SIEMENS

IGMR Institute of Mechanism Theory, Machine Dynamics and Robotics | RWTH AACHEN UNIVERSITY

This thesis was submitted to the Institute of Mechanism Theory, Machine Dynamics and Robotics

Operator Guidance System for Visual Inspections - Dynamic Trajectory Estimation for Complete Coverage of Inspection Items

Master Thesis

by:

Sawan Saxena M.Sc.

Student number: 394823

supervised by:

Markus Schmitz M.Sc.,
IGMR RWTH Aachen University

Guido Schröer,
Siemens Mobility GmbH

Examiner:

Univ.-Prof. Dr.-Ing. Dr. h. c. Burkhard Corves

Prof. Dr.-Ing. Mathias Hüsing

Aachen, 3 April 2023

Master Thesis

by Sawan Saxena M.Sc.
Student number: 394823

Operator Guidance System for Visual Inspections - Dynamic Trajectory Estimation for Complete Coverage of Inspection Items

---

The use of drones for asset and item inspection is gaining popularity as it offers significant benefits such as time-saving and increased efficiency. The majority of the inspection work is still performed manually. The control is entirely manual, even if the drone is used for reaching unreachable locations. With drones, quality inspectors can obtain a comprehensive overview of assets without being physically present at the inspection site. A special focus lies on ensuring consistent quality during the inspection process. Although a human inspector takes a lot of experience, repetitive tasks are prone to errors, and a human may miss certain parts of the inspection. Hence there is a need to reduce human error as much as possible.
Siemens Mobility recognizes the need to automate the inspection process of trains. By using drones, the inspection process can be made faster and more efficient. One critical aspect of automating drone operations is planning the trajectory to cover all necessary regions of the asset. It is essential to ensure that all optimal points are identified and visited and that the shortest possible trajectory is taken.
This thesis primarily focuses on designing algorithms to compute the essential viewpoints required to fully cover the region of interest. The main challenge is identifying all necessary viewpoints that ensure full coverage of the train's region of interest during the inspection. The developed algorithm is tested on train mesh models, and variations in the algorithm's hyper-parameters are analyzed. The thesis also proposes ideas and enhancements to improve the approach, which can be implemented in future work.

Internal Supervisor:
Markus Schmitz M.Sc.,
IGMR RWTH Aachen University

External Supervisor:
Guido Schröer,
Siemens Mobility GmbH

Eidesstattliche Versicherung

Sawan Saxena Matrikel-Nummer: 394823

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Master Thesis mit dem Titel

Operator Guidance System for Visual Inspections - Dynamic Trajectory Estimation for Complete Coverage of Inspection Items

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 3 April 2023

Sawan Saxena

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 3 April 2023

Sawan Saxena

The present translation is for your convenience only.
Only the German version is legally binding.

Statutory Declaration in Lieu of an Oath

Sawan Saxena Student number: 394823

I hereby declare in lieu of an oath that I have completed the present Master Thesis entitled

Operator Guidance System for Visual Inspections - Dynamic Trajectory Estimation for Complete Coverage of Inspection Items

independently and without illegitimate assistance from third parties. I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, 3 April 2023 Sawan Saxena

Official Notification:

Para. 156 StGB (German Criminal Code): False Statutory Declarations

Whosoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence

(1)If a person commits one of the offences listed in sections 154 to 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly. I have read and understood the above official notification: :

Aachen, 3 April 2023 Sawan Saxena

Contents

List of abbreviations **viii**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Related Work . . . 2
1.3 Problem Statement . . . 2
1.4 Thesis Outline . . . 4

2 Theoretical Background **5**
2.1 Fundamentals of 3D Models . . . 5
2.2 Coverage Path Planning . . . 6
2.2.1 Coverage Calculation . . . 7
2.2.2 Coverage Path Planning for Unmaned Aerial Vehicle (UAV)s . . . 7
2.3 Trajectory Generation . . . 10
2.3.1 Voxelization: Extention to Cellular Decomposition . . . 10
2.3.2 Obstacle Management . . . 15
2.3.3 Traveling Salesman Problem . . . 16
2.3.4 Greedy Approach for Trajectory Generation . . . 17

3 Data Acquisition and Processing Tools **19**
3.1 Data . . . 19
3.2 Libraries and Software . . . 22
3.3 Evaluation . . . 25

4 Methodology **26**
4.1 Coverage Calculation . . . 26
4.1.1 Generating Initial Viewpoints Set . . . 27
4.1.2 View Points Selection . . . 34
4.2 Trajectory Generation . . . 39
4.2.1 Sequencing the Generated ViewPoints . . . 40
4.2.2 Obstacle Management using Voxelisation . . . 42

5 Results and Analysis **46**
5.1 Coverage Calculation . . . 46
5.2 Trajectory Generation . . . 59

6 Conclusion **63**
6.1 Summary 63
6.2 Future Work 64

Bibliography **I**

List of Algorithms **IX**

List of Figures **X**

List of abbreviations

General abbreviations

FFT	Fast-Fourier-Transformation
UAV	Unmaned Aerial Vehicle
AGV	Automated Guided Vehicles
ROI	Region of Interest
CPP	Coverage Path Planning
ACO	Ant Colony Optimizer
TSP	Traveling Salesman Problem
GA	Genetic Algorithm
CUDA	Compute Unified Device Architecture
RRT	Rapidly Exploring Random Tree
RGB	Red Green Blue
SLAM	Simultaneous Localization and Mapping

1 Introduction

Unmanned vehicles, such as Automated Guided Vehicles (AGV), can take over routine tasks previously done by humans. Automated vehicles require advanced algorithms to detect and avoid static and moving obstacles and systems for path planning and traffic management. Mobility becomes an order of magnitude more difficult when the vehicle moves through the air. However, UAVs, popularly known as drones, have high potential in outdoor and indoor manufacturing environments [MN19]. Indoor industrial applications of drones are much less common than outdoor applications. For example, drones are already finding applications in emergency operations, infrastructure maintenance, logistics, and warehouse management [MN19]. Indoor drone flights pose additional challenges compared to outdoor flights, such as navigation, obstacle detection, explosion risk, and collision avoidance in the three-dimensional (3D) factory space. There is also a considerable risk to humans and assets when drone systems fail in indoor environments. Other limiting factors are noise and privacy concerns. Most importantly, established alternatives to drones, such as floor-based AGVs, conveyor belts, and wall- or roof-mounted cameras and sensors, can do many tasks usually attributed as potential applications of drones (e.g., surveillance, inspection, or intralogistics) [MNF21].

1.1 Motivation

Drones-based inspection of Assets and items is becoming a mainstream topic. It saves time and can increase the efficiency of an inspection process. The quality inspector gets a comprehensive overview of the assets without being on-site or close to them. Nevertheless, this task requires continuous quality for the inspection task. When a human does such a job repeatedly, flaws can quickly occur, and parts of the inspection might be missing. Sometimes, the traceability of flaws and defects is also compromised.

Towards a fully automated drone-based inspection, there is a hybrid transition phase where we still rely on a human pilot to record the asset or objects, especially when the objects for the inspection may be in varying or unknown places and orientations.

Digital representation of the environment is crucial in germinating the solutions for fully automated Inspection processes. It offers grounds for running simulations and analyzing the environment.

It is developing an approach to checking the assets(object) coverage on the fly or while exploring the environment via Drone. This approach helps the pilot to ensure he/she records/scans all Regions of Interest (areas) that are relevant and necessary.

1.2 Related Work

There have been a lot of research papers considering the generation of clouds of viewpoint covering the whole object. Such as image-based rendering from different viewpoints of a 3D object to synthesize new viewpoints [VFS01], where many camera viewpoints are generated using *viewpoint_entropy* over the bounding sphere, determining the amount of the scene that is visible through a camera viewpoint. In Online-Trajectory planning, Simultaneous Localization and Mapping (SLAM)s are incorporated to assign key-frames to speed up the process [SMD12]. The other technique makes the reconstruction problem tractable [FCS10; SSS08]. There has been some work done on short-term UAV path planning where nominal key-frame based SLAM system fuses both feeds sensor cues from both the monocular camera and inertial measurement unit [ATC17]. There have been many studies on multiple drone implementations for coverage planning either by swarm autonomy or base coordination [CWL09; CGT19]. The focus of this research is primarily to remove the involvement of the pilot in the movement of the drones. This is done by tethering the drones to some smart device that provides path information to the drones [ZCZ18].

Research has been done, which includes the area coverage where points of interest are known beforehand, interpreted as Traveling Salesman Problem (TSP) for single or multiple pathfinding [OY21; CK21]. There have been many propositions for solving TSP problems. Some of the nature-based methods proposed are Bee Colony Optimization is very beneficial in solving the optimized problems [ZCZ18; Teo09]. Ant Intelligence is also a very beneficial approach to solving problems [DGB06; sel10]. The approaches related to full area coverage of the object covering all Region of Interest (ROI)s could be found in Cabreira et al. [CBP19]. These researches come under Coverage Path Planning (CPP) as discussed in Section 2.2.

1.3 Problem Statement

To generate an efficient trajectory that ensures complete coverage of the item being inspected, initiation with computing the trajectory around the 3D mesh model of the item/Asset and treating the 3D mesh model as the only possible obstacle for trajectory

generation in a digitized environment is essential. An application for guiding the pilot to maneuver the drone is needed. This application would ensure the coverage of the whole item covering all ROIs and the shortest path from a random initial drone position. Figure 1.1 shows the 3D mesh model of the Electric Locomotive, which is heavily detailed with around 1001264 triangle faces. This offers several challenges that associate triangles to the ROIs; many would be redundant due to proximity.

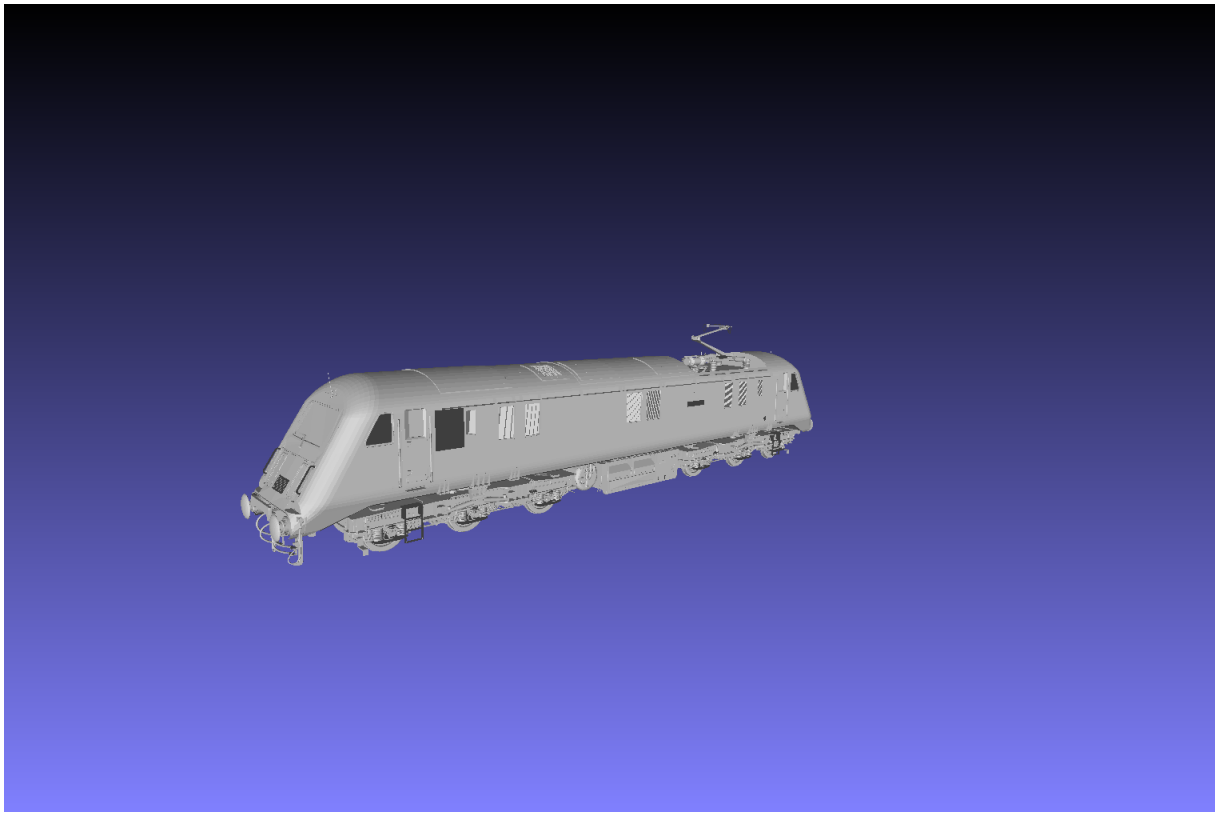


Figure 1.1: 3D Mesh Model

Based on the problem description described above, the following research questions examined in the thesis are:

- Is there a possibility to generate a trajectory that ensures the coverage of all ROIs considering a ground tolerance of how close a drone can fly near the ground?
- Is there any efficient way to generate the shortest trajectory visiting all the essential viewpoints?
- How are the obstacles managed when generating the trajectory?

