric data in the form of points, lines, surfaces, and volumes can be added to a voxel grid. One must identify what voxels the form lies within or intersects and apply some strategy to update the voxels. The grid's discretization down-samples space and selecting an appropriate update strategy is important. For example, the voxel grid in Figure 4D was created from the point cloud in 3B by averaging the color of each point inside of a voxel.

The organized structure of voxels is like pixels in an image; in addition to width and height, voxels add depth. Alternatively, one could view a voxel grid as a multi-dimensional array or tensor with three dimensions. This means that a voxel grid has a fixed size, regardless of how much data from point clouds or mesh formats has been added to it.

Also like an image, the implicit link between index and location means identifying a voxel's neighbors is a simple index lookup.

Voxels are common in computer graphics, scientific imaging, and video games[2]. Because of the diverse array of uses there are no standard file formats. The representation and storage tend to be application-dependent and specific to what industry a voxel rendering engine or library supports.

In reconstructions tasks voxel grids are a useful way to integrate many measurements into a fixed-size data structure. The voxel explicitly represents volume without managing connected edges and faces like a mesh. This is achieved by discretizing space which can limit the structures spatial resolution. But some approaches like truncated signed distance fields (TSDF) [22] let the structure represent surfaces with sub-voxel resolution. Most of the research in reconstruction with voxel grids is focused on what data to store at each voxel and what update rules to follow when integrating new measurements.

2.2.5 Octrees

A voxel grid sub-divides space equally between all its voxels. But interesting geometries rarely fit a rectangular prism perfectly and large, connected regions of voxels may end up with the same value. An octree data structure addresses this by using voxels of varied sizes organized into a tree-structure.

Each octree starts with a root node which defines a cube in space. Starting from the root, each internal node has exactly eight children which equally divide that node's cube into eight octants.

[2] Realistic rendering for fluids and gases typically uses voxels that light can pass through – rather than surfaces which reflect it.

Figure 4E shows the octree created from the voxel grid in 3D. The larger cubes highlight the regions of the voxel grid where all the data was the same. Rather than a default or empty value for each voxel, the octree stores one node to describe this entire region.

The tree has a minimum depth which defines the smallest resolution of space it can describe. As new data is added, the tree is branched and extended to whatever depth is required. If needed, a tree could be extended up, so its root describes a larger region of space. Tree-based algorithms efficiently search, modify, or condense the octree.

Usually, an octree is a secondary or supporting data structure. For example, in graphics rendering an octree created for a mesh can accelerate ray tracing processes. But the advantages it provides in data compression and the ability to easily increase the bounded region make it a useful representation in reconstruction tasks, especially mapping.

2.3 Ray Tracing in a Grid

Voxel grids are one focus of this work. Adding a single point to a voxel grid is simple – one only needs to convert a point's continuous spatial coordinates to the correct discrete index in the grid. However, adding a line between two locations requires slightly more consideration. It is worth introducing the common algorithm for tracing a ray through a grid to support later chapters where this process is extended for different data representations. The goal of this process is to create an ordered list of indices that the line passes through between its start and end point.

The line is often described as a parameterized ray and checking if it intersects a grid is a natural first step. When the grid is bounded by some coordinates or a maximum number of indices it is possible that many rays miss its region completely. Calculating the traced indices for these rays is inefficient.

To filter out non-intersecting rays, the problem is viewed as checking the intersection of an axis-aligned abounding box (AABB). The AABB intersection is a well-understood computer graphics problem: [23] provides a step-by-step intuition for the solution and several optimizations and [24] presents a full survey of methods for graphics applications.

In general, the ray is transformed into a reference frame where the grid's bounding box is aligned with the axis. From this perspective the ray's parameter value for intersecting each face of the box is easily solved. Comparing the order of intersection parameter values reveals if the box is intersected and the values at which the ray enters and exits the box.

Once a ray is verified to intersect the grid, the Amanatides-Woo algorithm [25] is used to find the ordered list of what voxel indices that it intersects. The algorithm also relies on a parameterized expression of the ray in a reference frame aligned to the grid's axes.

Beginning at the first index – either the ray's origin or when it first intersects the box – the algorithm uses differential analysis to identify which voxel is entered next. It looks at the ray's slope and tracks a parameter for each orthogonal direction in the grid. The process ends when all parameters pass that ray's end point or its second intersection with the box.

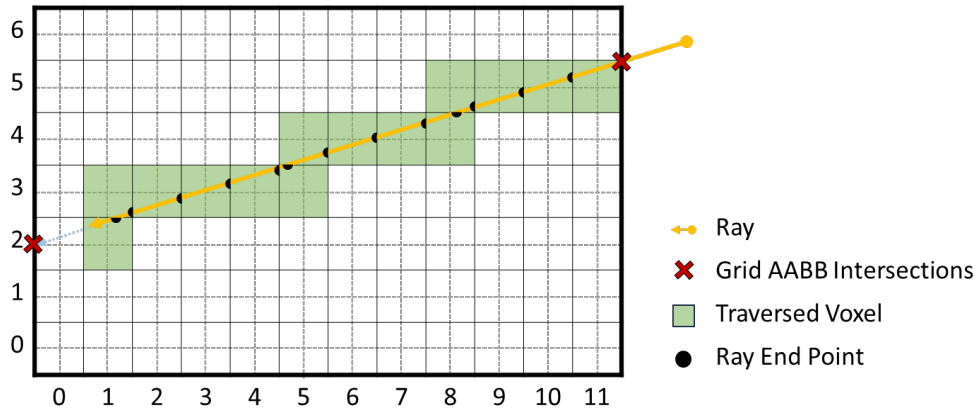


Figure 5 Ray tracing voxels in a 2D grid.

Figure 5 depicts what this looks like for a 2D case. Once the ray's intersections are found, the tracing algorithm begins in the voxel the ray enters at. It proceeds to calculate the parameter required to hit the next voxels in the ray's direction – to the left or below in this example – and chooses whichever direction has a smaller parametric value. The process repeats until the parameter in both direction is greater than that of the ray's end point. This algorithm is easily extended to higher dimensions with additional comparisons.

For voxel grids, adding a measurement from a sensor uses this process. The measured end point depicts where a surface is. But the space between the sensor and the surface must have been free to measure it. Tracking this free space is just as informative as recording the surface measurements.

Chapter 3. Related Work

Scanning and mapping are hardly new challenges. For decades, research has explored reconstruction tasks using various robotic platforms, view-selection policies, and data representations. This thesis builds upon that work. Some concepts, like specific voxel representations, are used directly while others have simply provided inspiration.

Introducing some of the key concepts and important works is valuable framing for the methodologies and results discussed later in this thesis. This chapter begins by exploring view selection policies. It shares why information-driven policies are desirable and describes some methods that generate simulated images to identify what view is best based on what it is expected to see. This chapter also compares specific voxel grid representations that are commonly used in reconstruction tasks and discusses how they are implemented.

3.1 Next-Best View Selection Strategies

Acquiring new measurements is one of the three primary steps of a reconstruction task. A single camera position is unable to see all sides of an object while scanning nor see all features in a scene while mapping. Therefore, a system must follow some policy to decide where to place its sensor next before collecting new data and integrating it into the reconstruction.

A policy must guide the system to explore the reconstruction area while only viewing small regions of it. Policies are also responsible for implementing the stopping condition for a reconstruction task. This could be a pre-determined number of views or some criteria that the reconstruction's data representation should meet.

Research on view selection policies is quite active. Specific policies are generally presented framed as either a mapping or scanning problem. Policies for the former must consider the system's movement through the environment so policies which ensure the proposed views are safely navigable are desirable. However, most policies presented for one framing can reasonably be adapted to another. To keep policies general, some do not attempt to model any specific navigation or kinematic constraints, which are system dependent.

The simplest class of policies operate with no knowledge of the reconstruction's data. Instead, these follow simple rules to propose view locations.

A view sphere is the foundation for many simple policies. It is generally used in scanning tasks and the only information about the reconstruction it requires is the scan region's bounding box. Views are positioned only on a sphere that shares the bounding box's center and are oriented towards this point. From here, one could iterate randomly around the sphere or implement a strategy for uniform sampling.

Similar methods could be implemented for geometries like cylinders (i.e., rotating around an axis, like a robot twisting a wrist joint), boxes, or more.

The policy ends after a specified number of views are added, guaranteeing a relatively constant runtime.

But without using any feedback from the reconstruction data, they are severely limited. There is no guarantee that a policy will view the entire part.

There are cases where these simple policies are useful. The ease of implementing these policies could make them an appropriate choice when accuracy is not the primary concern for a reconstructions task. But a different approach is needed if a policy is expected to perform optimally on a variety of geometries.

Policies for next-best view selection attempt to suggest views which will provide the most new or useful information to the reconstruction. All these policies operate by identifying a set of candidate views that are scored based on quality.

To determine the candidate set and a view's quality, these policies utilize the reconstruction data. This means a policy is generally dependent on a specific data structure. Many use voxels, but some use point clouds [1]–[3], meshes [4], or a combination of geometric data structures [5]–[8].

These policies may also use a model of the sensor to determine a view's quality by simulating an image from a candidate position. Some policies consider a view to be informative because it explores new, unseen areas of the reconstruction. Others prefer views that remeasure surfaces from new perspectives to increase confidence in these measurements. Many attempt to strike a balance between these goals.

Policies which utilize sensor models are referred to as generate-and-test methods [26]. These policies have some domain within which the sensor may be placed. The policy discretizes the domain to limit computation when evaluating views.

Some policies sample uniformly on a view sphere [27]–[29]. Others utilize platform-specific knowledge and sample a robot's configuration space [30].

One subset of policies is frontier-based. These methods track the boundary of seen and unknown space and only suggest views in this region. They are primarily focused on exploration and tend to be framed as mapping problems.

A model-free, frontier-based policy using point clouds is described in [1]. But many more are also generate-and-test methods.

The policies described in [6], [8] both use edges in triangle meshes to identify boundaries in the surface and voxels to differentiate between free, occupied, and unseen space when testing images. [4] describes a mesh-only representation where the boundary faces provided candidate views. The approaches in [31]–[33] all operate with voxel grids.

The method in [31] uses rapid random trees to identify a path of views that maximize exploration when mapping an unknown space. The system only moves to the first position of the selected path, but it retains some of the best paths for the next iteration. This results in much faster exploration than other frontier-based methods.

Other kinds of problems also involve view selection and may be adapted to reconstruction tasks. Image based rendering is a format for compressing 3D models as a set of depth

images and [34] presents an approach for selecting informative views of a known scene. Similarly, [35] outlines algorithms to select descriptive views of voxel grids for rendering scientific data.

3.2 Partitioning Space with Voxels

Computational domains like scientific imaging, video games, and rendering applications all utilize voxels to visualize data and accelerate computations. Section 2.2 introduces and compares common geometric data structure and describes the general advantages of voxel in Section 2.2.4 Voxels.

Since voxels are inherently volumetric and fixed size, they are ideal for reconstruction tasks. Incremental updates with new data are fast and the index-based spatial organization means local searches are straightforward.

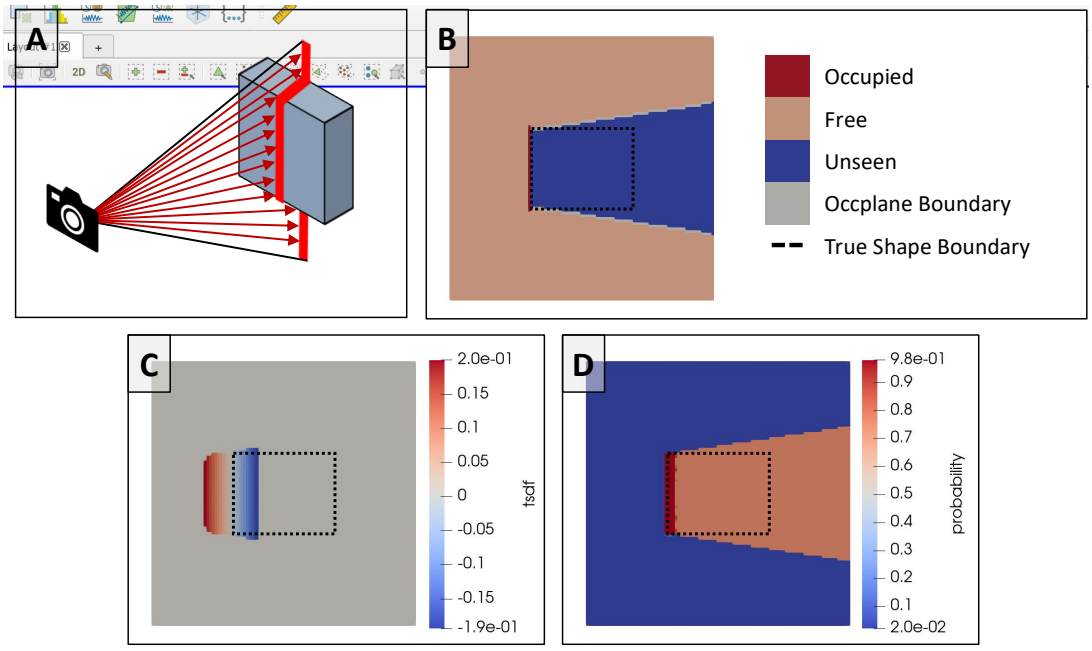


Figure 6 Representing (A) a line of depth measurements in (B) space carving, (C) TSDF, and (D) occupation probability methods.

This work explores three of the most common voxel implementations used in reconstruction tasks: space carving, truncated signed distance fields (TSDF), and occupation probability. Figure 6 shows a 2D diagram for each of these after the same single image of a box shape. All three methods record the sensed surface measurements and the free space on a ray between those and the sensor.

Space carving records a label for each voxel to describe its occupancy – or lack thereof. Initially, all a grid's voxels are labeled as occupied. New measurements can change their labels to non-occupied but cannot change the labels back.

TSDF records a signed value for each voxel that describes its distance along the measurement ray from the surface. This method implicitly records the exact surface of the scene at the grid's zero iso-level. While other methods can estimate a discretized volume that a part occupies, only TSDF can represent the object's surface and can do so with sub-voxel accuracy.

Occupation probability tracks how likely it is that voxels are occupied. If the sensor repeatedly measures a voxel as a surface, then its probability increases. If it is far away from the measured point, then its probability decreases. This acts as both a label – when thresholded – and a measure of confidence.

3.2.1 Space Carving – Binary and Categorical

Space carving is a simple representation that assumes all voxels are initially occupied. When adding new measurements, the voxels on the ray between the sensor and the recorded surface position are labeled as free. Voxels can only move from the occupied set to the free one. As new measurements are acquired, only those voxels on or below an object's surface will remain.

The division into two categories is useful. For tasks downstream on reconstruction, like navigation, knowing what space is free or occupied is typically all that is required. However, some works [29], [36], [37] recognize that sub-categories for these occupied labels are useful.

There are many reasons a voxel might be considered occupied. Initially, voxels are unseen, no ray has passed through them so assuming these are occupied is safe. If a voxel contains a measured point, it could be labeled more specifically as a surface. If a voxel were in the field of view but behind a surface – detected by continuing the measurement ray beyond the surface point – it is occluded.

In [36] two additional labels are introduced. Border voxels are labeled as free and adjacent to an unseen voxel. These describe regions at the limit of a sensor's field of view. Occplane voxels are labeled as occluded or unseen and adjacent to free voxels. These describe regions where there is no information because a surface blocked the view.

Table 1 Space Carving Voxel Labels

Binary Label	Sub-Category	Description
Free	Empty	Voxel was traversed by a measurement's ray.
	Border	Voxel is free/empty and adjacent to an unseen voxel.
Occupied	Unseen	Initial state of all voxels. No measurements included the voxel.
	Surface	Voxel was at the end of a measurement's ray.
	Occupied	Synonymous with surface in some works.
	Occluded	Voxel was on a measurement's ray and behind the measured point.
	Occplane	Voxel is occluded and adjacent to a free voxel.

These categories are summarized in Table 1. It is worth noting that the precise terminology of these sub-categories can differ slightly between works and not all works use every label.