1.4 Thesis Outline

The thesis is divided into six chapters. **Chapter 2** contains the theoretical background on the fundamentals of 3D models literature review on Coverage Path Planning. This chapter also mentions sub-modules like Voxelisation, obstacle management, Traveling Salesman Problem, and Greedy Approach for path planning. These are essential in designing the algorithms for this thesis. **Chapter 3** details the data used to implement and test the algorithms. It also mentions the libraries and software used to implement the algorithms and the hardware specification. **Chapter 4** discusses the methodology for generating viewpoints for coverage path planning and generating the trajectory passing through all the generated viewpoints implementing the fundamentals of Multi-Goal Travelling Salesman Problem and Path Planning using the Greedy Approach. **Chapter 5** discusses the results of the implemented algorithms. Analysis of the variation of hyper-parameters and its impact on the results can be found in the chapter. The Conclusion comprising of Summary and Future Work formulates **Chapter 6** and compiles the conclusions and results to get a macro understanding of all the steps and processes involved and some proposals for enhancements as future extensions to this work.

2 Theoretical Background

This chapter details the fundamentals of a 3D model along with a literature review on Coverage Path Planning and Trajectory Generation. Coverage Path Planning includes coverage calculation fundamentals and its integration in path planning for general robots and UAVs. Various topics such as Voxelization, Obstacle Management, Travelling Salesman Problem, and Greedy Approach that are essential for generating the trajectory are also contained in this chapter.

2.1 Fundamentals of 3D Models

In 3D computer graphics and solid modeling, a polygon mesh is a collection of vertices, edges, and faces that define the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons (n-gons) since this simplifies rendering. However, they may also be more generally composed of concave polygons or even polygons with holes.

The study of polygon meshes is a significant sub-field of computer graphics (specifically 3D computer graphics) and geometric modeling. Different representations of polygon meshes are used for different applications and goals. The various operations performed on meshes may include Boolean logic (Constructive solid geometry), smoothing, simplification, and many others. Algorithms also exist for ray tracing, collision detection, and rigid-body dynamics with polygon meshes. If the mesh's edges are rendered instead of the faces, the model becomes a wireframe model. Volumetric meshes are distinct from polygon meshes in that they explicitly represent a structure's surface and volume, while polygon meshes only explicitly represent the surface (the volume is implicit).
Objects created with polygon meshes must store different types of elements. These include vertices, edges, faces, polygons, and surfaces, as shown in Figure 2.1. Only vertices, edges, faces, or polygons are stored in many applications. A renderer may support only 3-sided faces, so polygons must be constructed of many of these, as shown above. However, many renderers either support quads and higher-sided polygons or can convert polygons to triangles on the fly, making storing a mesh in a triangulated form unnecessary [THI16].

- **Vertex** A position (usually in 3D space) and other information such as color, normal vector, and texture coordinates.
- **Edge** A connection between two vertices.

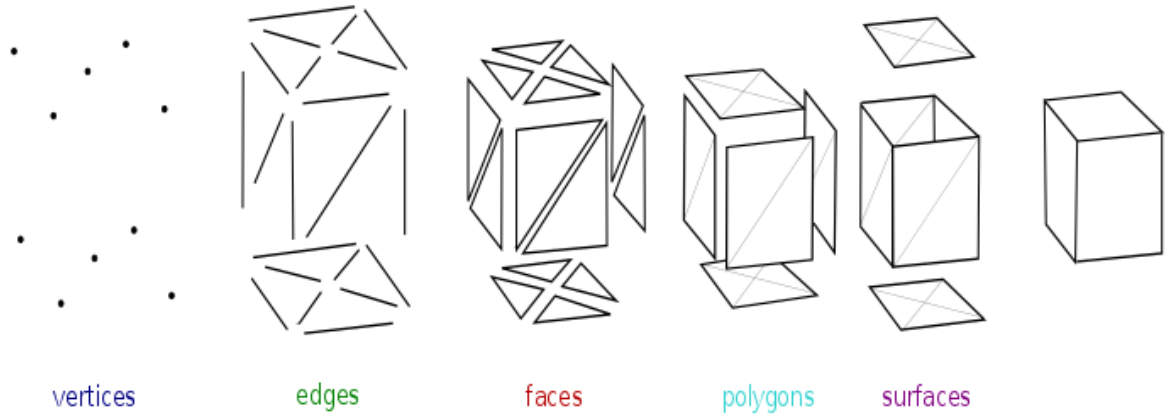


Figure 2.1: Components of a 3D Mesh Model [ČVU18]

- **Face** A closed set of edges is called Faces. The Faces can have 3 edges (Triangle Face) or 4 edges (Quad Face). A Polygon is a coplanar set of faces. Mathematically a polygonal mesh may be considered an unstructured grid, or undirected graph, with additional properties of geometry, shape, and topology.
- **Surfaces** In the mesh smoothing process, the mesh's topological structure is fixed, i.e., the nodal connections of the elements will not be altered. Still, the interior nodes are repositioned to produce triangles with somewhat improved shapes [ZTZ13]. The computationally efficient smoothing algorithm is the well-known Laplacian smoothing which repositions the internal node at the centroid of the polygon formed by its neighboring nodes.
- **Materials** Generally, materials will be defined, allowing different mesh portions to use different shaders when rendered. [Ope]

2.2 Coverage Path Planning

CPP determines a path that passes over all points of an area or volume of interest while avoiding obstacles. This task is integral to many robotic applications, such as vacuum cleaning robots, painter robots, autonomous underwater vehicles creating image mosaics, demining robots, lawnmowers, automated harvesters, window cleaners, and inspection of complex underwater structures, to name a few. One of the earliest works on CPP in the literature defined the requirements a robot must meet to perform a coverage operation. Albeit the target application in the aforementioned paper is a mobile robot moving in a flat 2-dimensional environment, the same criteria apply to other coverage scenarios [GC13].

Generally, the CPP problem is related to the covering salesman problem, a variant of the well-known Traveling Salesman Problem where, instead of visiting each city, an agent must visit a neighborhood of each city. However, in CPP, the agent must pass over all points in the target area instead of visiting all the neighborhoods [GC13]. Since the Traveling Salesman Problem is NP-hard, similar to the lawnmowers problem [AFM00], the computational time required to solve the problem increases drastically when the dimension of the problem increases. As per the classification originally proposed by Choset [Cho01], coverage algorithms can be classified as heuristic or complete depending on whether they provide complete free space coverage. Independently, they can be classified as either offline or online. Off-line algorithms rely only on stationary information, and the environment is assumed to be known in advance.

2.2.1 Coverage Calculation

Coverage Calculation for capturing a complete set of 3D points can be interpreted as a cover set problem, NP-Complete [Kar72]. Coverage Calculation has three main steps: Camera Placement, Camera Clustering, and View Selection [HWZ]. Camera Placement could be achieved using the triangles in the mesh and computing the possible camera position for a triangle considering the triangle center and the triangle normal (as shown in Figure 2.2, there are constraints on the angle made by the viewing direction with the triangle normal). Camera Clustering combines the viewpoints that have a similar view of the object. The constraints for viewing angle and visible area can be visualized in Figure 2.2. The clustering is primarily to set the importance of each camera viewpoint. Finally, viewpoint selection is done to reduce the number of generated viewpoints to the necessary number of viewpoints required for complete coverage of the 3D model. Multiple constraints could be implemented for a selection to be made.

2.2.2 Coverage Path Planning for UAVs

UAVs are the latest implementation of CPP, making it an extension to the classic 2D problem. In many UAV missions, the main purpose is to visit some predetermined checkpoints in operational space. Finding a feasible solution may take too long if the number of checkpoints and constraints increases. Galceran et al. [GC13] states that the problem is NP-Hard with increased Operation Space Dimensions. As per Pehlivanoglu et al., [PP21], the path planning problem of autonomous UAV in target coverage problems can be solved by using artificial intelligent methods including Genetic Algorithm (GA), Ant Colony Optimizer (ACO), Voronoi diagram, and clustering methods. The main contribution of this

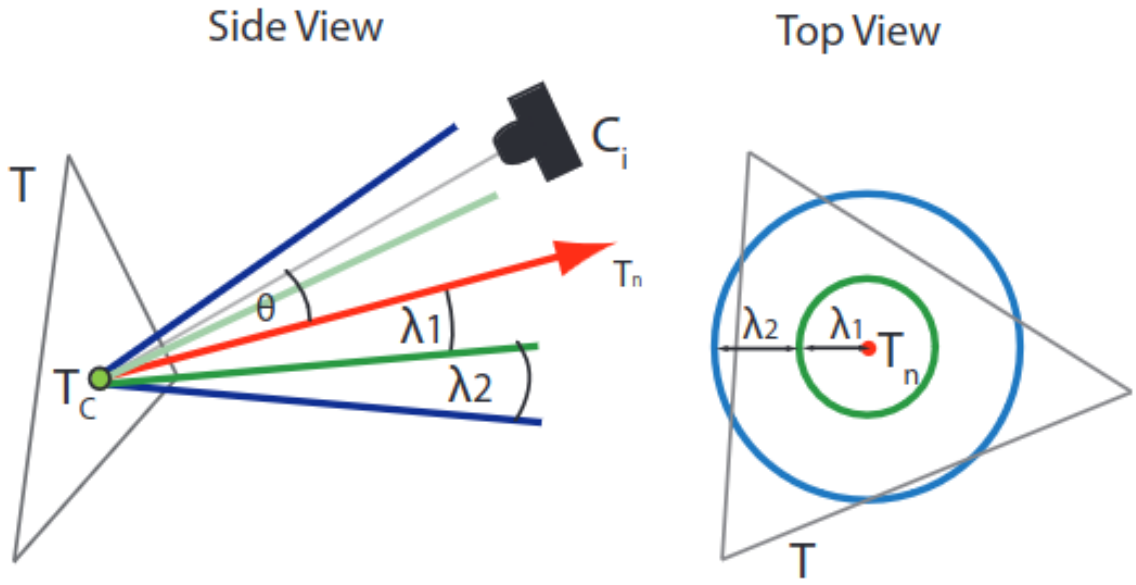


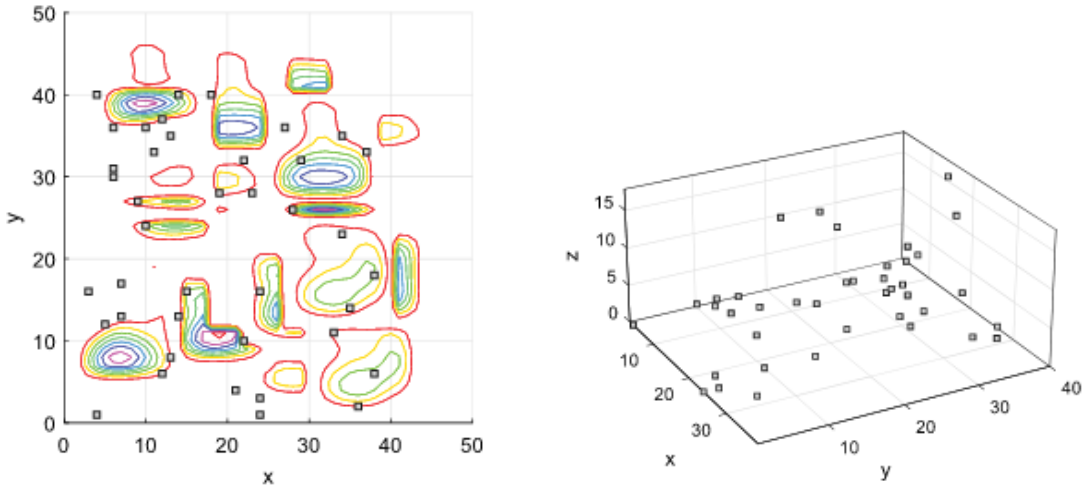
Figure 2.2: Camera Position Computation [HWZ]

article is to propose initial population enhancement methods in GA and thus accelerate the convergence process. The first common enhancement to the basic GA structure is to generate a sub-optimal path by implementing ACO. A sub-optimal path can be used to generate initial individuals. However, sub-optimal paths may have the problem of colliding with terrain. To avoid a UAV from any crash, three approaches are integrated into an initial population phase of the genetic algorithm. One of the approaches stated in Pehlivanoglu et al. [PP21] includes Voronoi vertices as additional waypoints to keep clear of trouble. The second approach consists of cluster centers that form Voronoi vertices as supplemental waypoints and the final approach comprises cluster centers based on a set of collision points again. The proposed methods are tested in different three-dimensional (3D) environments, and the results are compared. Performance results show that collision with the terrain surface is a local phenomenon. Solving this issue using the cluster center of collision points provides the best result, including at least 70 percent or much more decrease in the required number of objective function evaluations [PP21].
However, ACO and GA is not required for our use case due to the low complexity of the model to be covered. But it gives an insight into precomputing the viewpoints associated with the ROIs to plan the path using multi-goal TSP algorithms. Hence, there is a need to find an approach to precompute the positions where the camera must be to ensure full coverage.
According to Pehlivanoglu et al. [Peh12], Path Planning Problems can be classified into the following categories:

- **Start-Goal Problem** includes finding the shortest path between two locations. In target coverage path planning problems, UAV is addressed to determine the optimal

path which can fully cover a given area of interest. Targets can be categorized into point, line, and area targets. In a flight mission, targets may be homogeneous or heterogeneous. The literature has hybrid-type problems, such as start-goal problems and checkpoint/target coverage issues [Sko07].

- **Check-Point based UAV Path Planning Problem** is a target coverage problem that visits a set of target points. In most existing studies, the main purpose of checkpoint-based vehicle path planners is to generate a path for the aerial vehicles to visit all of the checkpoints while minimizing or maximizing a certain objective function under several constraints [MMP04].



(a) Check-Points with object contours **(b)** Check-Points

Figure 2.3: Check-Point-based Path Planning [PP21]

The checkpoint-based method is an NP-Complete problem due to its complex nature. There is no optimal solution in polynomial time, so this problem becomes a continuous optimization problem. As stated in Pehlivanoglu et al. [Peh12], finding the optimal path using classical continuous optimization methods is almost impossible. Converting it to a combinatorial optimization problem and finding an optimal sequence sounds feasible. The route can be determined via points connected by straight-line segments. To overcome the un-flyability due to the inability of the drone to make instantaneous turns through each target, each line segment can be smoothed to produce a continuous path [Dor].

Choset et al. [Cho01] describe another form of Coverage Planning approach consisting of Heuristics and Randomization. Here the exploration of the search space happens while following the Heuristics. For example, a robot following the wall has a heuristic parameter of repulsion from the other robot [MB96; BA94; Bro86], which helps both the robots to explore the search space. However, this random behavior does not guarantee full coverage.

This makes precomputed algorithms more relevant to our topic, where full coverage is guaranteed.

An older paper, Choset et al. [Cho00], describes a technique without randomization. It follows the Cellular Decomposition Technique, where the search space is decomposed into unit cells. The Boustrophedon Cellular Decomposition, described in Choset et al. [Cho00], follows the movement of an Ox in a field dragging a plow. Here the robot's free space is broken down into cells such that the robot can cover each cell with back-and-forth Boustrophedon motion as shown in Figure 2.4.

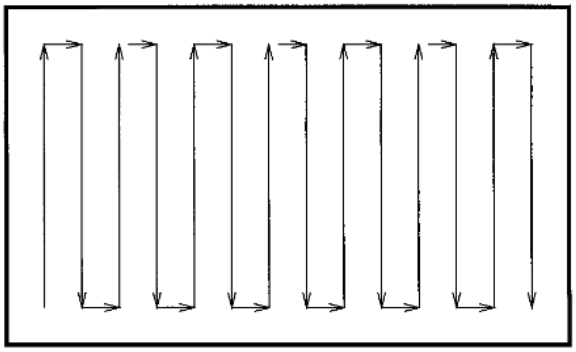


Figure 2.4: Boustrophedon Path [Cho00]

Although this article provides an interesting insight into guaranteed coverage planning, implementation in a 3D environment becomes very complex. However, using Cellular Decomposition for planning the path is feasible.

2.3 Trajectory Generation

Trajectory Generation in a complex environment is challenging in the case of UAVs due to high maneuverability in 3-dimensional space [LEW19]. For safe and efficient operations for these UAVs, obstacle avoidance and digitized environment formulation for designing trajectory planning algorithms are required. Voxelization is one way to discretize and digitize a model or an environment that enables modeling in a very efficient and effective manner. While Travelling Salesman Problem and Greedy Approach for trajectory computation form the backbone of the trajectory computation itself.

2.3.1 Voxelization: Extention to Cellular Decomposition

Covering the point cloud to a parametric model is time-consuming and complex, requiring expensive software and expertise in geometric modeling and mesh manipulation [ST19].

We are only concerned with the outer surface of the Asset for inspection. This makes the process analogous to Tool-tip path planning. The most common surface voxelization method is Rasterisation [CG12].

According to Laine et al. [LK10], the following are the trade-offs between a Triangle mesh and Voxels:

1. **Memory Usage** Triangles and displacement maps are more compact than voxels. But Voxel grids can adapt to finer resolutions to a much finer scale, although smaller in reality. Table 2.1 compares the memories consumed by both.

Table 2.1: Memory usage per voxel for uniquely textured, displaced triangles, and voxels. All numbers are in bytes. [LK10]

	Triangle	**Voxels**
Geometry	1	1
Color	0.5	1
Normals	0	2
Contours	-	1
TOTAL	1.5	5

2. **Downsampling** is the most attractive feature of the Voxel representation. It can represent downsampled dense opaque data.

3. **Zooming** is well handled by Triangle Mesh, similar to parametric patches. The sharp features of the object are hard to represent in a voxel representation, especially at higher resolutions.

4. **Deformation** Generic Deformation of triangles is a very efficient process, whereas there is no efficient way possible with the voxel representation.

5. **Authoring** depends on the creator wholly, as the content creation is entirely based on manual triangle modeling. Hence, this attribute is entirely dependent on the task requirements.

6. **Acquisition** from scanned samples to voxel representation is easier and simpler than conversion to Triangles or displacement maps.

7. **Rendering** Triangle data is very efficient with fast Rasterization, whereas Ray Casting algorithms are more efficient with Voxel Representation.

In the past works, there have been two distinctions in the type of 3D surface Voxelization, namely:

- **Thin Voxelization** is a 6-separating representation of the surface (Figure 2.5). It is cheaper to compute and is often more desirable in computer graphics applications. [HYF98]

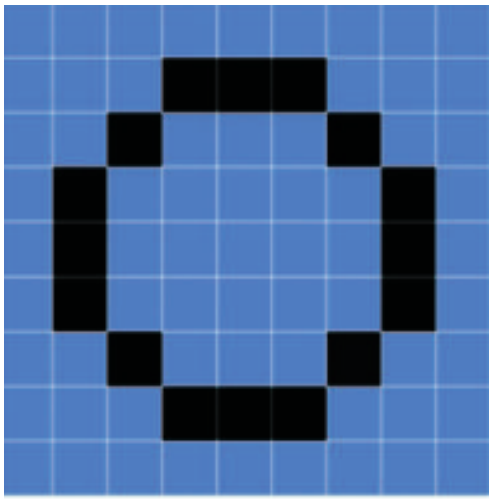


Figure 2.5: 4-separating 2D line rasterization equivalent to 6-separating surface voxelization in 3D [CG12]

- **Fully Conservative Voxelization** is where all voxels overlapped by a surface are activated or 26-separating as shown in Fig 2.6.

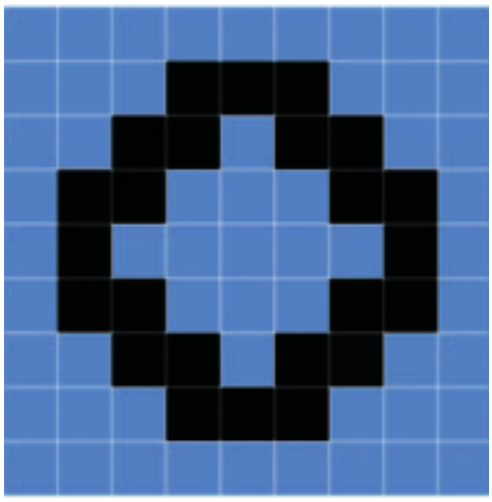


Figure 2.6: 8-separating 2D line rasterization equivalent to 26-separating surface voxelization in 3D [CG12]

Before the development of OpenCL and Compute Unified Device Architecture (CUDA), due to lack of random write access, these approaches had to use a multipass rendering technique, processing the volume slice by slice and retransforming the entire geometry with each pass. While achieving higher performance [ED08; HYF98] , process multiple slices by encoding grids with a compact binary representation [CG12].

Pure data-parallel algorithms provide more flexibility and allow new original voxelization schemes to take advantage of the freedom offered by compute mode. One of the examples is direct voxelization to Spree Tree [SS10; Pan11].

Simple Voxelization Pipeline

The voxelization pipeline based on the Thin Voxelization technique proposed by Crassin et al. [CG12] operates in four main steps inside a single draw call as shown in Figure 2.7.

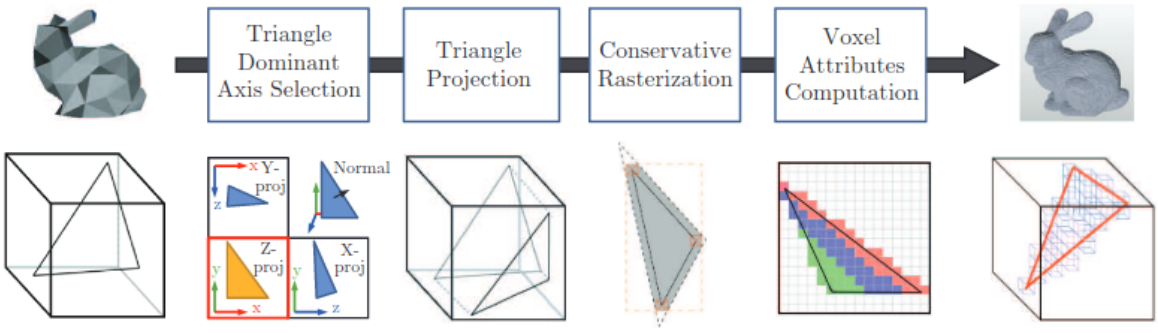


Figure 2.7: Simple Voxelization Pipeline [CG12]

The consisting steps of the Simple Voxelization Pipeline, as mentioned in Crassin et al. [CG12] and as shown in Figure 2.7 are:

1. **Triangle Dominant Axis Selection** consists of the orthographical projection of each triangle along the dominant axis of its normal(one of the main three axes of the scene). This maximizes the projected area, which is beneficial for conservative rasterization.

2. Once the Dominant Axis is selected, **Triangle Projection** is done using classical Orthographic Projection computed inside the Geometry Shader as shown in Figure 2.9.

3. This pipeline does not guarantee 6-separated Thin Voxelization; thus, more precise **Conservative Rasterization** is needed [CG12]. Here, for each projected triangle, a slightly larger bounding polygon is generated (as shown in Fig 2.8) that ensures that any projected triangle touching a pixel will necessarily touch the center of this pixel and thus will get a fragment emitted by the fixed-function rasterizer.

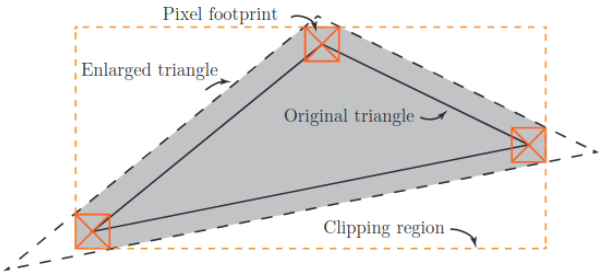


Figure 2.8: Bounding polygon of a triangle used for conservative rasterization [CG12]

4. While writing the values of the Voxel-Fragments generated in Fragment Shader into destination 3D texture, there can be arbitrary order of the voxel fragments from different triangles that ended up in a single voxel destination. Hence, there is a need for **Voxel Fragment Attribute Computation** to compose the Voxel Fragments.

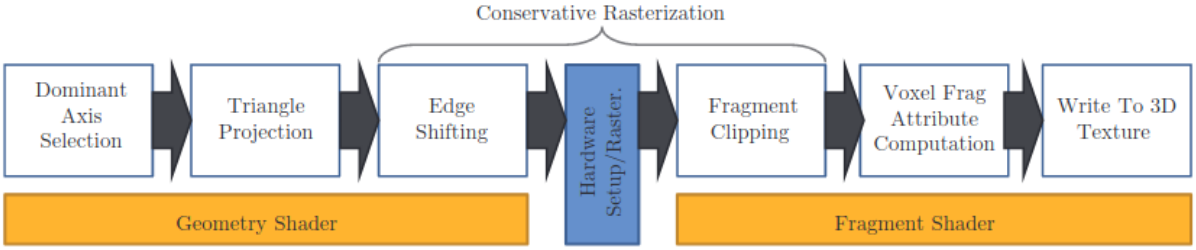


Figure 2.9: Implementation of our voxelization pipeline on top of the GPU rasterization pipeline [CG12]

Sparse Voxelization into an Octree

Sparse Voxelization is beneficial in the case of large and complex scenes. These large and complex objects can be represented using sparse-voxel octree [LK10; CNL09]. The Sparse Voxel tree, as described in Crassin et al. [CG12], is a compact pointer-based structure described in Crassin et al. [CNL09].

The root node of the Octree represents the entire scene, and each of its children represents the eighth of its volume. And as shown in Figure 2.10, this hierarchy follows through every forthcoming node. This structure allows querying filtered voxel data at any resolution. More details could be fetched by descending the tree hierarchy.

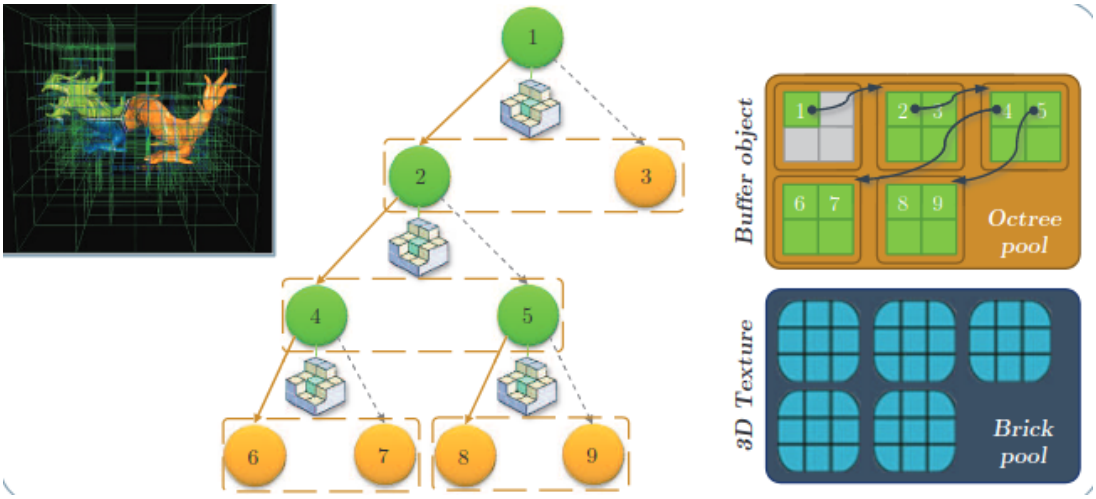


Figure 2.10: Memory Organization in Octree Representation [CG12]

2.3.2 Obstacle Management

Obstacle Management is an integral part of path planning of UAVs with a collision-free track that meets the requirements of continuity and output stability. Thus, a comprehensive study must obtain a track planning method to deal effectively with the most complex environments [YFG22].