Space carving is notable for its simplicity and explicit categorization of voxels. However, it handles disagreement between measurements poorly. Errors in the sensor's intrinsic model and uncertainty in its pose or measurements can cause the same surface measurements to appear in different voxels. The method simply overwrites a voxels prior label with the latest measurements.

3.2.2 Truncated Signed Distance Fields

For scanning tasks in particular, the reconstruction's goal is an estimation of the object's surface. All voxel grid methods inherently describe volumes and can be partitioned into a set of free or occupied voxels. But extracting the precise surfaces that the camera observed is not always possible. Recognizing this, [22], [38] introduce TSDFs which utilize signed

distance fields (SDFs) to implicitly represent the observed surfaces from a set of depth measurements.

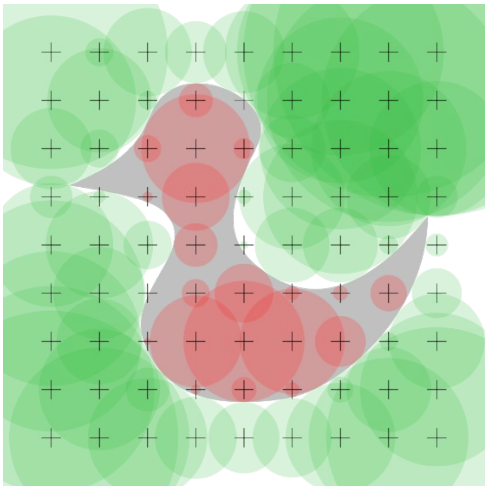


Figure 7 A simple signed distance field.

SDFs are functions which define the orthogonal distance of a given point from some boundary. A positive signed component means the point is external to the bound region while a negative sign is internal. Figure 7 shows an illustration of an SDF.

In reconstructions the updates from a measurement's ray are truncated within some positive and negative distance of the measured surface. These distances should be great enough to capture the entire noise distribution. However, making them too great may lead to needlessly updating voxels far from the surface that are less informative than those near the boundary.

As a new measurement is added, a weighted update is performed for each voxel the ray intersects:

Equation 1 TSDF Voxel Update

$$W_i(x) = W_{i-1}(x) + w_i \tag{1.1}$$

$$D_i(x) = \frac{W_{i-1}(x)D_{i-1}(x) + w_i(x)d_i(x)}{W_i(x)} \tag{1.2}$$

All voxel distances and weights are initialized to zero. $D_i(x)$ denotes the signed distance at a voxel and $W_i(x)$ its weight after i updates. Each ray that passes through the voxel updates the weight and distance with the voxel's truncated distance on the ray $d(x)$ and a weight $w(x)$ for the update.

The weight may be set to one to average all distance observations or it may be used to encode uncertainty. A weighted function that linearly decreases from one to zero as the negative values become closer to the negative truncation distance is used in [39]–[41].

Equation 2 TSDF Linear Weighting Function

$$w(x) = \begin{cases} 1 & d(x) \geq 0 \\ 1 - \dfrac{d(x)}{d_{neg}} & else \end{cases} \tag{2.1}$$

Projecting the ray beyond the sensed point is fundamental to this method. But this creates an assumption about unseen space and can lead to incorrect results around corners, small features, and thin parts.

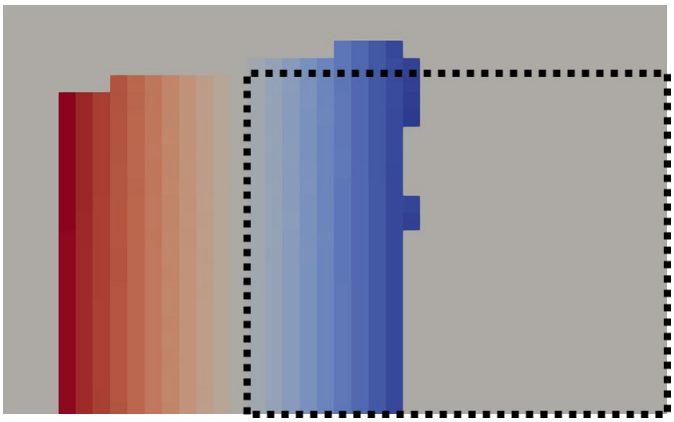


Figure 8 TSDF Projection Corner error.

The error created at a corner from one update is displayed in Figure 8. There are several strategies one could use to mitigate this – skipping updates near discontinuities in the depth image, adding more views, or adjusting the update weighting function. But there is no way to completely avoid any artifact corners [40].

The TSDFs implicit surface may be extracted at any time with the marching cubes algorithm [42] or a number of similar approaches. A complete overview of TSDFs and their use in robotic reconstruction is provided by [40].

3.2.3 Occupation Probability

Recording occupation probability is one of the most common representations, especially for mapping tasks. Like space carving, the voxels in this method describe if space is free or occupied. Rather than discrete labels, each voxel records a probability value that describes how likely it is to be occupied.

The approach used was first described in [43]. A probability profile is defined for the sensor and each measurement applies these probabilities to the voxels it intersects.

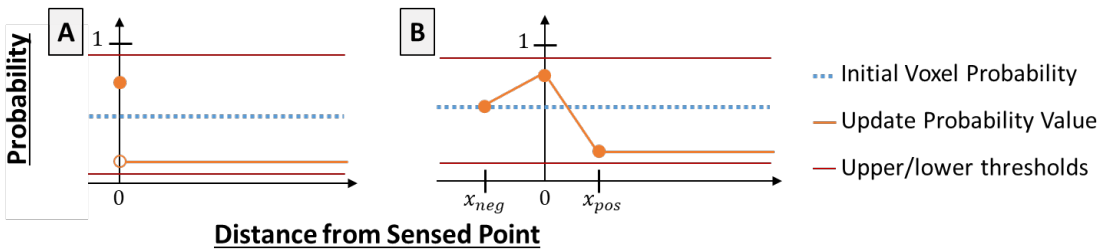


Figure 9 Occupation probability ray profiles (A) delta and (B) linear, piecewise functions.

The profile used encodes some assumptions about the sensor's accuracy and two examples are shown in Figure 9. The simplest profile is a delta function: the voxel containing the measured surface point is updated with a high probability while all others between this location and the sensor are updated with a low one. However, many other profiles could be used. For example, a piecewise function might better model a sensor's noise distribution in the same way the TSDF's truncation distance does.

Each voxel is assigned with an initial prior probability, for example $P(x) = 0.5$. The resulting probability of a voxel after a set of observations $z_{1:t}$ may be estimated using a log-odds representation [44]:

Equation 3 Occupation Probability Log-odds Update

$$L(x|z_{1:t}) = L(x|z_{1:t-1}) + L(x|z_t) \tag{3.1}$$

$$L(x) = \log \frac{P(x)}{1 - P(x)} \tag{3.2}$$

Expressing the probability in log-odds results in addition rather than multiplication. This expression is not only mathematically simpler but computationally preferable for speed and numeric stability.

Setting a maximum and minimum probability threshold puts an upper bound on how strongly a voxel can believe it is occupied. This indirectly controls the number of updates required to change the state of the voxel. If the environment changes, a low upper threshold allows just a few votes to change voxels from occupied to free.

OctoMap, [44] is one of the most commonly used reconstruction frameworks and has popularized the occupation probability representation. It uses an octree representation but many of the update strategies remain the same when using voxels.

This approach fuses noisy data better than space carving can. Setting an occupation probability threshold can divide the grid into occupied and free voxels. But the space carving method's sub-categories cannot be applied.

Chapter 4. Methodologies

4.1 ForgeScan

This work recognizes the need for a voxelization library that is simultaneously specific to autonomous reconstructions tasks and flexible for the research and development of new methodologies. In response, ForgeScan, a well-documented, C++17 library with a small set of dependencies, is proposed. All code is available at [9].

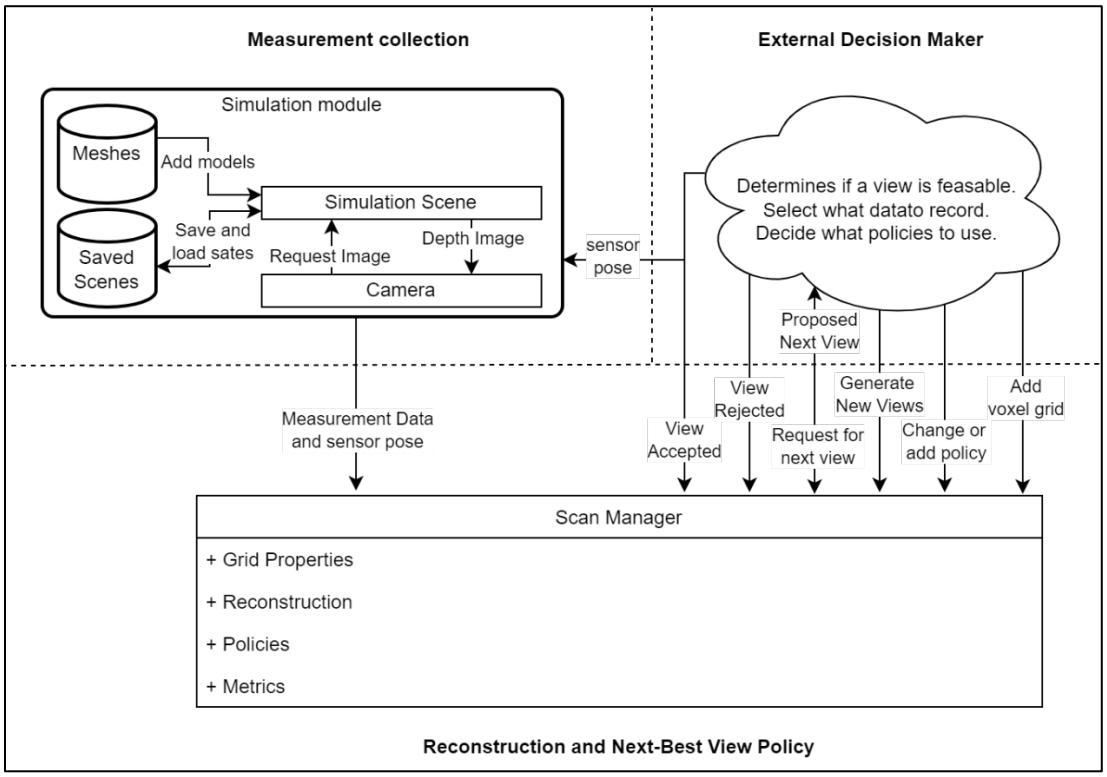


Figure 10 Simplified collaboration diagram between ForgeScan and an external measurement collection and decision maker.

At its simplest, ForgeScan is designed around a Scan Manager class that collaborates with two external modules, one for measurements and one representing an external decision maker. A simplified model of this is shown in Figure 10.

The Scan Manager contains a few key members:

- **Grid Properties** describes the voxel grid size and resolution.
- **Reconstruction** manages all voxel grids, each using the same Grid Properties.
- **Policies** is a set of next-best view selection policies. The external decision maker can select a policy to use or change between multiple policies.
- **Metrics** record some meta-information from a Policy or the Reconstruction as the process runs.

The measurement module represents a sensor and scene which provides data to the Scan Manager. ForgeScan includes a simulation module to generate depth images from meshes. This provides a straightforward way to evaluate policies and reconstructions without real sensor data. This module is described in more detail in 4.1.4 Simulation Module.

The decision maker represents an external client, like an autonomous system, using ForgeScan to generate a reconstruction. It can specify what data to record in one or more voxel grids and add or change between different view-selection policies.

Through the Manager, the client can request the next-best view using whichever policy is active. With specific knowledge of its implementation, it decides to accept a view or reject it (based on criteria like reachability, distance, or other client-specific knowledge). This

allows a simple form of negotiation. The Manager records which views were accepted and rejected for each policy.

Measurement collection is a second external source. In practice, this could be part of the autonomous system acting as the decision maker. However, modeling these as distinct parts allows the injection of measurement data through ForgeScan's light-weight depth camera simulator or from a set of externally recorded poses and depth images.

From whatever source, the Manager accepts a depth image with camera intrinsics or point cloud and the sensor pose, relative to the Reconstruction's lower bound. It passes this information to Reconstruction, which updates its grids with the voxels these measurements traversed.

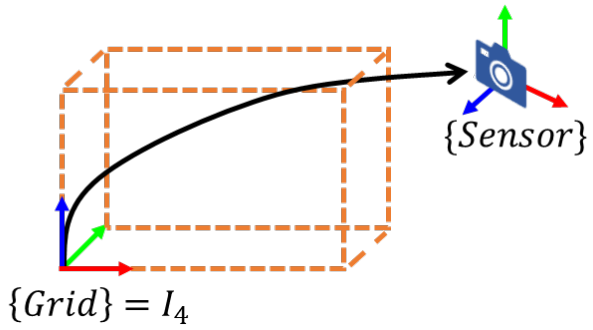


Figure 11 The use of reference frames in ForgeScan.

Internally, all grids in one reconstruction use the same properties. This means they share the same lower and upper bounds. This greatly simplifies the integration of new data and ensures each voxel grid is comparable.

A system may place the actual reconstruction and sensor relative to any static or dynamic reference frame specific to its task. ForgeScan only needs their relative poses.

To provide easy addition of policies and voxel grids at runtime, all core classes in ForgeScan support string parsing to specify parameters. Users can, through the command line, interactively and expressively describe what to construct.

Similarly, systems are not required to know any specific class constructors or special message formats. The only communication requirement is that they create strings with the appropriate flags and options.

4.1.1 Grid and Reconstruction Classes

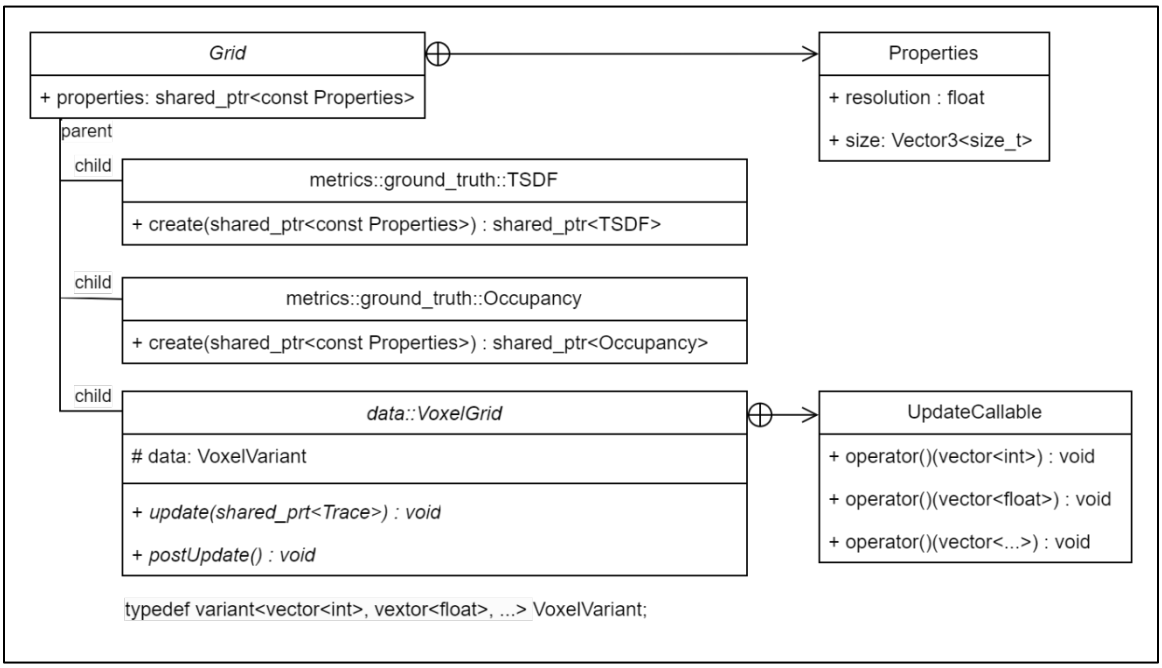


Figure 12 Simplified inheritance diagram for Grid classes.

The abstract class Grid is the foundation for the voxelized data structures in ForgeScan. It defines the fundamental properties of voxel resolution and grid size – the number of voxels in each direction – and provides logic for accessing voxel indices. A simplified inheritance

diagram is shown in Figure 12. The classes Occupancy and TSDF implement Grid for ground-truth metrics (see 4.1.3 Metrics and Scene Classes).