Various types of work are cited in Yang et al. [YFG22]. Such as:

- Evers et al. [EBM15] focuses on the uncertainty of UAV mainly due to the existence of most random sampling points in the planned path.
- Zhang et al. [ZML15] solve the UAV track problem using a specific reward matrix.
- Ollero et al. [OLM05] shows that the path planner is essential in the procedure schedule given specific constraints of UAVs.
- Merino et al. [MCD05] focus on increased fire localization accuracy while decreasing the uncertainty of fire detection.
- Oh et al. [OSK15] share the concept of differential geometry for cooperative missions or path planning of multiple UAVs.

Rapidly Exploring Random Tree (RRT) Algorithm

Yang et al. [YFG22] optimized the new idea of expanding trees initially proposed by Feng et al. [FL14] and improvements made on tree nodes by Lin et al. [LWW13]. Path planning with generic RRT, once the leaf nodes of the random tree are included in the

points in the target area, the expansion is stopped, and the line connecting the 2 points is displayed.

Extension to RRT, as described by Yang et al. [YFG22] is shown in Figure 2.11

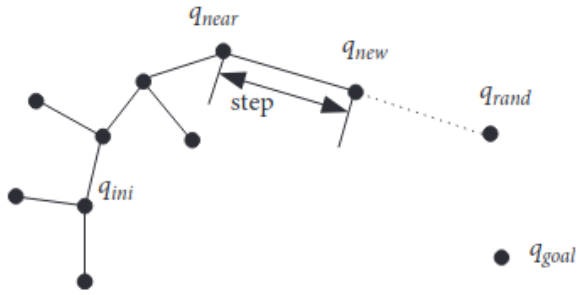


Figure 2.11: RRT algorithm expansion process [YFG22]

As shown in Figure 2.11, the RRT algorithm has **T** as the existing expansion tree and $\mathbf{q}_{\text{rand}}$ as the random sampling point in the state space. The selection of $\mathbf{q}_{\text{rand}}$ is based on a certain probability and the selection of a random point with the complementary probability. Apart from $\mathbf{q}_{\text{rand}}$, there is the closest tree node $\mathbf{q}_{\text{near}}$, and then the connection of $\mathbf{q}_{\text{rand}}$ and $\mathbf{q}_{\text{near}}$ a new $\mathbf{q}_{\text{new}}$ is intercepted. If no obstacle is found while moving to the new node $\mathbf{q}_{\text{near}}$, then $\mathbf{q}_{\text{near}}$ is added to the expansion tree; otherwise, the $\mathbf{q}_{\text{rand}}$ is re-selected. The iterative calculation is continued until $\mathbf{q}_{\text{new}}$ reaches the target area. The algorithm ends with the path beginning from $\mathbf{q}_{\text{ini}}$ to $\mathbf{q}_{\text{goal}}$ to be added to expanding tree, **T**.

2.3.3 Traveling Salesman Problem

TSP is one of the most talked about problems in path planning. The origin of TSP could be traced back to "The Traveling salesman problem: a guided tour of combinatorial optimization" [Law85]. Primarily TSP determines a Hamiltonian path of minimum length on a grid/graph so that the salesman visits all the vertices. A slight variation to the Hamiltonian path could be Euclidean TSP, where the minimum length is found in the Euclidean space [CBS15]. The algorithms for solving ETSP could be categorized into exact and heuristic. Heuristic algorithms are mainly featured for producing good solutions in acceptable computational time. However, optimality is not guaranteed. In the case of Exact Algorithms, they tend to find optimal solutions. The drawback for exact algorithms is the computational time to generate optimal solutions [CBS15]. The heuristic approaches are feasible if we look for trajectories generated from many viewpoints. Heuristics proposed by Lin et al. [Lin65] offer the solution to the Travelling Salesman Problem based on the proximation of viewpoints to the neighboring viewpoints.

Generally, a grid is created where all the necessary positions to be covered are captured. These potential positions are the keyframe points used in the algorithm [Lin65]. CETSP is a variation in the algorithm where a sphere surrounds each viewpoint, and it is ensured that there are no overlaps between the two distinct points. The Hamiltonian path with minimum distance value is computed by sequencing the points hitting the spheres defined above [CBS15]. The algorithm is based on a Non-linear programming subproblem and branch-and-bound to determine optimal routes in two or three dimensions.

2.3.4 Greedy Approach for Trajectory Generation

Combining various path-planning algorithms with the Greedy approach helps solve time-complex problems with close-to-optimal solutions if heuristics are chosen appropriately. Similar work could be seen in Xiang et al. [XLO22], where the greedy approach is combined with the A-star pathfinding algorithm to find the optimal path in polynomial time. A greedy algorithm always makes locally optimum choices. While greedy algorithms tend to give faster solutions, they are not optimal [CSu09]. The greedy approach comes into consideration when dealing with activity selection tasks [CSu09], and while TSPs are selection-based problems, the greedy approach selects the best location from a current location and gives us fast sub-optimal solutions. The recursive greedy algorithms are beneficial in bypassing Dynamic Programming and instead using activity selectors according to the heuristics in a recursive mode. A Greedy algorithm computes a sub-optimal solution by making a sequence of choices. A generic Greedy algorithm has the following necessary elements [CSu09]:

1. Determination of optimal substructure of the problem. A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to subproblems within it.

2. Development of recursive solution, where recursive activity is formulated. The recursive activity is in correspondence to the above substructure.

3. Opting for only one sub-optimal selection using greedy choice based on heuristics. When considering which choice to make, we make the choice that looks best in the current problem without considering the results of a global problem.

4. Checking for constraints when making a greedy choice, if the constraints are not met, steps 3 and 4 are repeated.

5. The whole algorithm is designed keeping the above steps into consideration.

6. The recursive algorithm is implemented in an iterative format to review all the possible selections.

3 Data Acquisition and Processing Tools

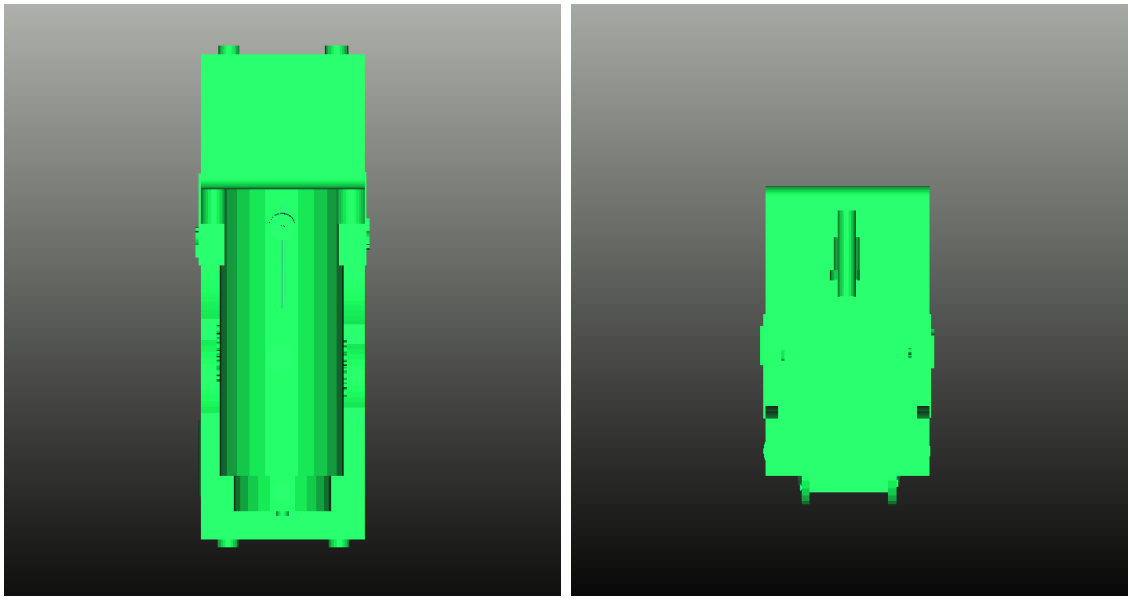
This chapter contains details on the 3D models that were used for implementation and generating results for the implemented algorithms under the subheading Data (3.1). The libraries and the software used for implementation and analysis are also contained in the chapter under the subheading Libraries and Software (3.2). Finally, the chapter also enlists the specification of the computing system used for the thesis.

3.1 Data

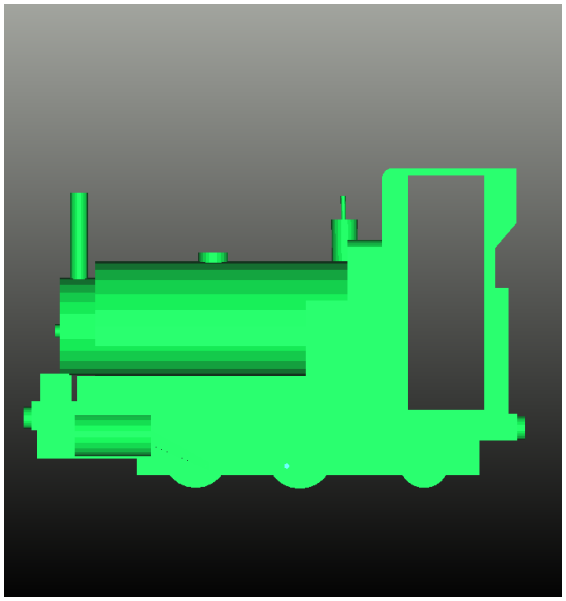
Two models have been considered for algorithm creation and generating results. The toy Train model has been primarily used to develop the Viewpoint generation, selection, and Trajectory generation algorithm due to its simple shapes and features and significantly fewer triangles. This is especially beneficial while creating implementation and debugging. And the decimated version of the Train Model has been used for generating results for the algorithms. The decimated model contains fewer triangles than the original, but the shape and feature complexity remains comparable.

Toy Train Model The model used for developing the algorithms has 3335 vertices and 13754 triangular faces. Figure 3.1 shows that the model has two protruding features from the main body, the chimney, and the driving compartment's roof. These features were very beneficial for designing the obstacle management algorithm, as these protrusions provided obstacles for the proximal viewpoints. The primary reason to use this model is a lower number of triangular faces while providing analogous features to our main model.

Decimated Train Model This model is part of the problem statement; it resembles the real-world situation for the inspection process. The original model has 499446 vertices and 1001264 triangular faces. Due to the very high number of triangular faces, the model was decimated (as shown in Figure 3.2) to 35367 vertices and 80097 triangular faces. This decimation aided in retaining the structure of the outer facade, meanwhile reducing the number of faces. It can be observed that the basic structure of the model is not degraded (comparing the original model in Figure 1.1 and decimated model in Figure 3.2) while all irrelevant triangles that only add to the minuscule details that are not required for the generation of the viewpoints.

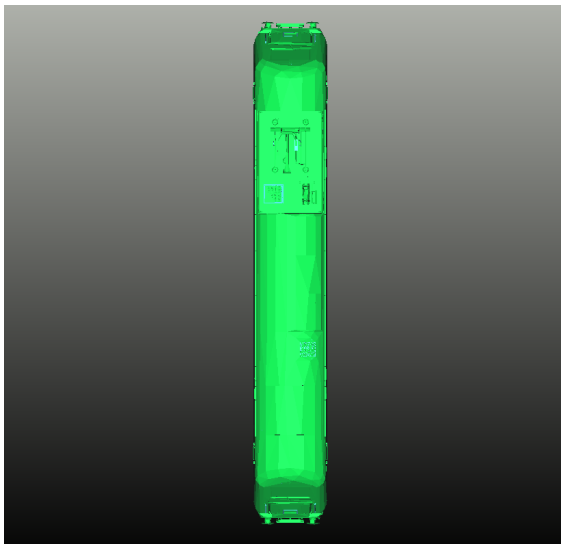


(a) Top View **(b)** Front View

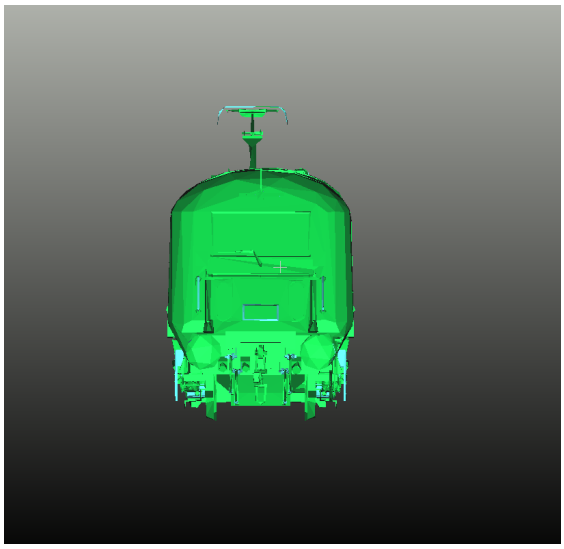


(c) Side View

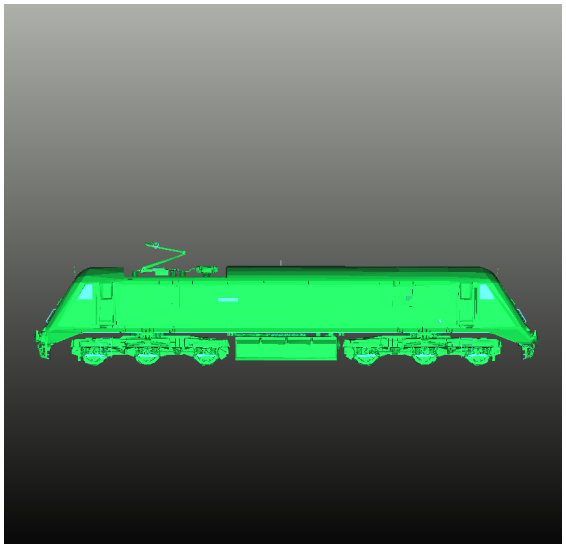
Figure 3.1: Toy Train Model



(a) Top View



(b) Front View



(c) Side View

Figure 3.2: Decimated Train Model

3.2 Libraries and Software

Primarily three libraries are used for the development of the algorithms. Open3D for processing 3D mesh models, OpenCV for processing 2D rendered Images, and NumPy for matrix manipulation.