The VoxelGrid class inherits from Grid too. As an abstract class, it provides an interface for specific voxel grid methods to implement. It also provides abstraction for what datatype the grid operates on.

The member variable data is a variant of vectors for float, double, and unsigned and signed integers of 8, 16, 32, and 64-bit. The desired datatype is a parameter during the construction of a specific Grid implementation. A voxel grid implementation may explicitly declare what it supports in its call to the VoxelGrid constructor. For example, the TSDF is designed to operate on floating point types.

The subclass UpdateCallable provides a callable interface for the C++17 visit function to select the data-type appropriate update function at runtime. Concrete voxel grids are required to define their own callable as a subclass of UpdateCallable. The subclass provides function overrides that implement the grid's desired update method only for the datatypes it supports.

The Reconstruction class manages the voxel grids. It uses the same grid properties as the Manager and passes these to each voxel grid. This ensures every grid in the same reconstruction has the same resolution and size. This simplifies ray tracing and makes it easy to directly compare grids.

4.1.2 Policy Classes

View selection policies are another core element of ForgeScan. They generate a set of proposed camera poses. As represented in Figure 11, these poses are relative to the common voxel grid origin. The decision-making system can perform transformations to express this position in whatever format it requires to move the sensor.

One or more policies can be added to the Scan Manager with one always designated as active. Calls to generate a new set of views or retrieve the next best view in a set go to the active policy.

If a system deems a view undesirable – out of reach or too far – it can signal that the currently proposed view is unacceptable. In this case, the active policy moves this view to a list of rejected views and either proposes the next view in the current set or regenerates a new set of views. Any accepted views for a policy are recorded in a second list.

When generating a set of views, a policy's algorithm may utilize any of its previously accepted or rejected views. A policy can also view any voxel grids in the reconstruction. If the user has not added the type of grid it requires to run, then it may add this.

Most of this operation is defined in an abstract Policy class which user-defined policies must inherit from. Some simple open-loop policies are defined already. These include methods based on spherical sampling or rotation around an axis. Othe policy methods are described in 4.3 Occplane Clustering and 4.4 Precomputing Views.

4.1.3 Ray Tracing

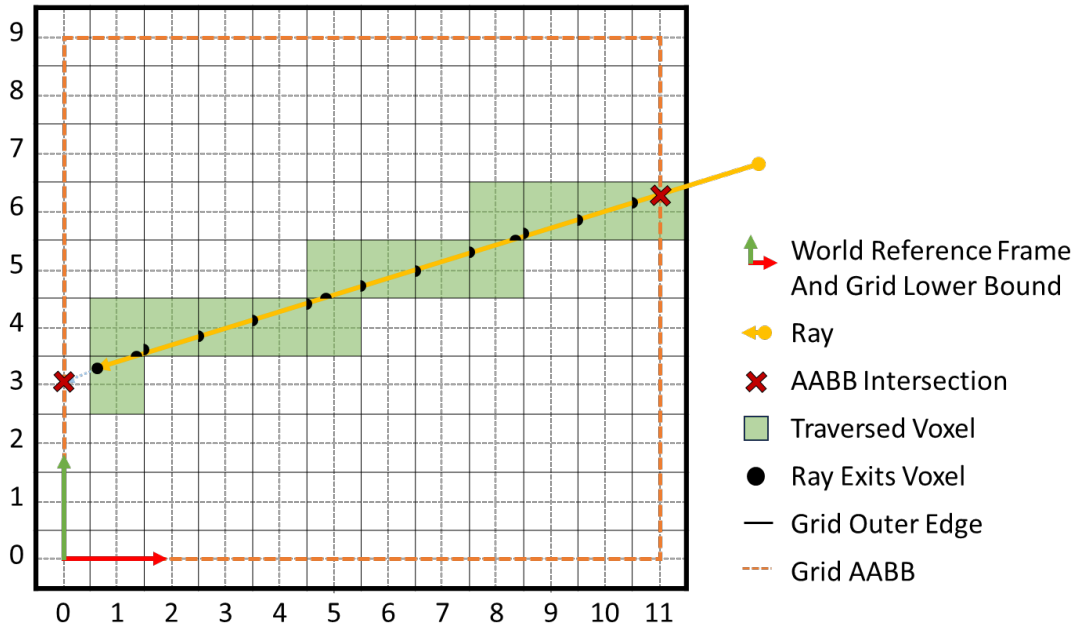


Figure 13 Ray tracing in ForgeScan.

All three common voxel representations utilize a measurements ray rather than just the end point. The general method for tracing a ray through a grid is described in 2.3 Ray Tracing in a Grid. The implementation in ForgeScan builds upon this slightly. The process is shown in Figure 13.

The center of a grid's first voxel is located at the world origin. Its bounding box begins at the origin and ends at the center of the grid's final voxel. Drawing the bounding box between centers reduces its overall size by a half of the voxel's resolution in each dimension. Doing so ensures that a traced ray always begins firmly within the voxel grid. It filters out any rays that only clip the boundary and provide little information anyway.

The ray's direction in the Amanatides-Woo algorithm is scaled so the parameter value is in the same units as the reconstruction. This distance is recorded for each voxel. The ray is

then traced beginning at its end point – either the sensed location or a point projected beyond it – and progressing to the sensor origin. This progression tracks the distance on the positive side of the ray slightly better.

4.1.4 Simulation Module

As a stand-in for real data, ForgeScan includes a module for simulating depth images. It consists of a class to define what objects are in a scene and a pinhole depth camera to image these objects.

The Scene class utilizes Open3D [45] to load triangle meshes. Each mesh may be scaled and repositioned when loaded. A specific configuration of meshes within a scene may be saved and re-loaded. Rays can be cast into the scene to collect simulated measurements and test for occupancy or signed distance from a single watertight mesh.

The Camera class implements a pinhole depth camera model. A collection of rays may be cast into the scene to simulate a depth image. Its images and position may be passed to the Manager class just like a real sensor's image and position.

Testing methods against measurement error is important. Real cameras experience errors based on triangulation, pattern matching, signal analysis, or optics. Because the Camera class directly measures distances rather than modeling physics, it does not attempt to model any of these specific errors.

Instead, a simple but general error model is used. Users define p, a percentage accuracy, for the camera's depth measurements, $d(x)$. A uniform distribution is sampled for a scaling factor that is applied to each measurement.

Equation 4 ForgeScan Depth Camera Noise Model.

$$d(x) = d(x) + d(x) * \mathcal{U}(-p, +p) \tag{4.1}$$

4.1.4 Metrics and Ground Truth Classes

The Metrics class hierarchy provides the option to observe some aspects of the policies or reconstructions as the system runs. Presently, the primary use is to compare a reconstruction's voxel grids against ground-truth data in a voxel grid. This provides an insight into the information gained of each new view from a policy and the present accuracy and precision of each voxel grid.

To perform this comparison, the simulation module, described in Figure 10, is utilized. A Scene class can read triangle mesh files from memory and apply affine transformation to each mesh. The mesh file locations and transformation can be saved to and loaded from memory so the same configuration may be reused.

The GroundTruthScene extends this class functionality. By accepting a lower bound location and voxel grid properties it can calculate what voxels are occupied and the signed distance between each voxel's center and the closest triangle in the mesh.

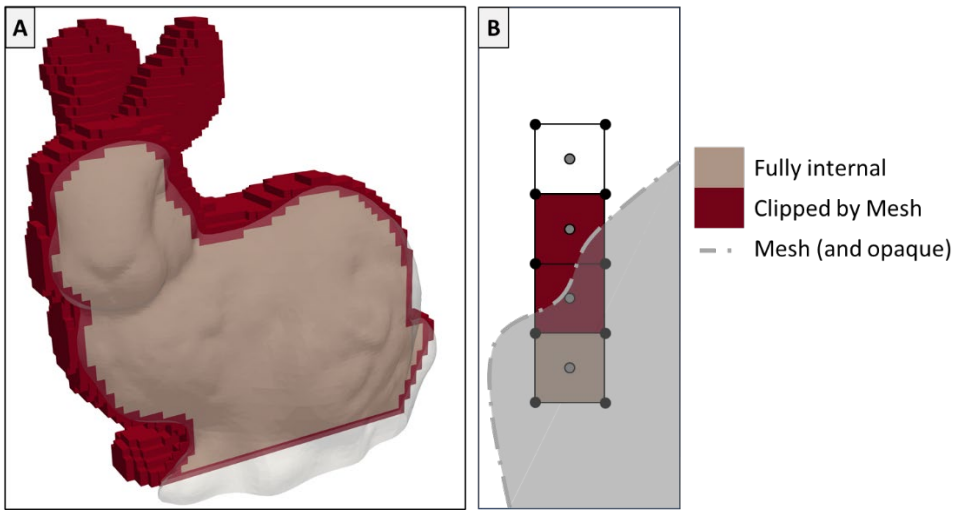


Figure 14 Ground truth voxelization of the (A) Stanford Bunny mesh and (B) an example mesh showing the two ways a voxel may be clipped.

Occupation may be divided into two binary states: occupied or free. But when a mesh is discretized, some of the occupied voxels may be fully internal but others might be clipped. Figure 14A shows this on the Stanford bunny. A clear boundary of clipped voxels exists on the meshes surface. In 13B, the two cases for clipping are shown along with a fully internal and external, or free, voxel.

Dividing occupied voxels into the categories of fully internal or clipped is useful when evaluating a reconstruction method. The clipped voxels in a ground truth occupation grid represent the location of the surface. In an ideal case where a sensor has no uncertainty and its position is exactly known, then all measurements will be on this mesh surface and thus in these clipped voxels.

For scanning tasks, one might be more interested in identifying the surface voxels correctly than the internal ones. One could further sub-divide clipped voxels based on the number of

internal vertices and location of the voxel center. However, for the sake of this project, a distinction is not made.

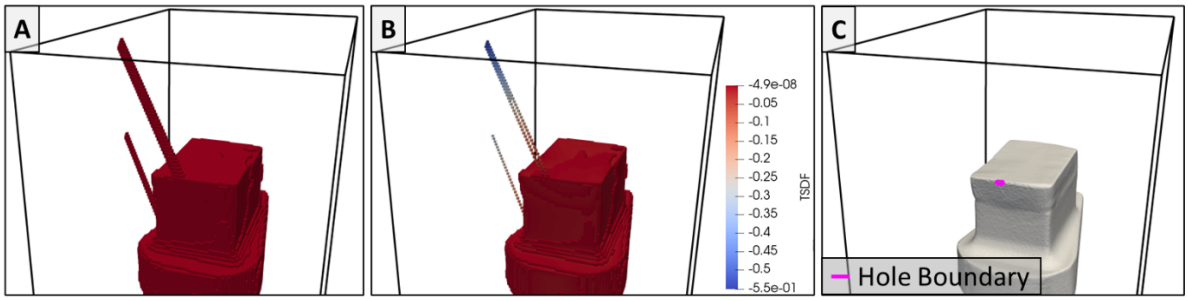


Figure 15 Erroneous ground truth (A) occupancy and (B) TSDF and (C) the source mesh with the largest hole highlighted.

When generating ground truth data, it is important that a mesh be watertight. The presence of holes (or of intersecting meshes) confuses the notion of inside versus outside. This is especially an issue if the source mesh was generated from a manual 3D scan of a part, like in Figure 15. The result is a line of voxels, starting at the hole locations, in both the occupancy and TSDF data that incorrectly believe they are internal.

4.1.5 Storing and Saving Data

Unlike other methods, data compression and transmissibility are not primary focuses of ForgeScan. As a development tool, easy and direct access to any data element is preferred, so values are stored explicitly and generally implemented in a vector-like container. While many voxels far from a region of interest may be empty or contain uninteresting data, it is easily written to a HDF5 format. A second XML file allows tools like ParaView to interpret the HDF5 contents as geometric information.

4.2 Occupancy Confusion

Each representation method has a unique way of describing the scanning space. This makes it challenging to directly compare voxels between methods. The values inside a TSDF and occupation probability grid have no direct relative meaning. However, thresholding these grids divides them into distinct, binary groups.

Positive distances indicate TSDF voxels outside of the part while negative ones describe internal volumes. In the occupation probability grid, a probability value is used. Values above this are considered occupied while the rest are labeled as free. The value is defined by the use and describes how confident they are in the representation. The labels of a categorical space carving grid are also directly convertible to these binary types.

Voxels not seen by any representation are considered occupied. These voxels could be occupied and internal to an object or free but occluded in all views. With no data to make a judgement, this conservative approach is safer than the alternative.

Table 2 Confusion Matrix for Voxel Occupancy.

		Measured Label	
		Occupied	Free
True Label	Occupied	True positive (TP)	False negative (FN)
	Free	False positive (FP)	True negative (TN)

Now, two representations with similar grid properties (resolution and dimensions) are evaluated against each other. The best approach is comparing each to a common ground truth grid. Table 2 outlines a confusion matrix used for this voxel-wise comparison.

Table 3 Confusion Matrix Metrics

Metric	Equation	Description
Accuracy	$Acc = \frac{TP + TN}{TP + TN + FP + FN}$	Percent of correctly labeled voxels
Precision	$Pre = \frac{TP}{TP + FP}$	Percent of labeled-occupied voxels that were correct.
Selectivity	$Sel = \frac{TP}{TP + FN}$	Percent of truly-occupied voxels that were detected.
Specificity	$Spc = \frac{TN}{TN + FP}$	Percent of truly-free voxels that were detected
Balanced Accuracy	$Ba = \frac{1}{2} * (Spc + Sel)$	Selectivity and specificity are averaged when one category is more prevalent than the other.

Several descriptive metrics may be derived by comparing a reconstruction's predictions against the true labels and are described in Table 3. Calculating these metrics after a reconstruction is performed is a useful way to compare how well different data representations model the scene.

These metrics may be evaluated at any point in the reconstruction process. Tracking the value at each step, and how many views are required for them to converge to a final value, describes a policy's performance, allowing policies to be compared too.

4.3 Occplane Clustering for View Selection Strategy

This work introduces a new next-best view selection algorithm for scanning reconstruction tasks. The policy's goal is to achieve full coverage of a part in as few views as possible. It utilizes a reconstruction with a categorical grid and searches for a view where the sensor will collect information about as many previously unseen voxels as possible.

In the categorical grid, voxels labeled as occplanes are of particular interest. These are a kind of unknown voxel that have some neighbors labeled free and others labeled as unknown. They create a surface between these regions.

The location of these occplanes is a simple heuristic for both coverage and information gain. A view of an occplane surface will either identify more of the part's surface or categorize a great number of voxels as free.

From the reconstruction's categorical grid, the method requests the voxel center and local surface normal for each occplane voxel. From each occplane, a point is projected some distance from its center along the normal. The resulting set of points is then treated as a point cloud and clustered using DBSCAN [46]. The clusters are ordered in ascending size and the suggested view is placed at the cluster's average location and oriented opposite to its average surface normal. Algorithm 1 summarizes this process:

Algorithm 1 Occplane View Selection.

```
INPUT offset
INPUT minClusterSize
INPUT minNoiseDistance
DEFINE getNextView
  centers, normals = grid.calculateOccplanesAndNormals()
```

```
  points = centers + offset + normals
  clusters = DBSCAN(points, minClusterSize, minNoiseDistance)
  maxCluster = clusters.largestCluster()
  avgPoint   = [0, 0, 0]
  avgNormal  = [0, 0, 0]
  FOR each point, normal in points, normals
    IF point in maxCluster
      avgPoint += point
      avgNormal += -1 * normals // Invert so view faces object
    END IF
  END FOR
  avgPoint  /= maxCluster.size()
  avgNormal /= maxCluster.size()
  view = Identity(4)
  view.setPosition(avgPoint)
  view.alignOpticalAxis(avgNormal)
  RETURN view
END DEFINE
```

The offset distance is how far all occplane center points are projected along their normals before the clustering step. This helps create clear, distinct groups. The offset parameter should be selected with respect to the sensor's measurement range.

The DBSCAN parameters describe the minimum number of points to form a cluster and minimum distance for a point to join a cluster.

This algorithm can be extended to record a set of top views rather than just the best, only re-running the full algorithm once the top set has been exhausted.