Open3D It is an open-source library that supports the rapid development of software that deals with 3D data [ZPK18]. The Open3D front end exposes a set of carefully selected data structures and algorithms in C++ and Python. The backend is highly optimized and is set up for parallelization. It provides data structures for three representations: point clouds, meshes, and RGB-D images. There has been a complete set of basic processing algorithms for each representation, such as I/O, sampling, visualization, and data conversion. In addition, we have implemented a collection of widely used algorithms, such as normal estimation [Pau03]. Open3D has nine modules, namely:

1. Geometry: Data structures and basic processing algorithms. There are two primary data structures:
   - Point Cloud: It has three data fields: PointCloud.points, PointCloud.normals, PointCloud.colors. They are used to store coordinates, normals, and colors.
   - Triangle Mesh: It has two master data fields: $TriangleMesh$.vertices and $TriangleMesh$.triangles, and three auxiliary data fields: $TriangleMesh$.vertex_normals, $TriangleMesh$.vertex_colors, and $TriangleMesh$.triangle_normals.
2. Camera: Camera model and camera trajectory
3. Odometry: Tracking and alignment of RGB-D images
4. Registration: Global and local registration
5. Integration: Volumetric integration
6. Input/Output: Reading and writing 3D data files
7. Visualization: A customizable GUI for rendering 3D data with OpenGL
8. Utility: Helper functions such as console output, file system, and Eigen wrappers
9. Python: Open3D Python binding and tutorials

Geometry, Odometry Input/Output modules were primarily used for the implementation.

OpenCV It is an open-source, computer-vision library for extracting and processing meaningful image data. That meaningful data might include finding all or parts of objects, recognizing all or parts of objects, tracking the movement of (parts of) objects in 2D or 3D between successive images, determining the 2D or 3D shape of objects from one or more images, and associating image data with a categorical meaning [Bra00]. The library offers the following 9 modules along with a full matrix algebra package:

1. Geometric Methods: contours, space tessellation, and triangulation.
2. Image Measures: image statistics, spatial moments, and contour moments.
3. Utilities: image pyramids, data structures, linked lists, image management, mathematical functions, fast pixel access, line and conic section drawing, and text display.
4. Segmentation: image morphology, thresholding, color and texture pyramids, background subtraction, histogram back projection, HMM Viterbi, K-means, and normalized cut.
5. Feature Detection: Hough transforms, Canny edge detector, corner detection, subpixel accurate line and corner location, contours, and image derivatives to the third order.
6. Recognition: histogram matching, template matching, Mahalanobis distance, HMM, embedded HMM, shape descriptors, Eigen objects, LDA, 3D gesture.
7. Tracking: mean-shift, CAMSHIFT, optical flow, affine flow, motion templates, energy snakes, Kalman, and Condensation filters.
8. Camera: calibration, view morphing, 8-point algorithm, correspondence, and stereo support.
9. Shape: 2D and 3D line, ellipse fitting, and shape toolbox.

OpenCV was utilized to compute image encodings and render images from the Open3D View Control.

NumPy It is the fundamental package for scientific computing in Python [Num]. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, etc. At the core of the NumPy package is the array object. This encapsulates

n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. NumPy fully supports an object-oriented approach, starting, once again, with ndarray. For example, ndarray is a class possessing numerous methods and attributes. Its methods are mirrored by functions in the outermost NumPy namespace, allowing the programmer to code in the preferred paradigm. This flexibility has allowed the NumPy array dialect and NumPy ndarray class to become the de-facto language of multi-dimensional data interchange used in Python.

Blender It is a free and open-source 3D creation suite [Bleb]. With Blender, 3D visualizations such as still images, 3D animations, and VFX shots can be created. Video editing is also possible. Blender runs a cross-platform application on Linux, macOS, and Windows systems. It also has relatively small memory and drive requirements compared to other 3D creation suites. The interface is based on OpenGL to provide a consistent experience across all supported hardware and platforms.

Blender was useful in decimating the original 3D mesh model to a computationally friendly version (as mentioned at 3.1). Decimate Modifier is the tool in Blender used to reduce a mesh's vertex/face count with minimal shape changes [Blea]. This is not usually used on meshes created by modeling carefully and economically (where all vertices and faces are necessary to define the shape correctly). But if the mesh results from complex modeling, sculpting, and/or applied Subdivision Surface/ Multiresolution modifiers, the Decimate one can reduce the polygon count for a performance increase or simply remove unnecessary vertices and edges. There are various options (enumerated below) to reduce the triangle count that can be found under the Collapse module of the Decimate Modifier:

1. **Ratio**: ratio of faces to keep after decimation, where 1.0 means no change at all and 0.0 means all triangular faces/vertices removed.
2. **Symmetry**: To maintain symmetry on a single axis.
3. **Triangulate**: To keep any resulting triangulated geometry from the decimation process.
4. **Vertex Group**: A vertex group that controls what mesh parts are decimated.
5. **Factor**: The amount of influence the Vertex Group has on the decimation.

3.3 Evaluation

To develop and test the algorithms, Lenovo Workstation has been used with the following configuration:

- Hardware Model: Lenovo ThinkStation P520
- Random Assess Memory: 64 GiB
- Processor: Intel® Xeon(R) W-2145 CPU @ 3.70GHz X 16
- Graphics Card: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti Rev. A]
- Disk Capacity: 1.5 TB
- Operating System: Ubuntu 22.04.2 LTS
- Operating System Type: 64-bit
- GNOME Version: 42.5
- Windowing System: X11

4 Methodology

The methodology adopted for this topic is first getting a set of initial viewpoints as described in Section 4.1 that covers the whole object with all ROIs, and then pruning out the unnecessary viewpoints. To do so, we are using the Greedy approach. After getting the pruned set of viewpoints, a trajectory is generated as described in Section 4.2 on the Multi-Goal Travelling Salesman Problem principle.
The idea proposed in this thesis ensures full coverage keeping the drone's proximity to the surroundings and the object as a constraint while getting a sequence of these viewpoints if we start from a random viewpoint.

4.1 Coverage Calculation

Getting a minimal set of viewpoints covering all ROIs is analogous to the algorithm proposed by Hoppe et al. [HWZ]. This subtask aims to generate a subset of viewpoints covering the whole object, as shown in Figure 4.1. The idea here is to generate as many viewpoints as possible that provide a surety of full coverage. After we have an initial set of viewpoints, we try to remove the redundant viewpoint (the viewpoints covering the same area multiple times). The priority is given to the viewpoints which are covering the most amount of area. In this case, the viewpoints only specific to certain area coverage covered by another viewpoint covering other significant ROIs would be pruned. The algorithm proposed in this thesis has two components for coverage calculation: Initial Viewpoints Generation and Selection of Viewpoints.

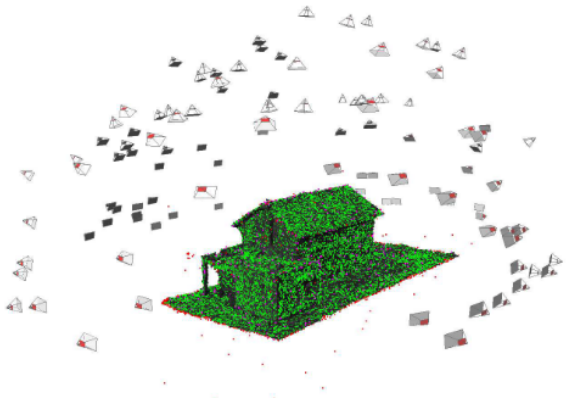


Figure 4.1: A cluster of Camera View Points surrounding the Object [HWZ]

4.1.1 Generating Initial Viewpoints Set

To generate an initial set of viewpoints that ensure that the camera considers all surfaces of the 3D object, it is proposed to generate viewpoints for each triangle in the 3D Object mesh as the combination of triangles constitute a surface. It is a brute-force approach to get the set of initial viewpoints. The idea is to get a camera viewpoint for each triangle at a specified distance along the triangle normal $\vec{\mathbf{N}}$ from the triangle center ($\mathbf{T}_C$) as shown in Figure 4.2. The specified distance from the triangle center is entirely a hyper-parameter and depends on the specifications of the drone's camera and the drone's capability to fly in proximity to the objects (the least proximal distance for a drone to fly). Thus, each triangle in the mesh is given a chance to generate a viewpoint.

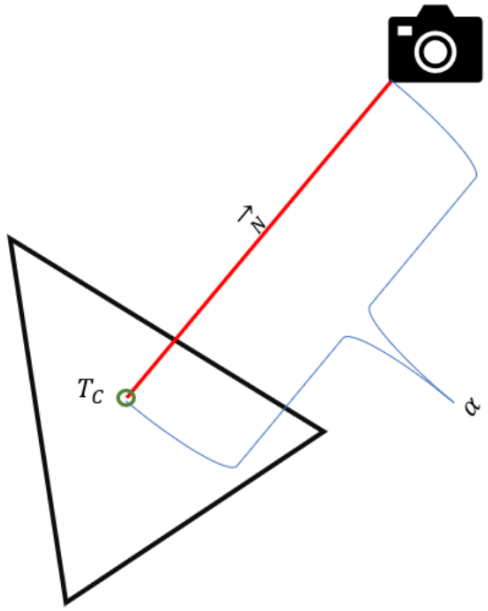


Figure 4.2: Camera View Point for a Trianlge in a 3D Mesh

Triangle centers are chosen instead of triangle vertices because most triangles have very small areas. However, we generated more than twice the initial viewpoints using triangle centers and got almost the same coverage.

Identifying Individual Triangles Uniquely

To distinguish the contribution of each triangle in viewpoint generation and the object's surface area, uniquely identifying each triangle in the 3D mesh is necessary to assign unique Viewpoints to them and to detect the visible area from a particular viewpoint. To get a unique key for a triangle in both Visual and value-based form, unique colors and

corresponding encodings are assigned to all the triangles in the model. This way, a value key can be generated by encrypting the Red Green Blue (RGB) values of the assigned unique color. The Mesh object has a color attribute for vertices only, which means we can uniquely color the vertices but not triangles. The triangle color will somehow be the average of the colors of all three constituent vertices. Essentially, three layers of triangles in place of each triangle are needed so that only a unique color to one vertex of each triangle is assigned. That is why a tripled mesh is created to make all the triangle faces have unique colors. Algorithm 1 shows how the unique colors are assigned to the triangle faces and unique color-key values are generated.

In Algorithm 1, *tripled_mesh* is created by first generating random colors in RGB color scale. As it is a triangular mesh, $faces$.shape will have three columns (pertaining to the number of vertices), and the number of rows is equal to the number of triangular faces in the 3D mesh model. The RGB scale is chosen because the three channels in the RGB are comparable to the number of vertices in the triangular mesh. After generating the random colors, the unique keys for each color with an encoding that is a weighted sum of all three color channel values are randomly generated (as shown in the line 8). To ensure no duplicity with the keys, the presence of the generated key in the list of keys already created is checked. If the existence is found, we generate another random color for that instance and the corresponding key for the newly generated random color.

Once the unique colors and their corresponding encodings are generated, a separate mesh with each vertex corresponds to the random color generated. As shown in the lines 20-26, a triangle gap is given, shifting three triangles so there is no sharing of vertices among the triangles. Hence now, if we assign a color to a vertex, that corresponds to the color of the triangle face.

The model shown in Figure 4.3a is the original 3D mesh model, and once the mesh has been tripled and assigned unique color values, the tripled mesh model (as shown in Figure 4.3b) is obtained with the same number of triangular faces, and each face assigned with unique color.

Generating the Initial Set of ViewPoints

Now that key values have been uniquely assigned to each triangle in the mesh, viewpoints can be generated, and the corresponding rendered view of the mesh model. As a triangle can be visible in more than one viewpoint and one viewpoint captures more than one triangle, it establishes a many-to-many relationship between triangles and viewpoints.