4.4 Precomputing Views for an Assumed Scene

It is common in industrial scanning tasks to have some assumptions about a scene before performing a reconstruction. In scanning tasks, this could be from a CAD model, a prior scan, or an approximate bounding surface. Especially if similar scans are performed

repetitively, pre-calculating a set of views is more efficient than beginning from scratch each time.

A good, precomputed view should contain as much information as possible while remaining far away from all other views in the set. The following generate-and-test policy outlines a simple way to construct this set.

The model geometry is loaded into a simulation scene and placed in the bounded reconstruction region with a similar pose to the real part. A view sphere located at the center of the region is used to generate a large set of candidate views at a fixed radius. All are oriented at the center of the sphere.

Rather than depth, surface normals are imaged. In the simulation these values are readily available. The similarity between the views' optical axes and the surfaces' normals in its image are used as a basis to score the image quality of each view. A minimum similarity value prevents views with high angles of incident from scoring too well. This is described in Algorithm 2.

Algorithm 2 View Image Quality Score.

```
INPUT minimumSimilairty
FOR each view in candidateViews:
  normalImage = generateSurfaceNormalImage(view)
  score = 0
  FOR each pixel in normalImage
    similarity = dot(pixel.surface_normal, view.optical_axis)
```

```
    IF similarity >= minimum_similairty
      score += similarity
    END IF
  END FOR
  View.quality = score
END FOR
candidateViews.normalizeQualityScores()
```

After scoring each view, the values are normalized. The highest-scoring view is selected as the initial view in the precomputed set. Algorithm 3 is used to add additions views to the set:

Algorithm 3 Precomputed Set Generation.

```
precomputedViews = [candidatePoses.popHighestQualityScore()]
INPUT numViews
INPUT weight // In range (0, 1)
WHILE precomputedViews.length() < numViews
  FOR each view in candidateViews
    minimumDistance = 1
    FOR each selected in precomputedViews
      distance = 1 – 0.5*(dot(selected.optical_axis, view.optical_axis) + 1)
      IF distance < minimumDistance
        minimumDistance = distance
      END IF
    END FOR
    view.combinedScore = view.quality * weight + (1 – weight)*minimumDistance
  END FOR
  nextView = candidateViews.popHighestCombinedScore()
  precomputedViews.append(nextView)
END WHILE
```

Since each view is placed on the same sphere and oriented at the center, the similarity of the optical axes is proportional to distance between two views. This value is scaled to range $[0, 1]$ and flipped so less similar views have higher values. A weighting parameter can adjust how influential the quality and distance are on the view's final score.

Chapter 5. Results

The selection of sensors, view selection algorithms, and data representations are critical design choices that impact the model generated by a reconstruction system. It is appropriate to view these as distinct elements when discussing how a reconstruction system operates. But designing a reconstruction to meet specific goals requires a holistic analysis.

Rather than specific algorithms, system design typically has more abstract goals: minimize cost, increase speed, reduce weight, or so many more. The direct influence of sensors, data representations, and algorithms on these goals is opaque.

This design process is often a matter of trade-offs. Can a better view selection policy speed up the reconstruction time? Would a different data representation model the environment more accurately or is a better sensor required? Is the hardware powerful enough to store complicated data structures or execute computationally expensive view selection algorithms? Answering questions like these requires a holistic understanding of each component along with the task's goals and operating environment.

Attempting to identify a single combination of policy, representation, and sensor that is best for all cases is not particularly useful. Instead, this chapter explores general rules by experimenting with these elements for scanning style reconstructions. These

reconstructions are performed at varying levels of sensor noise for specific combinations of view selection policy and object geometry.

To begin, the sensor noise model is demonstrated and used to show how the space carving, TSDF, and occupation probability methods react to uncertainty and derive information from multiple measurements. Extending this, basic spherical and bin geometries are reconstructed with uniform, spherical sampling for increasing amounts of sensor noise.

View selection policies are compared on a single test geometry. The performance is evaluated based on the number of views required for a representation to reach peak precision and minimum miss-rate. This is done for three simple policies as well as the occplane and precomputed policies discussed earlier in this work.

While each experiment discusses some general lessons learned, it is critical to remember that performance is also dependent on the nature of the geometry being scanned. This work does not attempt to characterize these geometries, rather it demonstrates how these experiments were carried out and evaluated. With access to ForgeScan, similar investigations may be performed for more specific objects of interest.

5.1 Voxel Comparisons

A reconstruction's data representation provides a format within which all collected measurements are stored. It is only a model of the real space being measured and choosing the right model for the system's goals and sensing abilities is important.

All voxel representations distinguish occupied space from empty. But only methods like TSDF can extract an estimation of surface locations. Others, like occupation probability, describe a level of confidence in the measurements of individual volumes.

It is safe to say that each of these methods performs well in idea conditions. With no error in the sensor locations or measurements, the reconstructions model ought to agree with the true scene. But this is not realistic outside of pure simulation environments, so understanding how robust these methods are is important for any real implementations.

Using the sensor error model described in 4.1.4 Simulation Module, a set of geometries were reconstructed with space carving, TSDF, and occupation probability for increasing amounts of sensor noise. An accuracy of 2% is stated for the Intel RealSense D455 camera [11], so this amount of noise is used to represent an ideally calibrated camera. A noise of 6% is used to demonstrate a camera in less ideal conditions or slight miscalibration. At a distance of 2.5, these errors mean the simulated camera is accurate to ± 0.05 and ± 0.15.

Table 4 TSDF and Occupation Probability Update Distances

Noise	**TSDF**	**Occupation Probability**
None	$x_{neg} = -0.06, x_{pos} = +0.06$	$x_{neg} = -0.06, x_{pos} = +0.03$
2%	$x_{neg} = -0.06, x_{pos} = +0.06$	$x_{neg} = -0.06, x_{pos} = +0.03$
6%	$x_{neg} = -0.18, x_{pos} = +0.18$	$x_{neg} = -0.18, x_{pos} = +0.10$

The distance parameters used in these experiments are described in Table 3. For occupation probability, all voxels are initialized to 60%. The update profile is a linear, piecewise

function starting at $p(x_{neg}) = 60\%$, increasing to $p(0) = 80\%$, then decreasing to $p(x \geq x_{pos}) = 10\%$. To threshold, any voxels above 51% are considered occupied.

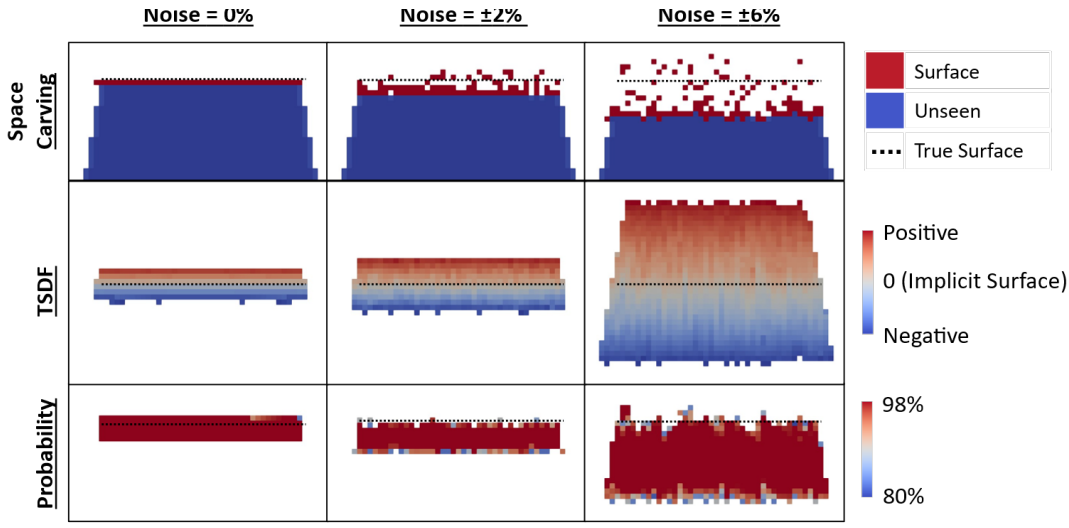


Figure 16 Single measurement of flat surface with increasing noise.

To demonstrate the error model, a single image against a flat surface is reconstructed with all three methods as sensor noise increases. This is shown in Figure 16. Each method responds slightly differently as the error increases. Space carving performs worst as it can only remove voxels. The TSDF and occupation probability methods respond better. However, both overestimate the distance slightly, pushing the zero-level and occupied voxels past the true surface location.

In this example, the sensor's measurement rays are dense – each voxel is intersected by about 30 of them. However, it is still only one view. The TSDF and occupation probability methods are designed to integrate multiple images into one representation. Understanding how these operate as the number of voxel updates increases is important.

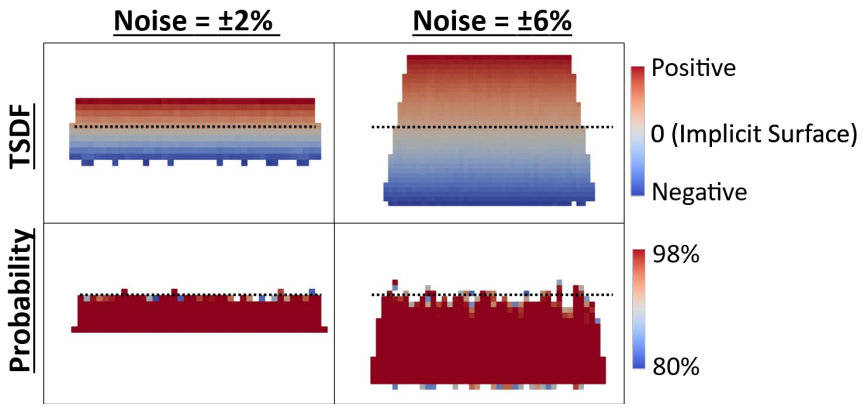


Figure 17 18 repeated measurements of a flat surface with increasing noise.

Figure 17 shows how the TSDF and occupation probability change with 18 images from the same perspective. Now, each voxel is intersected by about 600 rays.

For the TSDF, the weighted averaging strategy generates a smooth gradient from positive to negative voxels and the band of near-zero values is widened. Despite the wide region, the zero-crossing locates the true surfaces well. This demonstrates how, with enough measurements, the TSDF's update strategy can identify the average surface location.

As more data is integrated into the occupation probability grid it does not change much; the first occupied voxels are just behind the actual surface location. This implies that just a few updates will achieve the occupation probability representation's goal.

5.1.1 Sphere and Bin Geometries

Spherical and bin geometries provide a useful way to evaluate representations against precisely defined analytical geometries. The sphere has a diameter of $\frac{\pi}{8}$ and the bin has dimensions of $\left[\frac{\pi}{4}, \frac{\pi}{6}, \frac{\pi}{5}\right]$. On the positive Y face of the bin, a cutout of depth $\frac{\pi}{5}$ is made and positioned so the part has wall thicknesses of 0.02, 0.05, 0.10, and 0.20. Both are reconstructed in grids with $[101,\ 101, 101]$ voxels. The sphere's reconstruction uses a resolution of 0.01 while the bin uses a resolution of 0.02. Both are in the center of their reconstruction region and views are uniformly sampled at a radius of 2.5 from the center of the region.

The sphere should be easy to reconstruct; it contains no unique features. Any view has the same quantity and orientation of measurements.

The bin contains sharp corners, thin walls, and a large concave region. These features are difficult for a representation to capture since they are visible only from specific views.

The analytical geometries are discretized into voxel grids with the same dimensions and resolutions as the reconstructions. The same parameters described in Table 3 are used for these experiments.

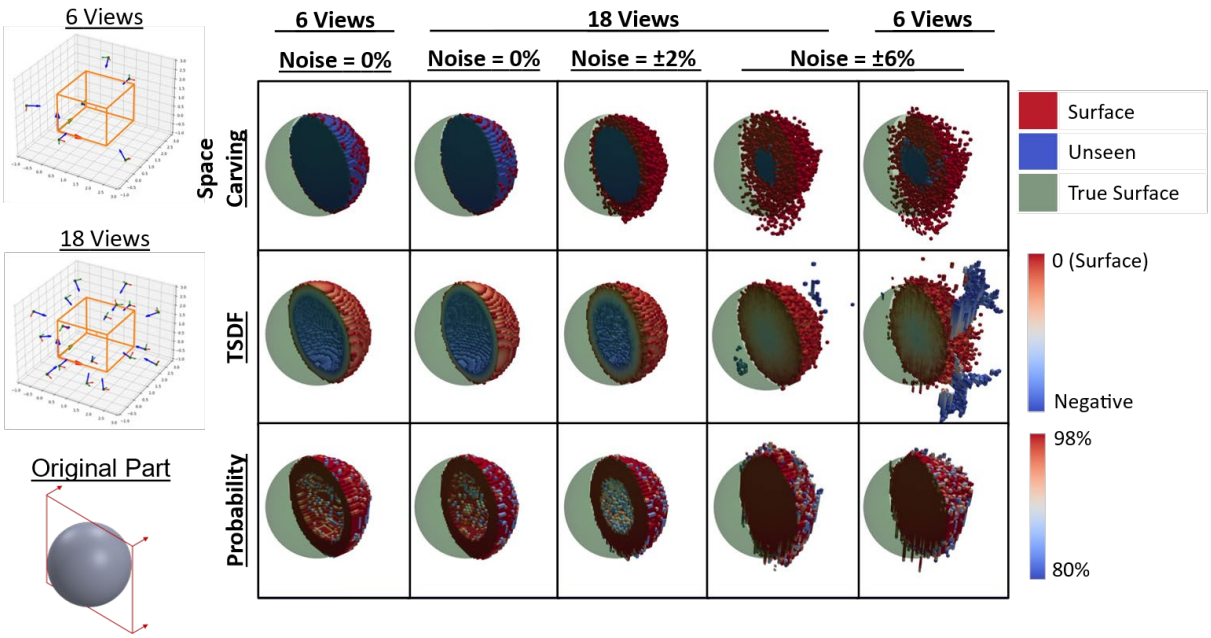


Figure 18 Reconstructions of a sphere with increasing sensor noise.

Figure 18 displays the results of this study for the sphere. One half of the analytical geometry's surface is shown in transparent green. As expected, increasing noise makes the surface location ambiguous. However, TSDF and occupation probability are improved as more views are added.

Comparing the results at 6 and 18 views for the TSDF method highlights how this representation requires more measurements as noise increases. With only a few views, large streaks of negative-distance voxels appear. While these have low weights for the averaging algorithm, not enough rays had positive-truncated regions that intersected these voxels.

The presence of these artifacts can disrupt the TSDF meshing algorithms, creating erroneous and isolated surfaces. More views eliminate some but not all isolated regions.

Post-processing on the voxel grid or mesh could identify and correct some of these but preventing errors is always preferable to correcting them.

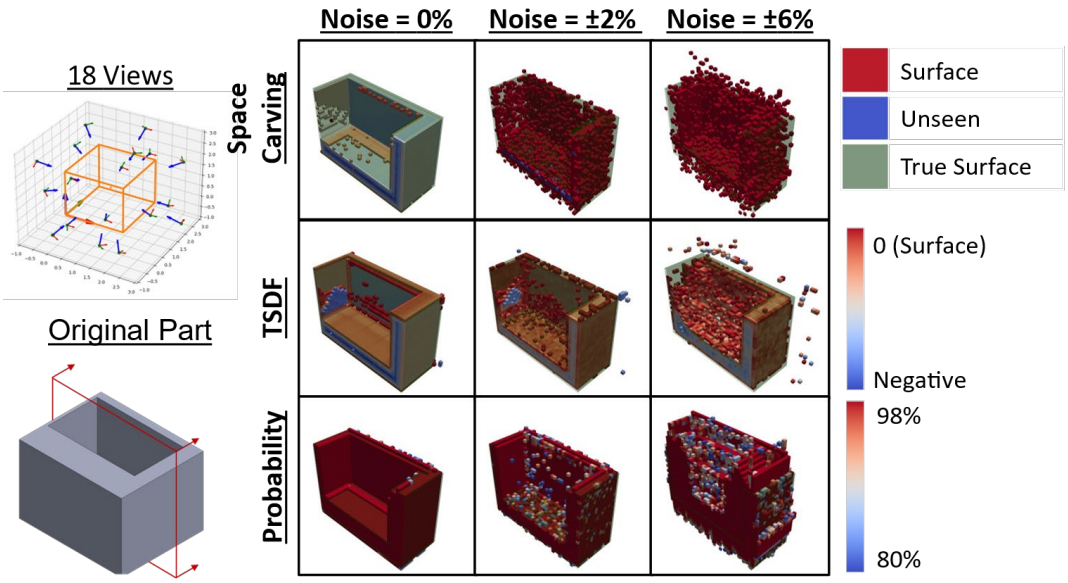


Figure 19 Reconstructions of a bin with increasing sensor noise.

The results from the bin's reconstruction are displayed in Figure 19 with the true shape shown in transparent green. The reconstructions are sliced to omit the thickest surface, highlighting the other more challenging walls and internal structure instead. Again, space carving is a poor choice when sensor noise increases.

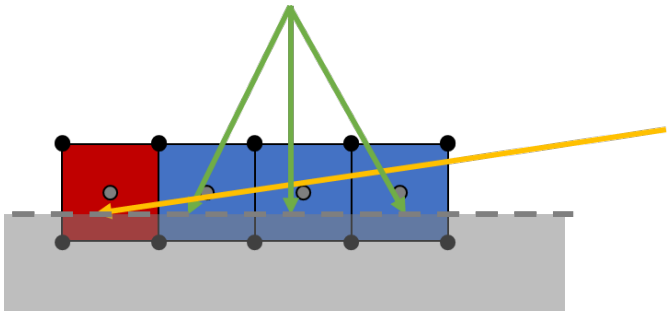


Figure 20 Surface erosion from large incident angle.

Even without noise the space carving method cannot represent the thin surface. As a view approaches orthogonality with a surface normal, it may erode voxels that should belong to the surface. A diagram of this is shown in Figure 20. The green measurements correctly identify the blue voxels as containing the surface, but the yellow measurement only sees the red one. As the update progresses back to the sensor location the blue voxels are updated to be free space.

This is an effect of the grid's discretization, and all ray-based updates suffer from this. The update processes of TSDF and occupation probability mitigate this slightly, but views aligned with a surface's normal are preferable.

The occupation probability grid appears to model the bin well under 2% noise. When this increases to 6%, it positively identifies all the occupied voxels but does so by grossly overestimating the bin. This might be acceptable for navigating to avoid this space but not to interact with it.

5.1.2 Resolution and Measurment Sparsity

When creating a voxelized reconstruction for scanning tasks, it is tempting to simply set the resolution as small as possible. This could make sense if the sensor used is extremely accurate. But measurement density also impacts the results.

The space carving, TSDF, and occupation probability methods only know information about a voxel if a ray passes through it. They all can make assumptions about unseen space, but this is not as reliable as actually measuring it. Additionally, the latter two methods

fundamentally rely on repeated updates of the same voxels so sparse sampling of surface points degrades their performance.

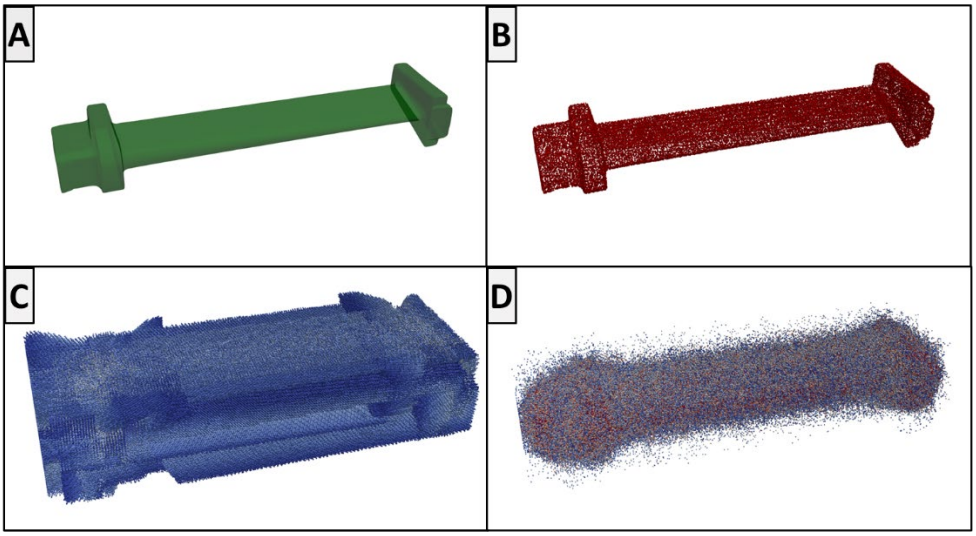


Figure 21 Fine-resolution reconstruction of a (A) rotor blade mesh: (B) surface voxels, (C) negative TSDF voxels, and (D) voxels with at least 80% occupation probability.

Figure 21 demonstrates this for a rotor blade part. From a view sphere at a distance of 0.9, 18 evenly spaced views are used. The real object is about 0.3 in length and a reconstruction is performed with a resolution set to 0.0005. It is reasonable that a well-calibrated, high-end sensor could achieve this accuracy. Even without noise, the measurements are sparse.

One could address this with adaptable voxel sizes, as [44] does with octrees for occupation probability. But not all methods are as easily adaptable to this format. Alternatively, erosion operations on the voxel grid could remove isolated, high-probability voxels could create a denser probability map at the same resolution.

A nearest-neighbor approach could resolve un-initialized voxels in a TSDF grid. But repeated updates are critical for estimating the surface. Using a coarse voxel resolution is

simpler than attempting to correct one that is too fine. Surface extraction algorithms, like marching cubes [42], interpolate the surface location between voxels, achieving a surface location resolution with greater precision than the voxel resolution.

Regardless of representation, voxel size and surface measurement density (a factor of sensor resolution and measurement standoff) should be considered in unison.

5.2 Policy Evaluation

Exploring simple geometries is useful for developing intuition about the basic nature of these representations. Unfortunately, real reconstruction tasks involve objects with unique features, surface curvatures, and occlusions. To evaluate how well these methods work on realistic parts, scanning tasks are performed on the Stanford bunny mesh.

The Stanford bunny, about 1 unit in all dimensions, is reconstructed in a grid of $[101,\ 101, 101]$ voxels with a resolution of 0.02. This reconstruction is performed at 2% and 6% noise.

5.2.1 Simple Policies

In scanning tasks, very simple policies like a view sphere or axial rotation are appealing since they are easy to implement and run quickly. While data-driven methods could propose a better view, they must process the reconstruction data to do so. If the scanning time and number of views are not constraints, then these methods could be reasonable.

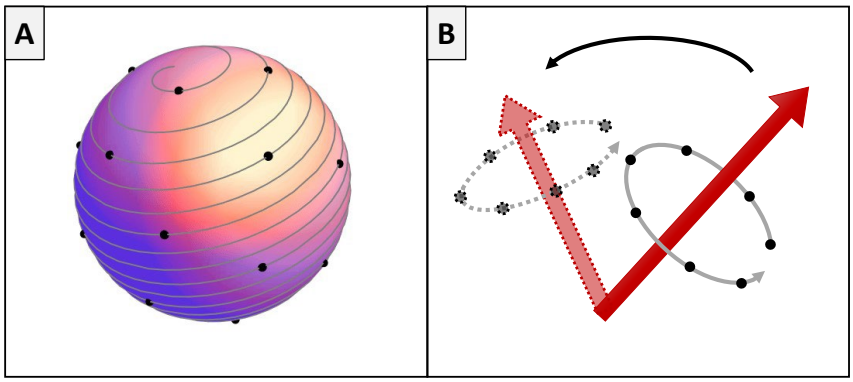


Figure 22 Simple policies based on (A) Fibonacci-spiral on a sphere and (B) evenly spaced views around a set of randomly selected axes.

Scans of the Stanford bunny are performed with three simple policies with a stop-condition of 6, 12, and 18 total views. The first policy, ordered sphere, follows a uniformly samples spherical points with a Fibonacci-spiral, seen in Figure 22A, visiting each view sequentially. The second, random sphere, uses the same spiral pattern but visits the locations in a random order. Figure 22B show the third method evenly spaces views around randomly selected axes. After visiting views around one axis, a new axis is selected, and the same number of views are sampled around it. In these experiments 6 views are sampled around an axis before it changes.

Reconstructions with the second and third policies are run 10 times. Each policy uses a radius of 2.5 and orients the views at the center of the reconstruction's bounding box. Only the TSDF and probability representations are evaluated.

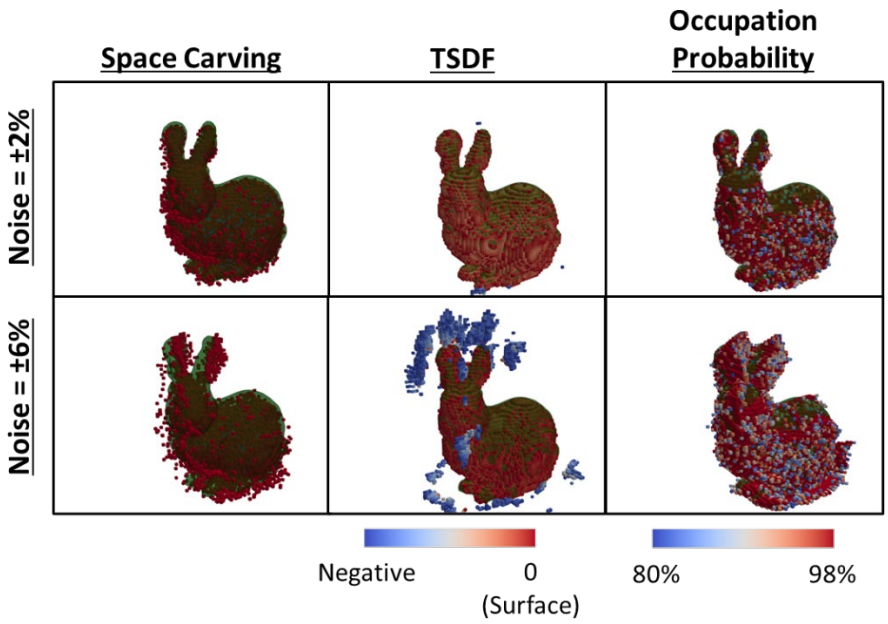


Figure 23 Reconstructions of the Stanford bunny following random spherical sampling.

A selection of the reconstructions after 18 views with the random spherical sampling policy are shown in Figure 23. The other policies generate visually similar reconstructions. Again, large negative artifacts are seen on the TSDF method as noise increases. These tend to be locations are around the near features, like the ears or neck, or edges, like the base, that cause spaces to be occluded from more perspectives than they are visible.

To evaluate both the reconstruction results and the policies which generated those results, the balanced-accuracy and precision are recorded after each additional view. These metrics are described in Section 4.2.

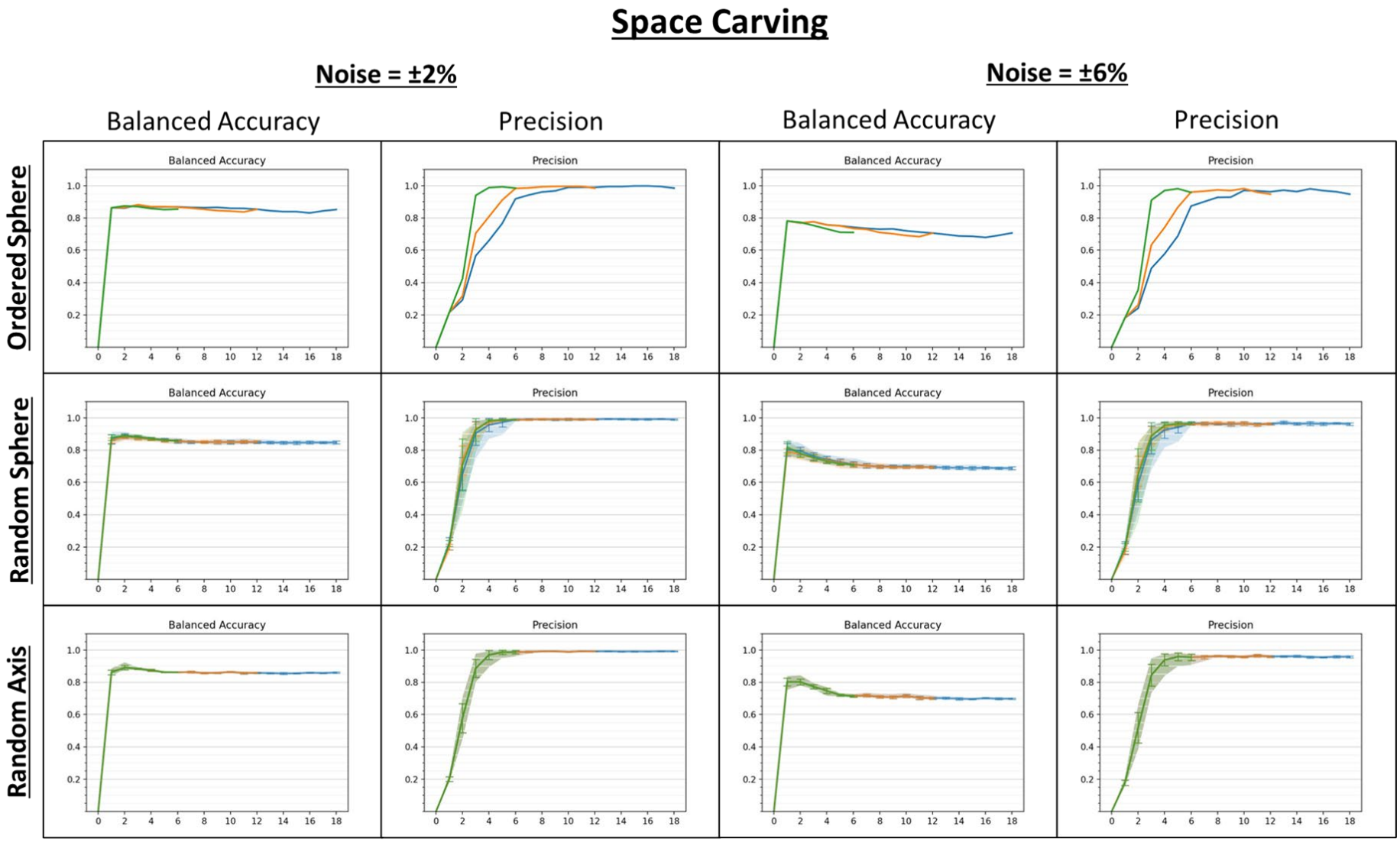


Figure 24 Evaluation of the space carving representation as views are added while following different policies.