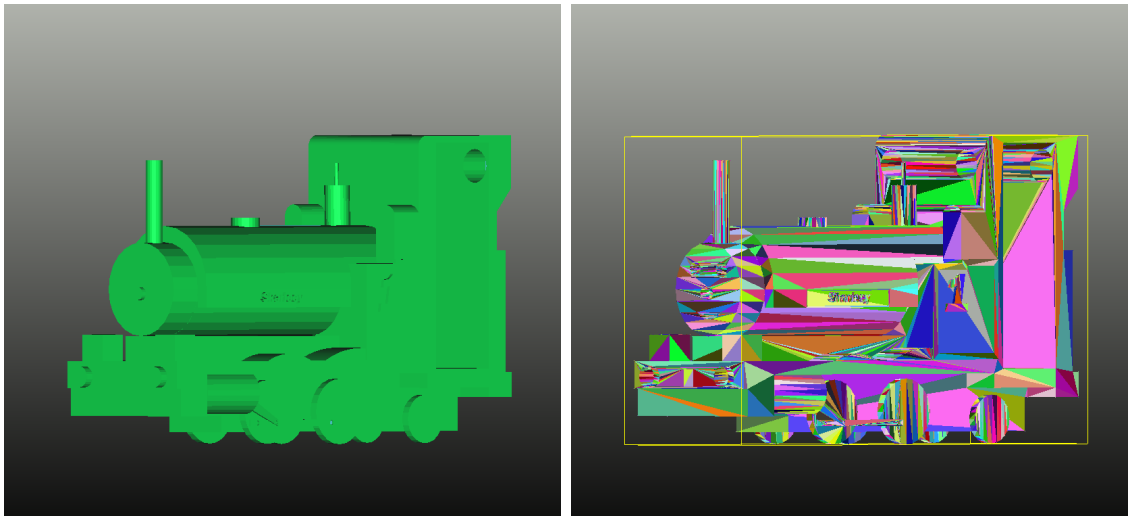
Algorithm 1: 3D mesh tripling and unique Face Color and Keys Generation

Data: mesh
Output: tripled_mesh

```
vertices = array(mesh.verteces)
faces = array(mesh.triangles)
colors = random(between 0 and 254, size=faces.shape) # Generating random color values for 3 color channels RGB
face_colors = {}
# Ensuring that no two faces have the same color
for i in enumerate(colors) do
    while True do
        key = colors[i][0] + colors[i][1]*256 + colors[i][2]*256*256 # Encoding to generate unique keys for color channel values
        if key not in face_colors.keys() then
            break
        end
        colors[i] = random(between 0 and 254, size=3)
    end
    face_colors[key] = i
end
colors = tile(colors, repeat 3 times).reshape(faces.shape[0]*3, 3)
vertices_3 = zeros(len(faces)*3, 3)
faces_3 = zeros(faces)
# Creation of tripled mesh
for index_triangle, t in enumerate(faces) do
    index_vertex = index_triangle * 3
    vertices_3[index_vertex] = vertices[t[0]]
    vertices_3[index_vertex + 1] = vertices[t[1]]
    vertices_3[index_vertex + 2] = vertices[t[2]]
    faces_3[index_triangle] = arange(index_vertex, index_vertex + 3)
end
tripled_mesh = o3d.geometry.TriangleMesh()
tripled_mesh.vertices = vertices_3
tripled_mesh.triangles = faces_3
tripled_mesh.vertex_colors = colors
```

In Algorithm 2, two empty dictionaries, *global_viewpoint_dictionary* and *global_triangle_dictionar* are defined, containing all the corresponding entities to register global associations between them. The structures of a viewpoint entity are shown in Figure 4.4, where we store the key values of the triangles visible from the viewpoint in *triangles* and the other necessary attributes exclusively pertaining to the viewpoint.

Similarly, as shown in Figure 4.5, the number of viewpoints from where the triangle is visible are stored in *viewpoints*. After the selection process, it has an attribute *selected_view_point* to assign a viewpoint to the triangle. *face_key* (as shown in Figure



(a) Original Mesh **(b)** Tripled Mesh

Figure 4.3: Tripling of Mesh to get Uniquely Colored Triangle Faces

Viewpoint	
triangles	List of all triangles visible from the viewpoint
eye	Viewpoint coordinates
front	Viewing direction from the view point
viewpoint_key	Key for accessing a specific viewpoint

Figure 4.4: Viewpoint Entity Structure

Algorithm 2: Initial View Points Generation

```
Data: mesh.triangles
1  α = Distance from the object
2  global_viewpoint_dictionary = Ordered_dictionary()
3  global_triangle_dictionary = Ordered_dictionary()
4  up = [0, 0, 1]
5  for face_key in face_keys.keys() do
6  |  global_triangle_dict[face_key] = Triangle(face_key)
7  end
8  foreach triangle in mesh.triangles do
9  |  compute the triangle center 'C' # Essentially the centroid of the Triangle
10 |  X_i = C + α · triangle.Normal # Computing Camera coordinates
11 |  if X_i[2] ≤ ground tolerance then
12 |  |  # Checking if Z coordinate of the Camera Location above ground tolerance
   |  |    continue
13 |  end
14 |  viewing_direction = X_i - C
15 |  rendered_image = Open3d.render_view_control(look_at=camera_coordinates,
   |    up_vector=up, front=viewing_direction)
16 |  triangle_face_keys = get_face_keys(rendered_image)
17 |  view_point = viewpoint(eye=X_i, front=viewing_direction, viewpoint_key=i)
18 |  foreach face_key in triangle_face_keys do
19 |  |  if compute_angle(triangle[face_key].normal, viewing_direction) ≤
   |  |    angle_threshold then
20 |  |  |  view_point.triangles.add(face_key)
21 |  |  |  global_triangle_dictionary[face_key].viewpoints.add(view_point.viewpoint_key)
22 |  |  end
23 |  end
24 |  global_viewpoint_dictionary.add(view_point)
25 end
```

4.5, the face key is the encoding corresponding to the triangle) and *area* (as shown in Figure 4.5, the area is the computed area of the triangle) are the other necessary attributes exclusive to the triangle.

The *global_triangle_dictionary* (as shown in algorithm 2, line 6) is populated with the color encodings generated in the previous part of the algorithm. Now, the iterations over all the triangles in the mesh are performed. The first step in the iteration is to compute the centroid of the triangle in consideration (centroid computation for triangles represented in 3D coordinates by averaging vertex coordinates). Once the triangle center C (based on the principle shown in Figure 4.2) has been computed, the camera viewpoint coordinates X_i are defined as shown in Algorithm 2, line 10. Here α is a hyper-parameter, which represents the distance to be maintained from the object. After computing the camera viewpoint coordinates, a set of constraints for the viewpoint are checked to qualify for the

Triangle	
viewpoints	List of all viewpoints that where the triangle is visible
selected_view_point	Viewpoint selected for the trianlge (set to none initially)
face_key	Unique triangle encoding (color value encoding)
area	Area of the triangle

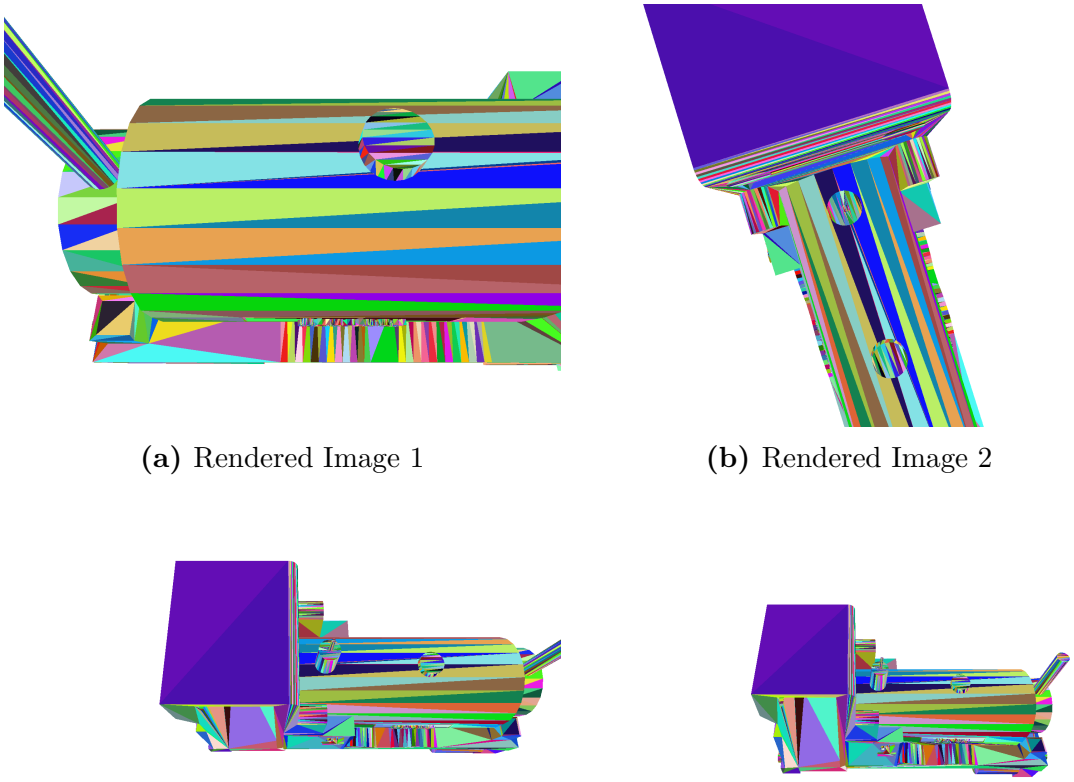
Figure 4.5: Triangle Entity Structure

selection process. The first constraint is the ground tolerance for the UAV. Here, it is checked if the computed camera coordinate is above a certain threshold from the ground by putting a constraint on the Z coordinate of the viewpoint. After the constraint checks, the viewing direction is computed as a vector from the triangle center C to the computed viewpoint X_i.

Now that all the necessary coordinates have been computed, the Open3d ViewControl is utilized to render the projection of the tripled mesh model from the computed viewpoint X_i as shown in Algorithm 2, line 15. The rendered images from four random viewpoints when using Open3d ViewControl are shown in Figure 4.6

After the image has been rendered from the ViewControl, all the triangles visible in the 2D projection are retrieved by first computing the list of encodings (encodings are computed in the same way as in Algorithm 1, line 8) of all the triangles $triangle\_face\_keys$ in the Rendered image. Now the triangles encodings are identified in $triangle\_face\_keys$, and it is checked whether the triangle's Normal $\vec{\mathbf{N}}$ makes an angle less than or equal to the hyper-parameter we set as $angle\_threshold$ with the $viewing\_direction$. The triangle's $face\_key$ is added to the $view\_point$ object if the constraint is satisfied. And similarly, we add the $view\_point$'s $viewpoint\_key$ to the $global\_triangle\_dictionary$'s corresponding triangle entity.

Finally, when all the triangle faces have been processed in a rendered image, the populated $view\_point$ is added to the $global\_viewpoint\_dictionary$. The steps described so far are iterated for all triangles in the 3D mesh model. Post iterations, the initial set of viewpoints is contained in $global\_viewpoint\_dictionary$. There are situations where the triangles are visible from a viewpoint in the rendered image. Still, the viewing angle



(a) Rendered Image 1

(b) Rendered Image 2

(c) Rendered Image 3

(d) Rendered Image 4

Figure 4.6: 2D Rendered Images for Initial View Point Generation

made by the viewing direction from the viewpoint with the normals of the triangles is not within the viewing angle threshold limits. As a result, they are not included in the list of visible triangles. The viewing threshold ensures the quality of the view from a particular viewpoint. In some situations, a triangle is also partly visible but is included in a viewpoint's list of visible triangles. Considering the small size of the triangles, partial visibility of the triangle does not impact the coverage map significantly.

4.1.2 View Points Selection

After generating an initial set of viewpoints, *global_viewpoint_dictionary* is obtained that contains more than the required viewpoints to cover the whole object. To retrieve the essential viewpoints, pruning is performed on the initial set based on the criteria that the more triangles a viewpoint cover, the higher the precedence it has to be selected. The greedy approach in selecting the viewpoints ensures the inclusion of necessary viewpoints only.