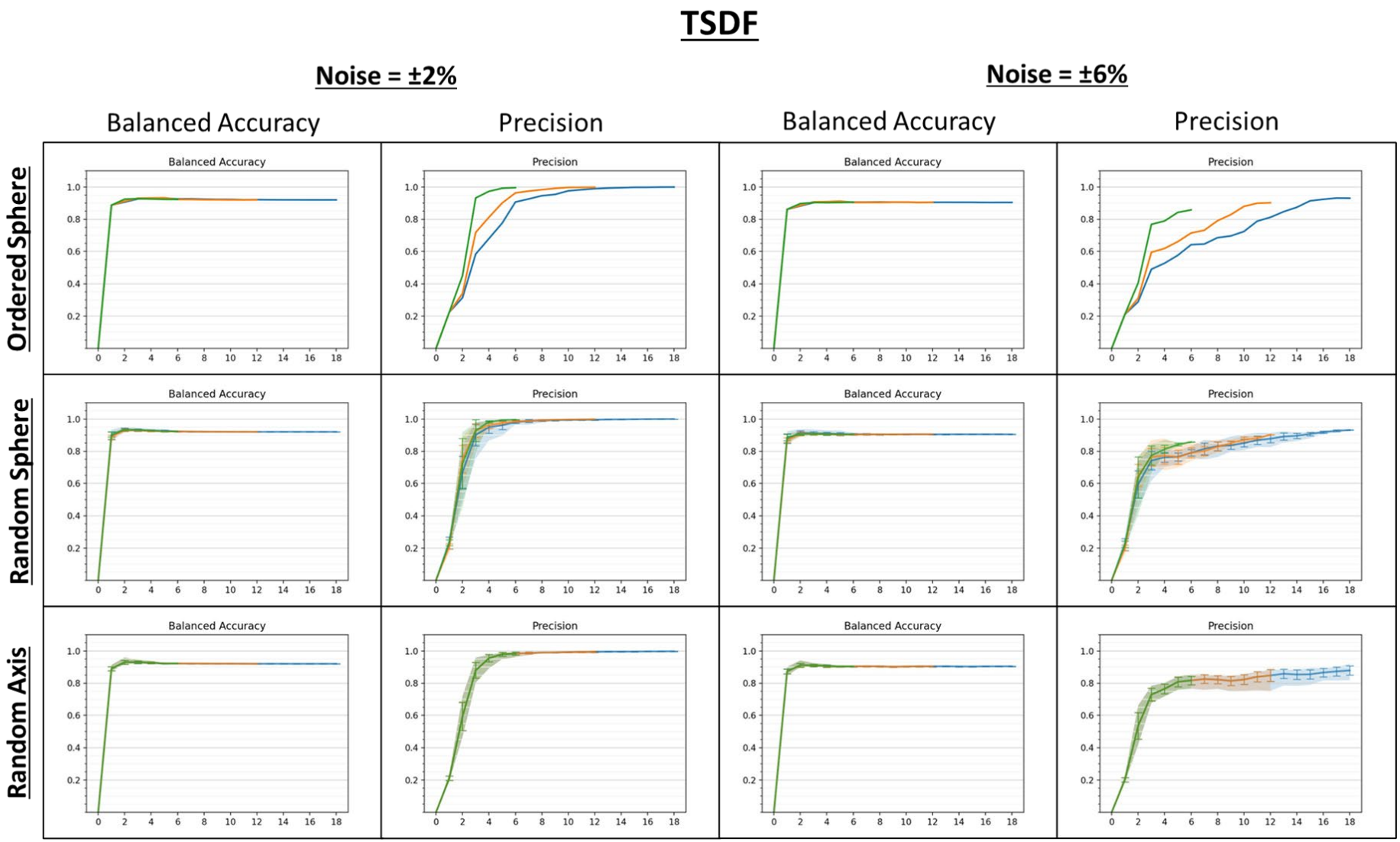


Figure 25 Evaluation of the TSDF representation as views are added while following different policies.

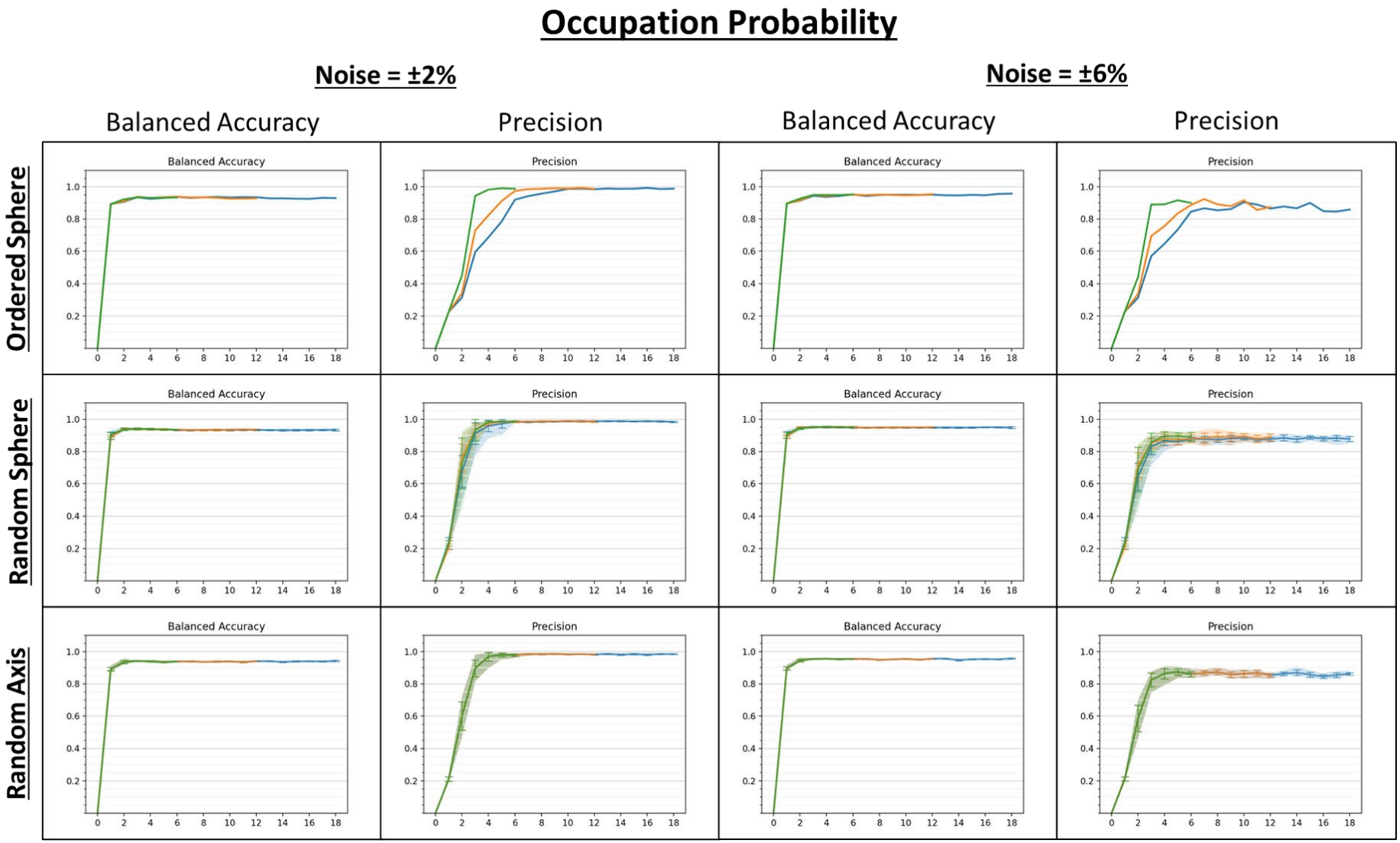


Figure 26 Evaluation of the occupation probability representation as views are added while following different policies.

The results of these experiments are divided by voxel representation into space carving, TSDF, and occupation probability in Figure 24, Figure 25, and Figure 26, respectively.

Across all methods accuracy converges quickly to its final value. Typically, after three views are added. At 2% noise, balanced accuracy approximately converges to 85% for space carving, 90% for TSDF, and 95% for space carving. At 6% noise this decreases to nearly 70% for space carving. The accuracy of TSDF and occupancy probability representations remains the same as the noise increases, demonstrating a robustness to uncertainty.

Balanced accuracy is the average of the sensitivity and specificity of the representation at a given time. With space carving, this value may decrease as measurements are added. The initial assumption that all voxels are unseen, thus assumed to be occupied, means each voxel is either true positive or false positive. However, the carving process can only move voxels to the free category, meaning that after enough measurements all free space will be correctly identified but the surface location will be underestimated. This decrease in balanced accuracy reflects the errors shifting from false positives to false negatives – sensitivity begins high and decreases while selectivity begins low and increases.

Precision describes the correctness of the voxels predicted to be occupied by a representation. Under 2% noise, all representations can reach near 99% precision. This reflects the elimination of false-positive voxels – an overestimation of the object's volume. Each representation archives this differently and the view selection policy affects its rate.

Space carving achieves a high precision – even at 6% noise – because it can only eliminate occupied voxels. Any false positives can only become true negatives. And any true positives can only become false negatives. This is related to the decrease in accuracy described in the paragraphs above.

The TSDF representation displays an interesting precision characteristic at 6% noise where it increases as views are added rather than converging. This is the result of negative artifacts present around the object. These false-positives, seen in Figure 23, are located behind thin features and edges. Only a small number of views will see these regions from a perspective where these voxels will be within the positive truncation region and updated. Sampling more views make it more likely that this will be eliminated, slowly increasing the precision.

It is likely that the precision of this representation would improve if these artifacts were filtered out before calculation. One could save only the largest connected region in the reconstruction. Other representations could provide heuristics to overwrite false positive voxels. For example, a voxel with a very negative value in the TSDF but low occupation probability overruled and marked as free.

The precision of the occupancy probability at 6% noise reaches anywhere between 85% and 90%. A high precision and accuracy mean this representation is robust to uncertainty.

The rate of precision convergence is useful for comparing the performance of different policies. Across all three representations, all three policies reach their maximum precision value by three to six views (except for TSDF at 6% noise, discussed above).

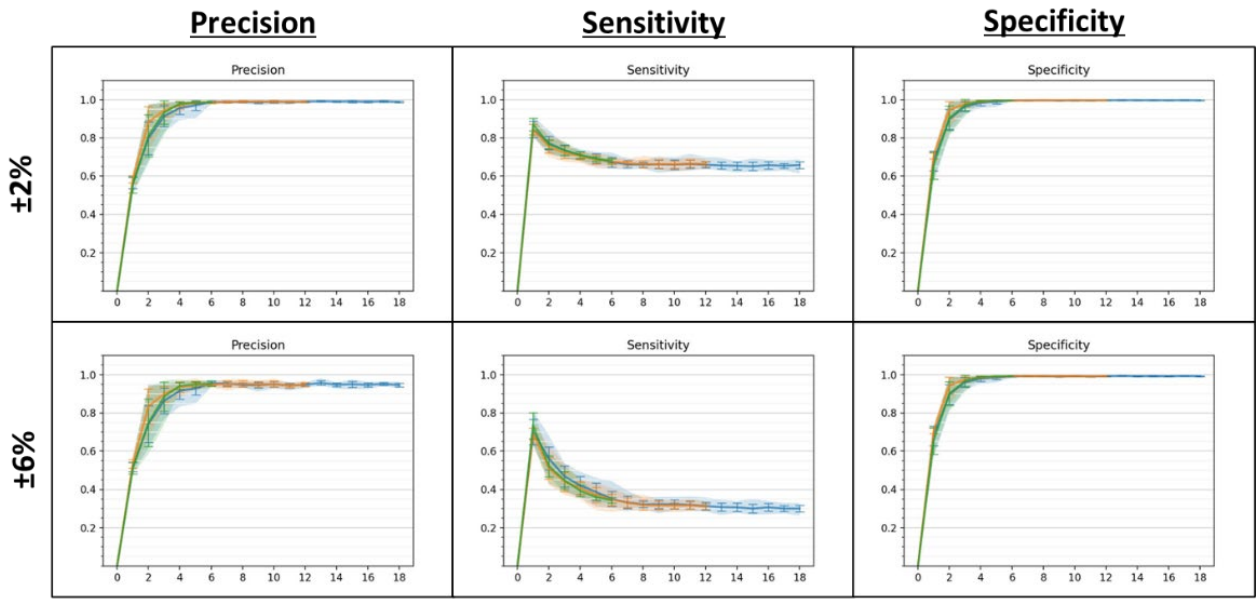
The ordered sphere policy converges quickest when sampling only six views and the convergence rate decreases as the number of views grows. As the number of requested views increases so does the overlap between each sequential view. At 18 views, more time is spent re-measuring previously seen surfaces than exploring the part. This leads to an intuition that rapid exploration is preferable. The only case where rescanning surfaces from new perspectives is useful is the TSDF representation at high levels of noise.

The random sphere policy does not experience this relationship between convergence rate and number of views as the ordered version. Randomly visiting each location reduces the chance that views will overlap, especially in the first few views. It is not impossible that views will have high overlap, but at worst this policy will perform as well as the ordered sphere.

If a system is constrained to just a few views, then a deterministic policy, ordered spherical, is advantageous to guarantee that views are well spaced. But random, low-discrepancy sampling performs better and faster as more views are allowed.

The random axis policy works well and is consistent, even when just one axis is used. It is subject to less variance than the random sphere method. Views are guaranteed to be well-spaced while the changing axis allows random exploration.

Evaluating policies under system constraints, like reachability or movement time, is not a focus of this work. But it is worth mentioning that this may be important to consider. If the object is held by a robot in front of a fixed camera, then a policy like random axis could be preferable for its low actuation cost between views – usually just rotating the wrist joint.



The comparisons above consider every voxel in the reconstruction area, and each contributes an equal vote when constructing the confusion matrix. Of the nearly one-million voxels in the reconstruction only 41-thousand contain the Stanford bunny. Intuitively, it is more important to correctly identify voxels near the surface if a system needs to interact with an object. Each representation is equally good at categorizing the free voxels, what occurs near the surface is more interesting.

Figure 27 Truncated comparison of space carving from the random sphere policy.

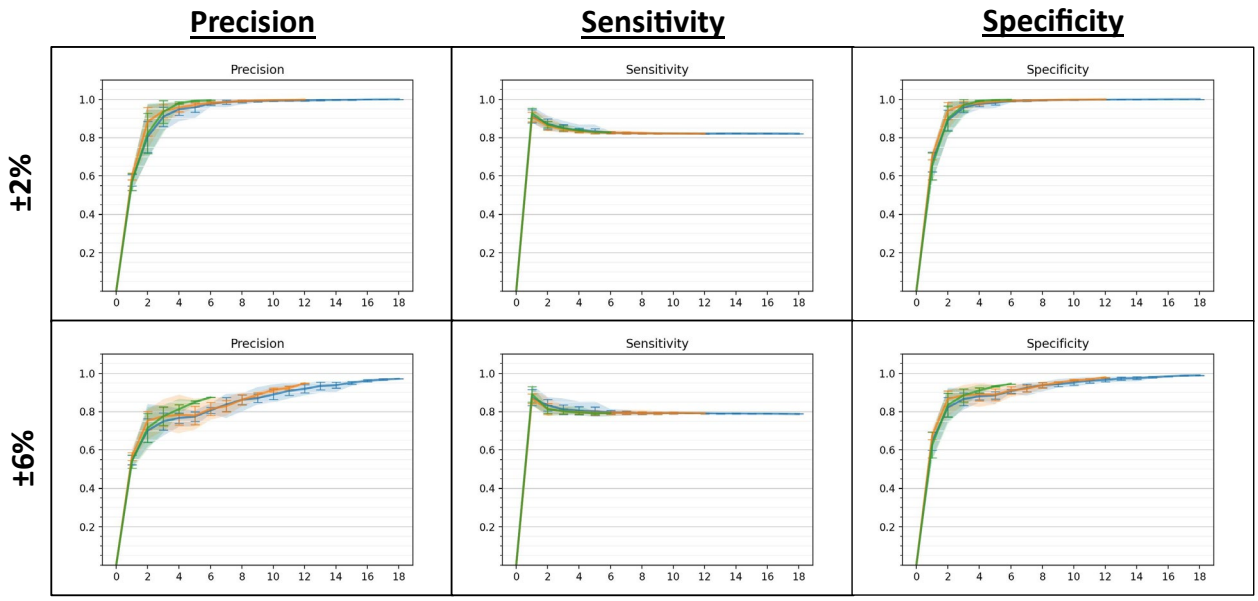


Figure 28 Truncated comparison of TSDF from the random sphere policy.