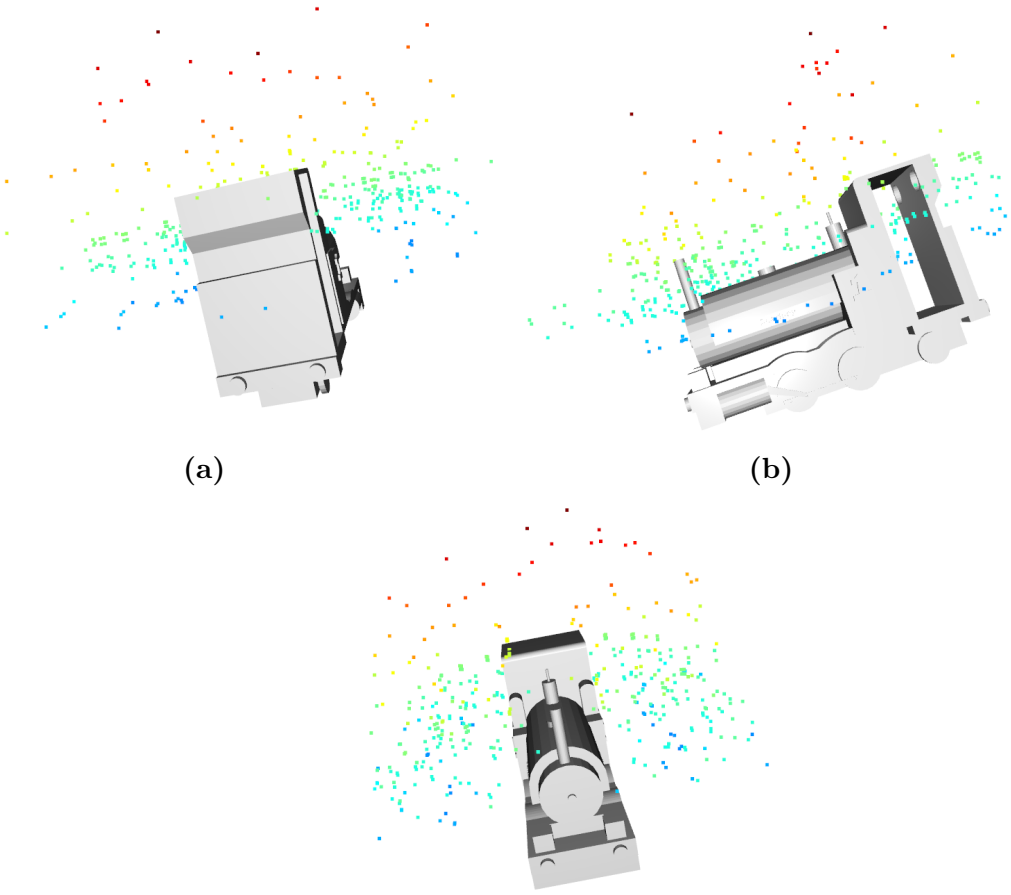
Algorithm 3 describes + the initial set of viewpoints. *global _ viewpoint _ dictionary* is copied to *global _ viewpoint _ dictionary _ pruning* and sorted in descending order based on the number of triangles a viewpoint can view.

Iteration is performed until *global _ viewpoint _ dictionary _ pruning* is empty. The first item in the *global _ viewpoint _ dictionary _ pruning* (essentially the viewpoint with the most number of triangles visible) is selected as *viewPoint*. And an empty list *triangles _ to _ remove* is defined. The list of triangles in *viewPoint* is sorted in descending order of the area of the triangles.

Post sorting for every triangle in *viewPoint*.triangles, it is checked whether the *triangle* has already been assigned a viewpoint. If not, the *viewPoint* is assigned to the *triangle* in *global _ viewpoint _ dictionary*, and consequently, the triangle is added to the *triangles _ to _ remove* list.

Once all the triangles are processed in the *viewPoint*.triangles list, the *viewPoint* is removed from *global _ viewpoint _ dictionary _ pruning*. And the triangles already assigned a viewpoint (the triangles in *triangles _ to _ remove* list) are removed from all the viewpoints in *global _ viewpoint _ dictionary _ pruning* if present. Then the *global _ viewpoint _ dictionary _ pruning* is again sorted in descending order of the number of triangles in a viewpoint.

After the *global _ viewpoint _ dictionary _ pruning* has been completely processed, *selected _ view _ Points _ pruned* is populated with all the viewpoints from *global _*



(a) **(b)**

(c)

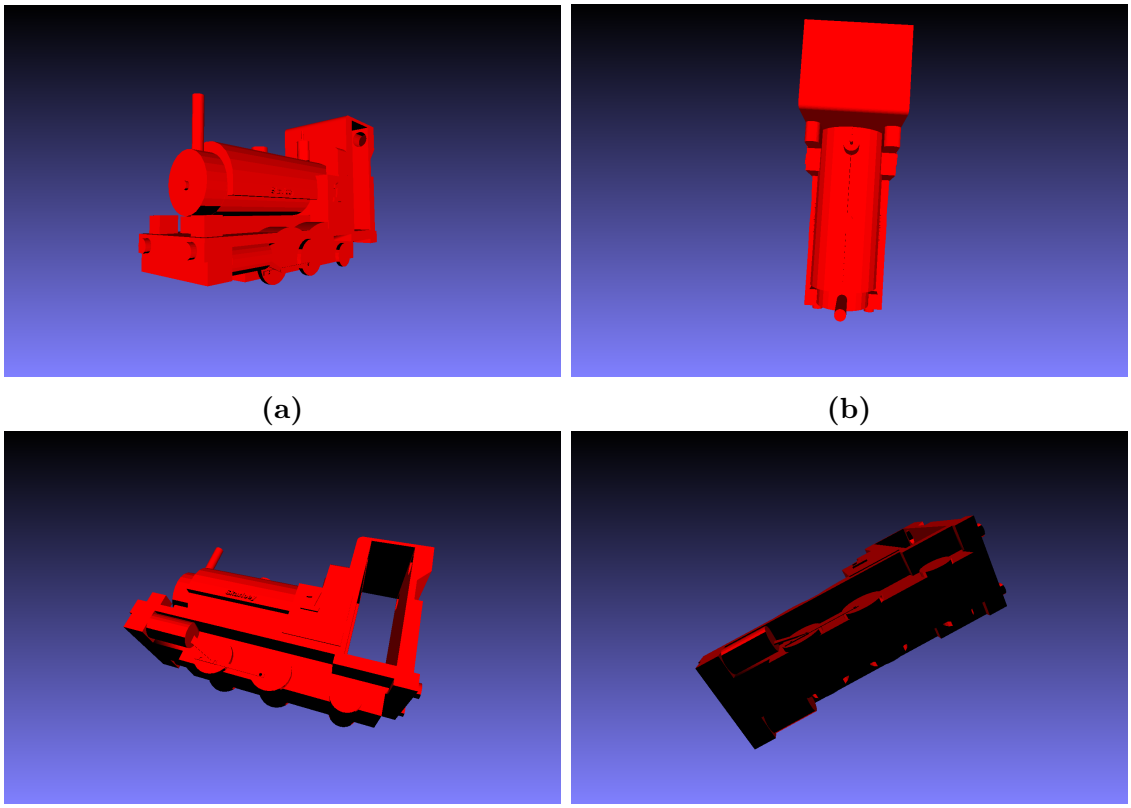
Figure 4.7: Selected View Points around the Object

Algorithm 3: Selection of ViewPoints

```
Data: global_viewpoint_dictionary , global_triangle_dictionary
global_viewpoint_dictionary_pruning = global_viewpoint_dictionary.copy()
sorted(global_viewpoint_dictionary_pruning, key = numberOfTriangles, reverse)
while global_viewpoint_dictionary_pruning do
    triangles_to_remove = [ ]
    viewPoint_key = list(global_viewpoint_dictionary_pruning.keys())[0]
    viewPoint = global_viewpoint_dictionary_pruning[viewPoint_key]
    sorted(viewPoint.triangles, key = area, reverse = True)
    for triangle in viewPoint.triangles do
        if global_triangle_dictionary[triangle].selected_view_point == None then
            global_triangle_dictionary[triangle].selected_view_point =
             viewPoint_key
            triangles_to_remove.add(triangle)
        end
    end
    global_viewpoint_dictionary_pruning.pop(viewPoint)
    remove_triangles(triangles_to_remove) # Removing the triangles associated
     with this viewpoint from the other viewpoints
    sorted(global_viewpoint_dictionary_pruning, key = numberOfTriangles,
     reverse)
end
selected_view_Points_keys_pruned = [ ]
for triangle in global_triangle_dictionary do
    selected_view_Points_keys_pruned.add(triangle.selected_view_point)
end
selected_view_Points_pruned = [ ]
for key in selected_view_Points_keys_pruned do
    selected_view_Points_pruned.add(global_viewpoint_dictionary[key])
end
```

triangle _ dictionary where *selected _ view _ point* is not **NONE**. Figure 4.7 shows the exported viewpoints as a Point Cloud. In the Figure, the different colors of the points depict different altitude levels, with red as the highest altitude.

In Figure 4.8, the visible triangles are indicated by red markings. The black color markings denote that the triangle is not visible from any viewpoint. The bottom of the train can not be covered due to drone position constraints. Similarly, in Figure 4.8c, the inner side of the roof of the operator's cabin is marked black. The reason behind this is the orientation of the surface's normal. As the normals of all the triangles belonging to the inner surface of the roof are pointing vertically downwards, there are no possible viewpoints locations from where the surface can be seen. There can be some blind spots visible on the upper body of the object, that is due it obstructions to the viewpoint from the protrusions in the object. The visibility of the triangles is binary, and if the triangle is partially visible,



(a) **(b)**

(c) **(d)**

Figure 4.8: Visible Triangles from the Selected View Points (red color denotes that the triangle is visible, and black means that the triangle is not visible from any of the viewpoints)

it is considered visible completely in the coverage map. As a result, some artifacts in the visualization of the maps are observed. But the area of an individual triangle is very small, and the partial visibility is not significant.

4.2 Trajectory Generation

The essential viewpoints required to cover the whole object were generated in the previous section. There is a need to generate a trajectory for the UAV that passes through all these viewpoints. This problem is a form of Travelling Salesman Problem (as described in Section 2.3.3) with a variation of multiple goal orientation in the approach. This becomes an NP-Complete problem, where an optimal solution could be found in a polynomial time, but the complexity increases with the rising number of viewpoints to be visited. Thus we opted for a Greedy approach to this problem. If started from a viewpoint, the sequence of viewpoints next in line to be visited based on the mutual distance between the viewpoints is generated. A trajectory using line segments joining the consecutive viewpoints can be generated (as stated in Dorméus et al. [Dor]). The idea of the trajectory of viewpoints covering the object is visualized in Figure 4.9.

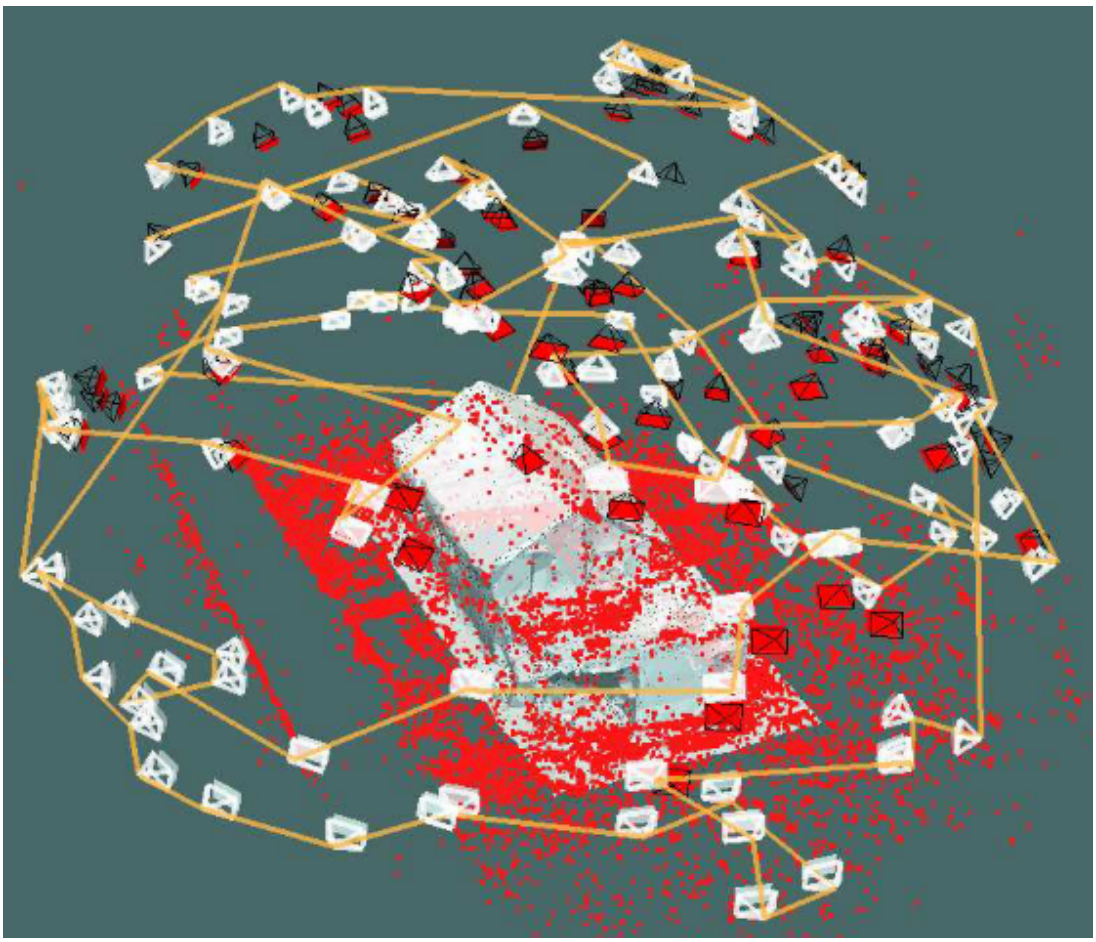


Figure 4.9: Camera View Points Trajectory covering an Object [HWZ]

4.2.1 Sequencing the Generated ViewPoints

Sequencing the viewpoints is essential to generate the trajectory for all the viewpoints. Algorithm 4 describes the steps to generate a trajectory using Greedy Approach.