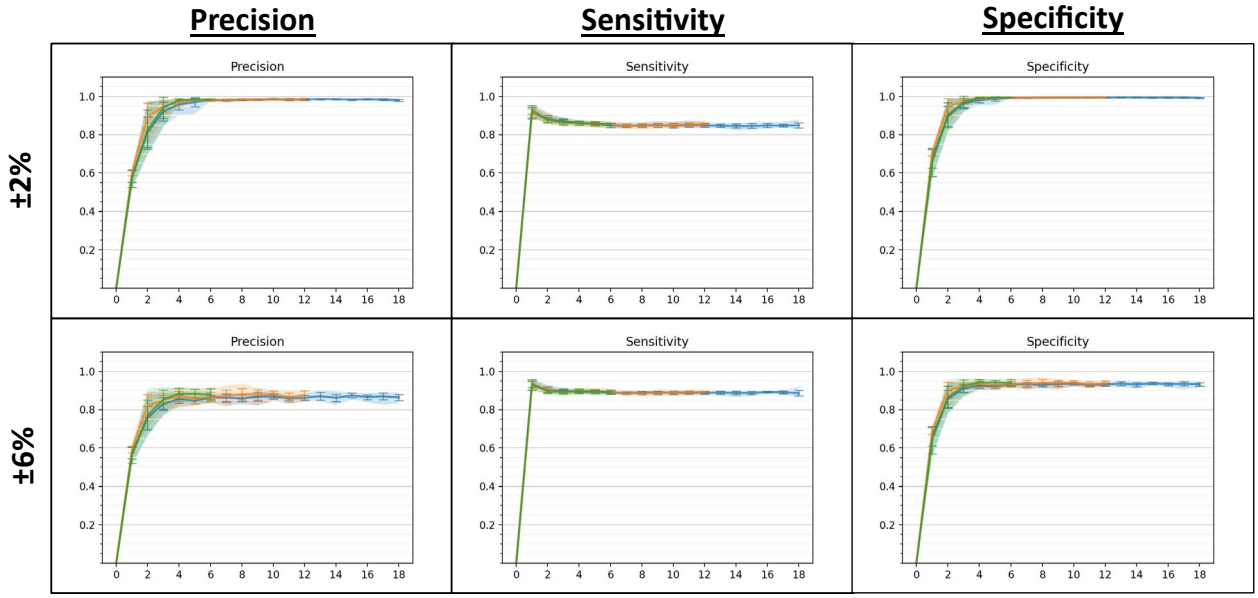


Figure 29 Truncated comparison of occupation probability using a random sphere policy.

Figures 27, 28, and 29 evaluate each representation only for voxels ± 0.15 of the true surface. Balanced accuracy is divided into its constituent values: sensitivity, correctly identify occupied voxels, and selectivity, correctly identify free voxels.

For space carving the seperation of sensitivity and specificity highlights how unreliable its negative predictions are. It is prone to severe underestimation as noise increases, resulting in many false negative labels.

Using TSDF sensitivity is similar at both noise levels. But with higher noise more images improve the precision and specificity. This shows that the representation errors are driven by overestimation of the surface. And addition measurements of the same voxels from different perspectives improves its estimation of the surface.

With the occupation probability representation, the final values and number of view for convergence are similar between the full comparison in Figure 26 and the truncated on in Figure 29. There is slightly greater variance in the precision and balanced-accuracy (the average of sensitivity and specificity) when looking only at the truncated region.

The performance of a policy is also dependent on the scanned object. The Stanford bunny has many qualities that make it relatively easy to measure. Its large, smooth features mean each image contains informative measurements and the few occluded areas are viewable from as many views as they are hidden. Achieving a high precision by five to six views is impressive but only possible because this geometry is simple. For more objects containing more complex features, these scanning strategies could easily miss important details.

5.2.2 Occplane Policy

The combination of geometries, reconstruction parameters, and sensor intrinsics are used to test the occplane policy described in section 4.3. This policy is intended to generate a diverse set of views that maximize early exploration and identify occluded features.

This policy requires at least one initial view to add information to a space carving grid searching for occplanes. This is randomly selected from a view sphere at a radius of 2.5.

In each run the policy saves the top three views and executes them in order. This is repeated three times for a total of ten views – one random and nine from the policy.

Ten repetitions are performed with sensor noise set to 0%, 2%, and 6%.

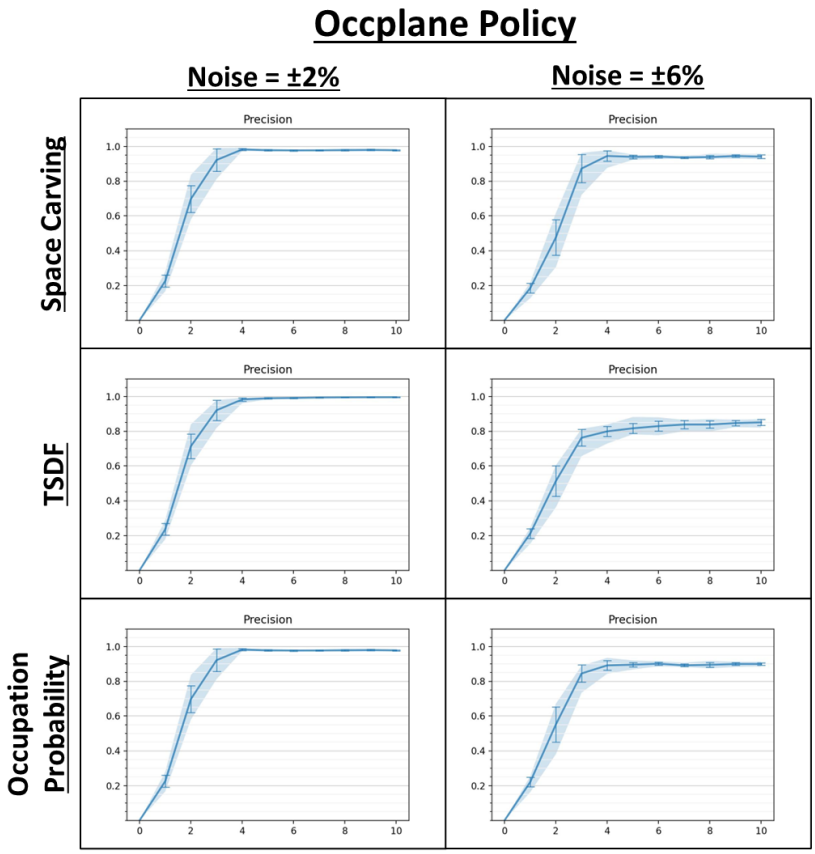


Figure 30 Precision of each voxel representation as views are added following the occplane policy.

The resulting precision and miss-rate are displayed in Figure 30. The TSDF and occupancy probability representations display similar characteristics, so only the latter is shown.

The precision tends to converge to the same values as the simple policies at all noise levels. Precision converges by three to five views, a slight improvement over the simpler policies.

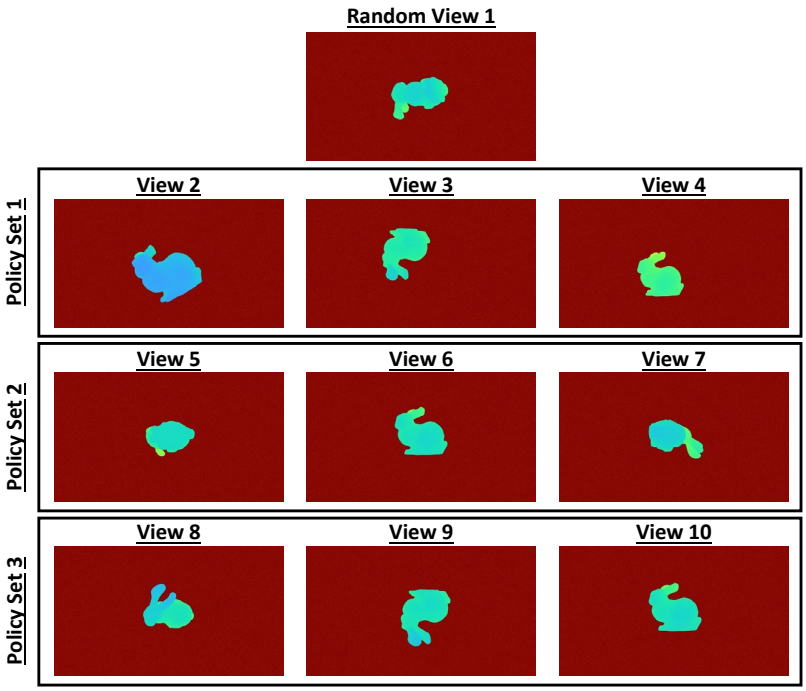


Figure 31 Sets of views from the Occplane policy views in one reconstruction run.

The lower miss-rate is a result of re-sampling locations. Figure 31 shows the views generated during one reconstruction with the occplane policy. A diverse set of views is created but repetition occurs across runs; views 4, 6, and 10 are nearly identical.

Under repeated measurements, TSDF improves the miss-rate by calculating the average surface location while occupation probability achieves this by expanding the volume it predicts to be occupied.

While this repetition reduces the miss-rate, it highlights one limitation of this strategy. When occplane voxels are clustered to propose a view, the policy assumes that view will see these voxels. But that is not always true.

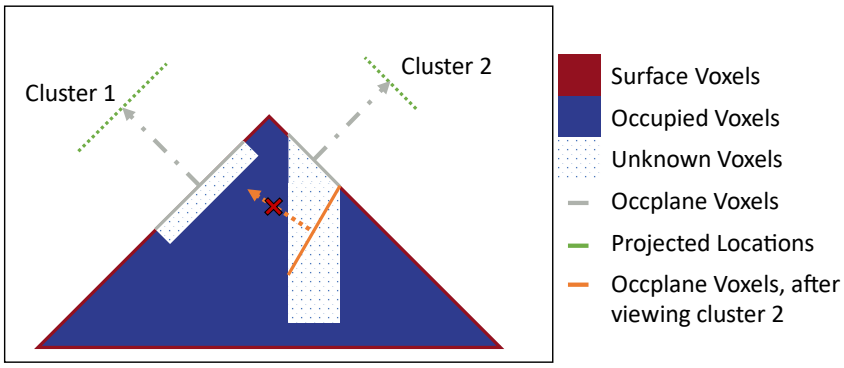


Figure 32 2D visualization of the occplane policy limitations.

Figure 32 visually describes how this policy might fail to image occluded voxels. The policy requires at least one view to generate an initial representation which contains occplanes. In this example the policy begins with the occupied, unknown, and free voxels. It locates the occplane voxels and their surface normals then projects this into two distinct clusters with a suggested view.

Cluster 1 contains more points and is viewed first. The policy knows nothing about what is behind the occplane, but this happens to be a good perspective to measure from. All unknown voxels are categorized as occupied or free. This is the best-case scenario.

Next, the policy selects cluster 2's view. The measurement sees many new voxels, but it is sub-optimal. The policy cannot infer from the occplane location that the best view of this feature would be from the top of the image down.

The true limitation arises when the next occplanes are generated. Only one surface of occplane voxels remains after viewing these two locations. Projecting and clustering this surface will result in a view that would either be inside the part or unable to see the surface.

This can be a good algorithm to begin a scan with before switching to another strategy. It can gather a few well distributed initial views that another strategy can then continue from. But it is not designed to search for fine-details or view self-occluded features on the part. It could be expanded to consider the set of surface points in combination with the occplane surface normals to mitigate view occlusion.

5.2.3 Precomputed Views

The algorithms described in Section 4.4 outline a way to precompute views based on an assumed geometry. This prior knowledge is not guaranteed for all tasks, but when it is available it is extremely useful. A reasonable number of tasks, especially in manufacturing or other situations where the same scan is executed repeatedly, can benefit from precomputed views.

To demonstrate this, the Stanford bunny is reconstructed using a box, a convex hull of its mesh, and its actual mesh as the approximate geometry. Ten views are pre-computed from each source and then reconstructed with 2% sensor noise.

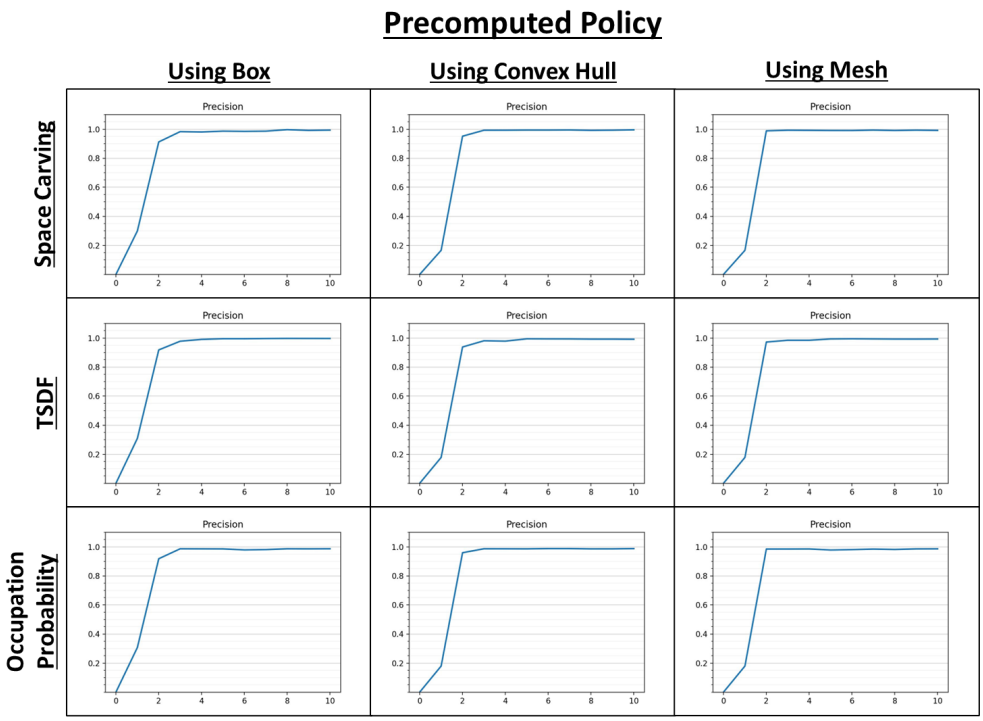


Figure 33 Precision of each voxel representation as views are added following the precomputed views policy.

Using prior knowledge, this policy converges even faster than all other policies. With just three views its precision converges. This hull and mesh lead to a slightly better order of views than the box. But all perform well.

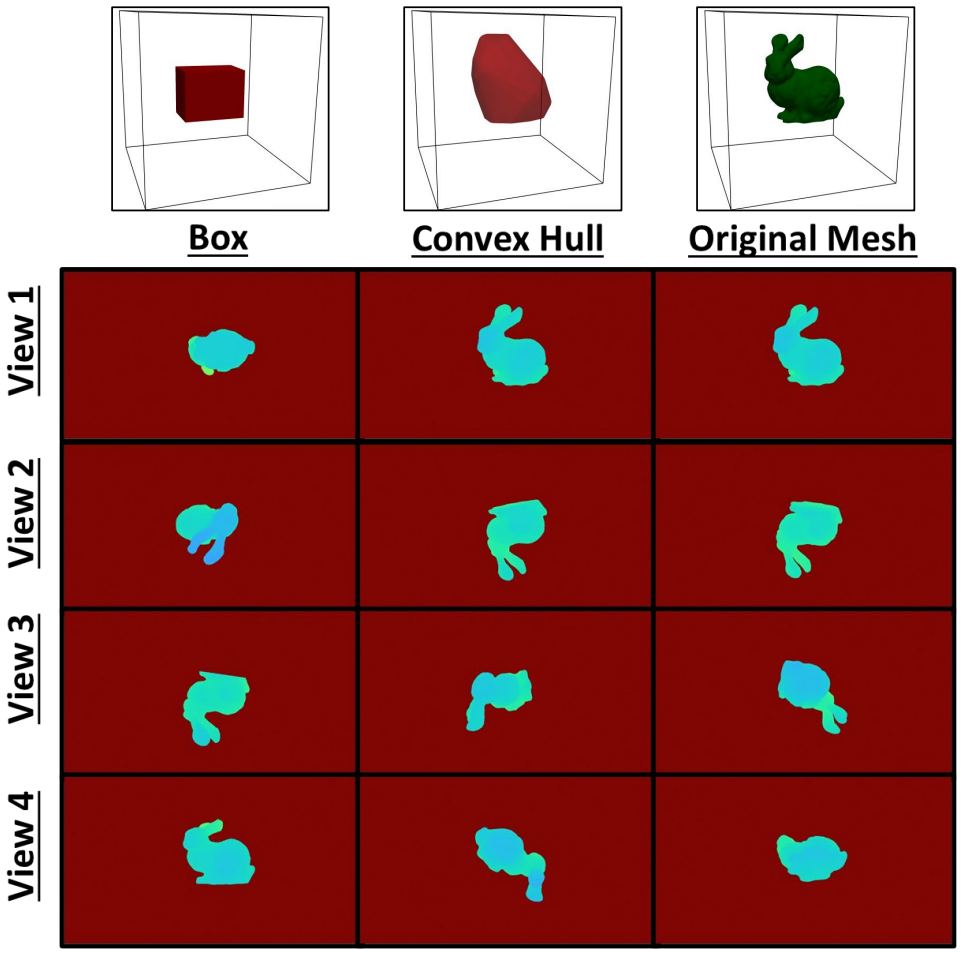


Figure 34 First four views from each precomputed set.

Exploring the first four views, shown in Figure 34, reveals how each assumed geometry generates diverse, informative views. In each case, the policy identified two sets of opposing views.

The box geometry has different dimensions than the Bunny and its top face has the largest surface area. The policy identifies the bottom and top of the box as the most informative views, followed by the front and back.

With the convex hull and original mesh, the policy identifies that viewing the front and back first will see more of the object. Selecting these before the top and bottom allows it to converge slightly faster.

Whenever possible, utilizing known information can accelerate a scanning task. The model does not need to be perfect to provide good intuition for explorative scanning. Surface normals provide a simple heuristic that all meshes record. But other algorithms could search the mesh for edges, holes, or user-defined locations to improve the scanning around specific features.

Chapter 6. Conclusions and Future Work

The preceding chapters of this thesis discuss reconstruction tasks in detail. Mapping and scanning tasks consist of three interdependent components: data representation, view selection algorithms, and depth sensors. Ensuring reliable and correct reconstruction results requires understanding these parts independently and holistically.

This work focuses on view selection policies and data representations. Depth sensors are important, but they are often dictated by design constraints like weight, size, and cost. Instead, different algorithms and data structures are readily implementable.

The selection of these two elements and what parameters are used can affect the reconstruction's model as much as the sensor choice. Proper tuning can improve reconstructions done with a less accurate, but cheaper, sensor while improper tuning can diminish performance with a more accurate, and expensive, sensor.

Voxels are certainly not the only data structure for reconstructions, but their convenient and explicit volumetric format makes them a common choice for many works. In reconstruction tasks, voxel grids are characterized by the value each element stores and the update rule it follows when integrating new data. Space carving, TSDF, and occupation probability are incredibly popular methods, seen repeatedly in literature.

Algorithms for view selection are varied. Some are designed primarily for mapping tasks with a focus on exploration while others are intended to capture highly accurate models in scanning tasks. A division can be made between simple next view algorithms which follow some geometric pattern or other explicit rules and those that utilize the data representation for an informative next-best view.

Intuition and experience may guide the selection of sensors, data structure, and view selection algorithms. But it cannot quantitatively compare their performance. Would a smaller voxel resolution improve accuracy? Can a better view selection algorithm and data representation compensate for a less-accurate sensor? Answering questions like these requires implementing, testing, and comparing all three parts.

To meet this need, ForgeScan, an open-source library for autonomous reconstruction tasks, is introduced. It considers next-best view selection algorithms and voxelized data representations as modular components of the task. With simulated depth images, different combinations of algorithms and data structure may be rapidly prototyped and evaluated on test geometries.

6.1 Future Work

Autonomous reconstruction research exists at the intersection of many fields. Concepts from computational geometry, machine learning, information theory, computer vision, topology, and many more fields are routinely incorporated into reconstruction works. This thesis is just one step towards a comprehensive understanding of reconstruction tasks. As

in any field worth researching, the knowledge gained here exposes new questions and the need for further experiments.

This work focuses solely on voxels because of their frequent use in the field. But several interesting next-best view selection algorithms utilize meshes or point clouds instead of or in addition to voxels. Extending ForgeScan to include point clouds, octrees, and other data representations would be advantageous to compare more unique algorithms.

For voxel grids, a confusion matrix provides a convenient way to compare different representations to a common truth. Precision and miss-rate are useful to describe how well a representation models a scene while a policy is run.

These metrics describe the quantity of correct predictions. But they do not describe how incorrect a prediction is. For example, the TSDF and occupation probability reconstructions in Figure 23 have similar precision and miss-rates although the locations of these errors clearly differ. Additional analysis methods to describe the size and locations of incorrectly-predicted voxels are needed.

This does suggest another interesting line of research: combining information between methods. Either during the reconstruction or once all data is collected, the predicted occupancy could inform the weights of TSDF updates or filter erroneous regions.

The inter-method comparison of the space carving, TSDF, and occupancy probability representations by generalizing each to a set of free and occupied voxels. But each is suited for slightly different goals. Intra-method comparisons should be considered in future work.

For example, extracting the surface from the TSDF grid and comparing it to the original model. Or a study of ray probability profiles for occupation probability methods.

Additionally, the sensor's measurement error is modeled. But real systems also experience uncertainty in sensor placement. This induces a systemic error to each image's measurements which could be modeled in a general way.

Parallelizing certain parts of the library to run on GPUs would be advantageous. Presently, the library is written only for CPU. It is reasonably fast for the prototyping and experimenting done in this work; depending on the hardware integrating a 1920x720 depth image takes about a second. Optimizations, like GPU parallelization, could allow the library to be used in real systems without becoming a bottleneck.

6.2 Closing Thoughts

Every day people process complex decisions, interact effortlessly with objects, and navigate new, changing environments. The ease with which we observe and interpret our environments is impressive. And, for autonomous systems, enviable.

Scanning and mapping are foundational for so many decisions in autonomous systems. Advancements in robotics, embedded computation, and imaging have not made autonomous reconstruction tasks easy. They have made them possible.

The adoption of these systems is evident: fleets of robots navigate logistics warehouses and mobile platforms perform routine inspections of industrial equipment. Slowly, these systems are entering the public sphere too.

Despite the amount of research and effort in this area, there are no general solutions to ensure coverage, accuracy, and speed in all cases. It is possible to derive some rules and best-practices. But these cannot replace a firm understanding of the system's design and the task at hand.

Bibliography

[1] R. Border, J. D. Gammell, and P. Newman, "Surface Edge Explorer (see): Planning Next Best Views Directly from 3D Observations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD: IEEE, May 2018, pp. 1–8. doi: 10.1109/ICRA.2018.8461098.

[2] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Basel: IEEE, Oct. 2011, pp. 127–136. doi: 10.1109/ISMAR.2011.6092378.

[3] Y. Han, I. H. Zhan, W. Zhao, and Y.-J. Liu, "A Double Branch Next-Best-View Network and Novel Robot System for Active Object Reconstruction," in *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA: IEEE, May 2022, pp. 7306–7312. doi: 10.1109/ICRA46639.2022.9811769.

[4] C. Mineo, D. Cerniglia, V. Ricotta, and B. Reitinger, "Autonomous 3D geometry reconstruction through robot-manipulated optical sensors," *Int J Adv Manuf Technol*, vol. 116, no. 5–6, pp. 1895–1911, Sep. 2021, doi: 10.1007/s00170-021-07432-5.

[5] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3D object modeling," *The International Journal of Robotics Research*, vol. 30, no. 11, pp. 1311–1327, Sep. 2011, doi: 10.1177/0278364911403178.

[6] S. Kriegel, C. Rink, T. Bodenmuller, A. Narr, M. Suppa, and G. Hirzinger, "Next-best-scan planning for autonomous 3D modeling," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 2850–2856. doi: 10.1109/IROS.2012.6385624.

[7] C. Potthast and G. S. Sukhatme, "A probabilistic framework for next best view estimation in a cluttered environment," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 148–164, Jan. 2014, doi: 10.1016/j.jvcir.2013.07.006.

[8] J. Cui, J. T. Wen, and J. Trinkle, "A Multi-Sensor Next-Best-View Framework for Geometric Model-Based Robotics Applications," in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada: IEEE, May 2019, pp. 8769–8775. doi: 10.1109/ICRA.2019.8794423.

[9] A. Schellenberg, "ForgeScan." Nov. 2023. Accessed: Nov. 15, 2023. [Online]. Available: https://github.com/Schellenberg3/ForgeScan

[10] L. E. Ortiz, V. E. Cabrera, and L. M. G. Goncalves, "Depth Data Error Modeling of the ZED 3D Vision Sensor from Stereolabs," *ELCVIA*, vol. 17, no. 1, p. 1, Jun. 2018, doi: 10.5565/rev/elcvia.1084.

[11] "Intel® RealSenseTM Product Family D400 Series Datasheet," Intel, Product Revision 017, Sep. 2023. Accessed: Oct. 24, 2023. [Online]. Available: https://www.intelrealsense.com/download/21345/?tmstv=1697035582
[12] A. Grunnet-Jepsen, J. Sweetser, Winer, A. Takagi, and J. Woodfill, "Projectors for Intel® RealSense™ Depth Cameras D4xx," Intel, Whitepaper. Accessed: Oct. 25, 2023. [Online]. Available: https://dev.intelrealsense.com/docs/projectors
[13] P. Rachakonda, B. Muralikrishnan, and D. Sawyer, "Sources of Errors in Structured Light 3D Scanners," SPIE Defense and Commercial Sensing 2019, Baltimore, MD, US, Apr. 2019. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927473
[14] Y. He, B. Liang, Y. Zou, J. He, and J. Yang, "Depth Errors Analysis and Correction for Time-of-Flight (ToF) Cameras," *Sensors*, vol. 17, no. 1, p. 92, Jan. 2017, doi: 10.3390/s17010092.
[15] "MultiSense S30 Spec Sheet," Carnegie Robotics. Accessed: Oct. 23, 2023. [Online]. Available: https://assets-global.website-files.com/63d32fab754bc97c27b98158/63d32fab754bc956c8b98323_Spec%20Sheet%20-%20MultiSense%20S30%20(2).pdf
[16] A. Grunnet-Jepsen, J. Sweetser, and J. Woodfill, "Best-Known-Methods for Tuning Intel® RealSense™ Depth Cameras D400 series for Best Performance," Intel, Whitepaper. Accessed: Oct. 23, 2023. [Online]. Available: https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance
[17] "Intel® RealSense™ Camera Depth Testing Methodology," Intel, Nov. 2017. Accessed: Oct. 25, 2023. [Online]. Available: https://dev.intelrealsense.com/docs/camera-depth-testing-methodology
[18] O. Wasenmüller and D. Stricker, "Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision," in *Computer Vision – ACCV 2016 Workshops*, vol. 10117, C.-S. Chen, J. Lu, and K.-K. Ma, Eds., in Lecture Notes in Computer Science, vol. 10117. , Cham: Springer International Publishing, 2017, pp. 34–45. doi: 10.1007/978-3-319-54427-4_3.
[19] S. Lee, "Depth camera image processing and applications," in *2012 19th IEEE International Conference on Image Processing*, Orlando, FL, USA: IEEE, Sep. 2012, pp. 545–548. doi: 10.1109/ICIP.2012.6466917.
[20] Makerbot, "Stanford Bunny.stl." Jan. 23, 2020. [Online]. Available: https://thingiverse.com/thing:88208/files
[21] "3D Viewing: the Pinhole Camera Model," Scratchapixel. Accessed: Nov. 02, 2023. [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/implementing-virtual-pinhole-camera.html
[22] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, Not Known: ACM Press, 1996, pp. 303–312. doi: 10.1145/237170.237269.
[23] T. Barnes, "Fast, Branchless Ray/Bounding Box Intersections, Part 3: Boundaries." Jul. 27, 2022. Accessed: Oct. 23, 2023. [Online]. Available: https://tavianator.com/2022/ray_box_boundary.html

[24] A. Majercik, C. Crassin, P. Shirley, and M. McGuire, “A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering,” *Journal of Computer Graphics Techniques (JCGT)*, Sep. 2018, [Online]. Available: https://jcgt.org/published/0007/03/04/
[25] J. Amanatides and A. Woo, “A Fast Voxel Traversal Algorithm for Ray Tracing,” *Eurographics*, pp. 3–10, Aug. 1987.
[26] K. A. Tarabanis, P. K. Allen, and R. Y. Tsai, “A survey of sensor planning in computer vision,” *IEEE Trans. Robot. Automat.*, vol. 11, no. 1, pp. 86–104, Feb. 1995, doi: 10.1109/70.345940.
[27] L. M. Wong, C. Dumont, and M. A. Abidi, “Next best view system in a 3D object modeling task,” in *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA’99 (Cat. No.99EX375)*, Monterey, CA, USA: IEEE, 1999, pp. 306–311. doi: 10.1109/CIRA.1999.810066.
[28] M. Krainin, B. Curless, and D. Fox, “Autonomous generation of complete 3D object models using next best view manipulation planning,” in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 5031–5037. doi: 10.1109/ICRA.2011.5980429.
[29] J. I. Vasquez-Gomez, L. E. Sucar, and R. Murrieta-Cid, “Hierarchical Ray Tracing for Fast Volumetric Next-Best-View Planning,” in *2013 International Conference on Computer and Robot Vision*, Regina, SK, Canada: IEEE, May 2013, pp. 181–187. doi: 10.1109/CRV.2013.42.
[30] J. Yang, J. Rebello, and S. L. Waslander, “Next-Best-View Selection for Robot Eye-in-Hand Calibration.” arXiv, Mar. 12, 2023. Accessed: Jul. 05, 2023. [Online]. Available: http://arxiv.org/abs/2303.06766
[31] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, “Receding Horizon ‘Next-Best-View’ Planner for 3D Exploration,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden: IEEE, May 2016, pp. 1462–1468. doi: 10.1109/ICRA.2016.7487281.
[32] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, “Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments,” *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1699–1706, Apr. 2019, doi: 10.1109/LRA.2019.2897343.
[33] R. Almadhoun, A. Abduldayem, T. Taha, L. Seneviratne, and Y. Zweiri, “Guided Next Best View for 3D Reconstruction of Large Complex Structures,” *Remote Sensing*, vol. 11, no. 20, p. 2440, Oct. 2019, doi: 10.3390/rs11202440.
[34] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, “Automatic View Selection Using Viewpoint Entropy and its Application to Image-Based Modelling: Automatic View Selection,” *Computer Graphics Forum*, vol. 22, no. 4, pp. 689–700, Dec. 2003, doi: 10.1111/j.1467-8659.2003.00717.x.
[35] U. D. Bordoloi and Han-Wei Shen, “View Selection for Volume Rendering,” in *VIS 05. IEEE Visualization, 2005.*, Minneapolis, MN, USA: IEEE, 2005, pp. 487–494. doi: 10.1109/VISUAL.2005.1532833.
[36] M. T. Lozano Albalate, M. Devy, J. Miguel, and S. Marti, “Perception planning for an exploration task of a 3D environment,” in *Object recognition supported by*

user interaction for service robots, Quebec City, Que., Canada: IEEE Comput. Soc, 2002, pp. 704–707. doi: 10.1109/ICPR.2002.1048036.

[37] J. I. Vásquez and L. E. Sucar, "Next-Best-View Planning for 3D Object Reconstruction under Positioning Error," in *Advances in Artificial Intelligence*, vol. 7094, I. Batyrshin and G. Sidorov, Eds., in Lecture Notes in Computer Science, vol. 7094. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 429–442. doi: 10.1007/978-3-642-25324-9_37.

[38] B. Curless and M. Levoy, "New methods for surface reconstruction from range images," PhD, Stanford University, 1997.

[39] D. Werner, A. Al-Hamadi, and P. Werner, "Truncated Signed Distance Function: Experiments on Voxel Size," in *Image Analysis and Recognition*, vol. 8815, A. Campilho and M. Kamel, Eds., in Lecture Notes in Computer Science, vol. 8815. , Cham: Springer International Publishing, 2014, pp. 357–364. doi: 10.1007/978-3-319-11755-3_40.

[40] D. R. Canelhas, *Truncated signed distance fields applied to robotics*. Örebro: Örebro University, 2017.

[41] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed Distance Fields: A Natural Representation for Both Mapping and Planning," p. 6 p., 2016, doi: 10.3929/ETHZ-A-010820134.

[42] W. E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, in SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. doi: 10.1145/37401.37422.

[43] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, St. Louis, MO, USA: Institute of Electrical and Electronics Engineers, 1985, pp. 116–121. doi: 10.1109/ROBOT.1985.1087316.

[44] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Auton Robot*, vol. 34, no. 3, pp. 189–206, Apr. 2013, doi: 10.1007/s10514-012-9321-0.

[45] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing." arXiv, Jan. 29, 2018. Accessed: Sep. 28, 2022. [Online]. Available: http://arxiv.org/abs/1801.09847

[46] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, in KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.