Algorithm 4: Generation of the Order of the ViewPoints

Input: *initial_Location*
Data: *selected_view_points*, **D**
Output: *ordered_visited_view_points*

```
1 current_Location ← initial_Location
2 ordered_visited_view_points.add(current_Location)
3 while len(ordered_visited_view_points) ≠ len(selected_view_points) do
4     D[current_Location] ← sorted(D[current_Location], key=distance) # D matrix
        contains corresponding distances between Selected View Points
5     for viewPoint in D[current_Location] do
6         if viewPoint not in ordered_visited_view_points and not
            isblocked(start=current_Location, end=viewPoint) then
7             current_Location ← viewPoint
8             ordered_visited_view_points.add(current_Location)
9             break
```

In Algorithm 4, an initial location of the drone (the viewpoint to start from) is given as an input. Initially, a distance matrix **D** is constructed by computing all mutual euclidean distances amongst the viewpoints. We start with assigning the *current _ Location* with the *initial _ Location*. The iteration is initiated by sorting the distances of *current _ Location* with other viewpoints in ascending order of distance value. The iterations would last until the length of *ordered _ visited _ view _ points* is equal to *selected _ view _ points* (essentially iterating over all selected viewpoints). As the viewpoint coordinate signified by *initial _ Location* is already visited, it is added to *ordered _ visited _ view _ points* before initiating the main iteration. After sorting and initial definition and population of *ordered _ visited _ view _ points*, iteration is done over all the viewpoints according to ascending distance from *current _ Location*. A viewpoint is added to *ordered _ visited _ view _ points* if the viewpoint is already not present in the list *ordered _ visited _ view _ points* and if there is no obstacle in between the *current_Location* and the iterator viewpoint, and that viewpoint is assigned to *current _ Location*. The iteration runs until the last viewpoint is reached. Figure 4.10 shows the trajectory generated with a random starting viewpoint. The ordered viewpoints are connected using line segments.

As shown in Fig 4.11, the generated line-segment trajectory covers the whole object without violating the ground tolerance.

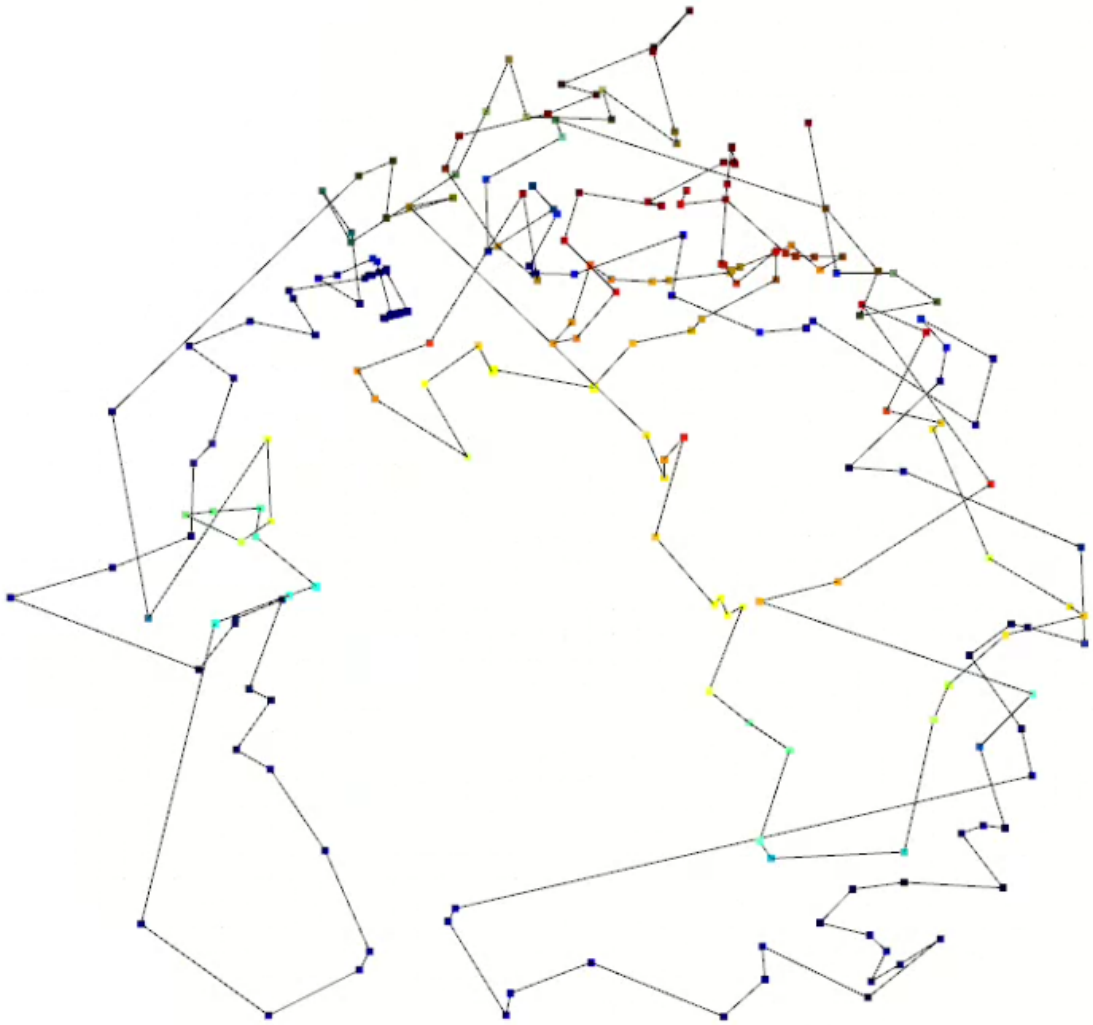


Figure 4.10: Trajectory Generation using Line Segments

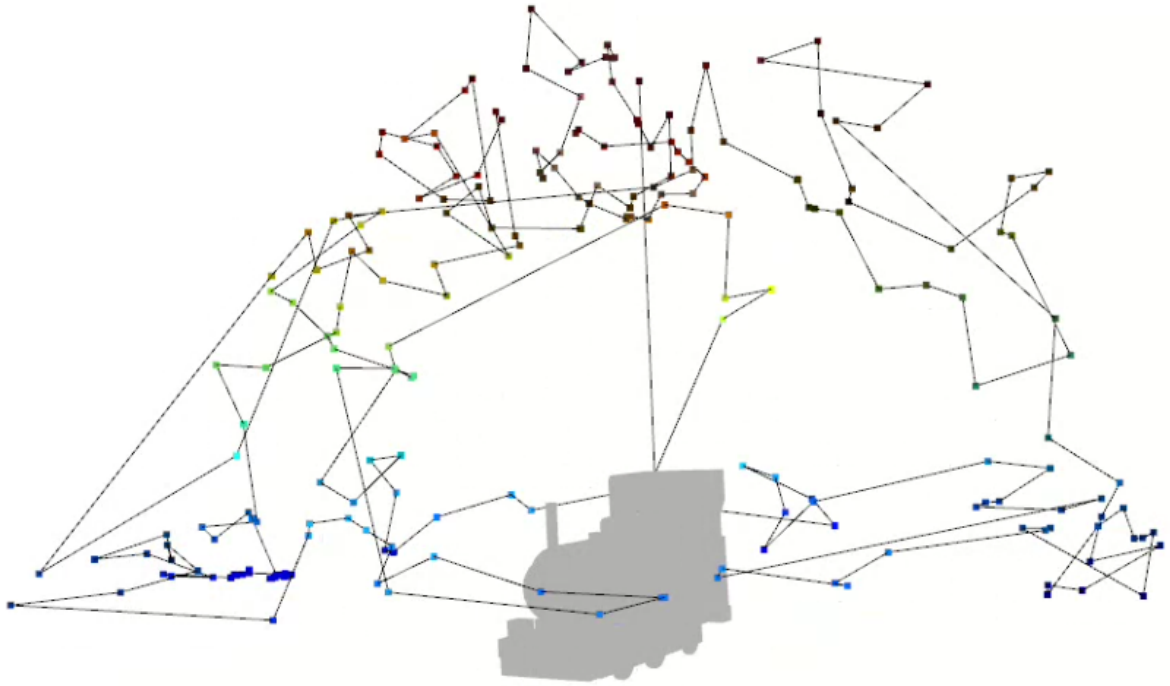


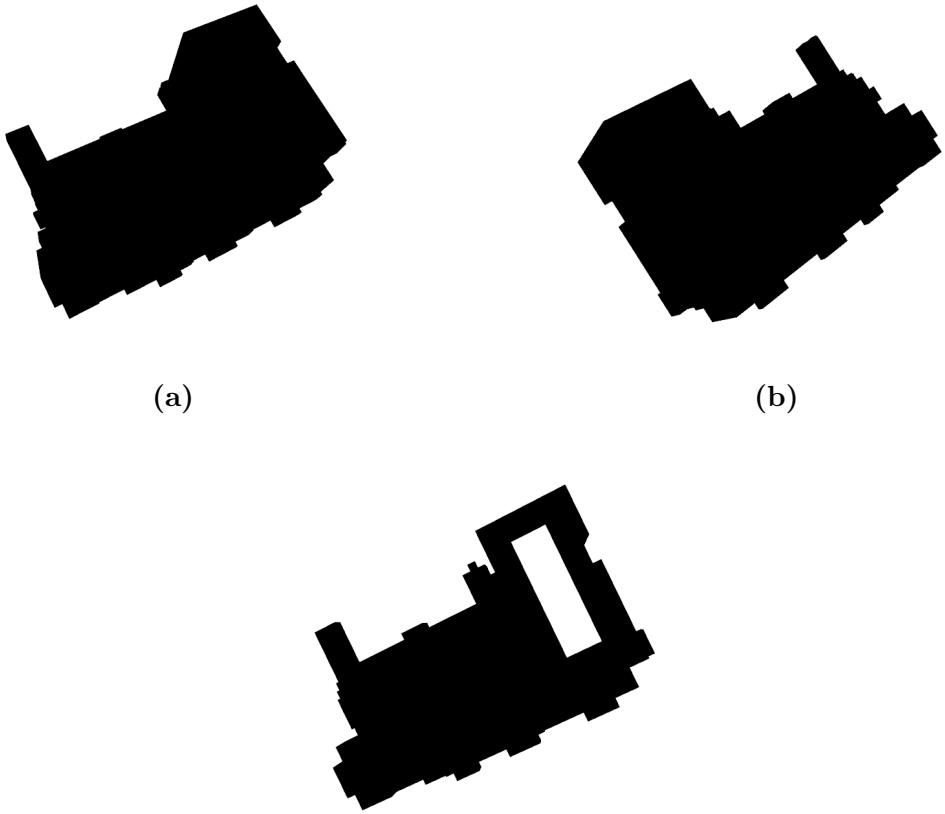
Figure 4.11: Line-Segment Trajectory around the Object

4.2.2 Obstacle Management using Voxelisation

For Obstacle management, voxels of the object (as shown in Figure 4.12) are used to detect whether there is any intersection between the line segment joining the viewpoints and any of the object (obstacle) voxels. To get the points of intersection with voxels, orthogonal plane equations within the bounding limits of the object/obstacle are defined. Algorithm 5 shows the steps after we have computed the orthogonal plane equations of the object's voxel grid.

In Algorithm 5, the intersection points of the line segment joining two viewpoints and all the voxel grid planes are computed by brute force. Iteration starts with a *plane_eq* in *planes*, and the intersection point of a line segment between the *start* viewpoint and the *end* viewpoint and the plane defined by *plane_eq* is computed. Once the intersection point is computed as *intersection_point*, the voxel pertaining to the coordinate of the point is fetched if it is within the bounding box of the voxel grid. True is returned if a match is found. Otherwise, the iteration for all the plane equations in *planes* continues. If no matching voxels are found, the path is declared obstacle free by returning false.

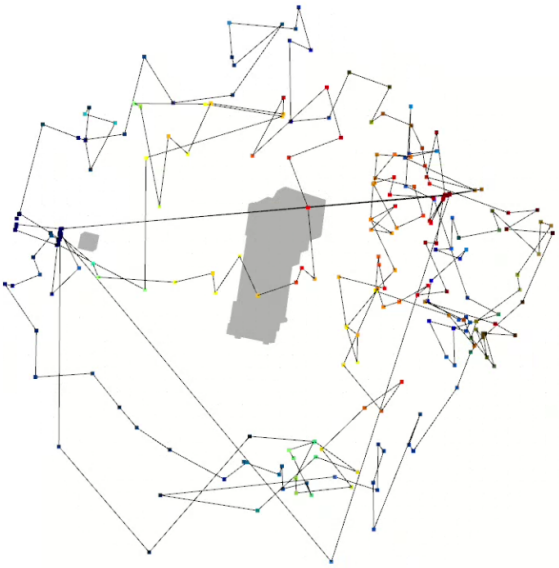
The obstacle management is visualized by adding a dummy obstacle (small cube) in the proximity of the viewpoints and then generating the line-segment trajectory. The obstacle avoidance by the trajectory generation algorithm is observed in Figure 4.13.



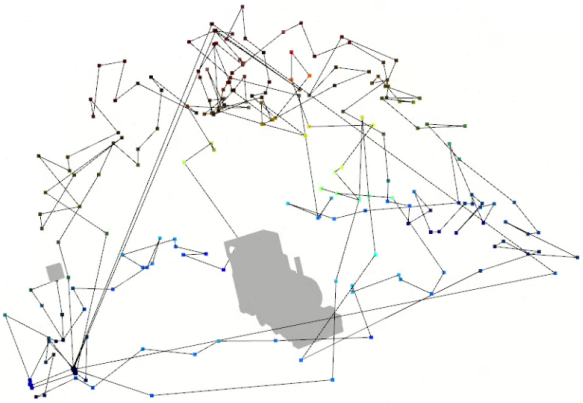
(a) **(b)**

(c)

Figure 4.12: Voxelised Model



(a)



(b)

Figure 4.13: Line Segment Trajectory with Dummy Obstacle

Algorithm 5: Obstacle Management using Voxelisation

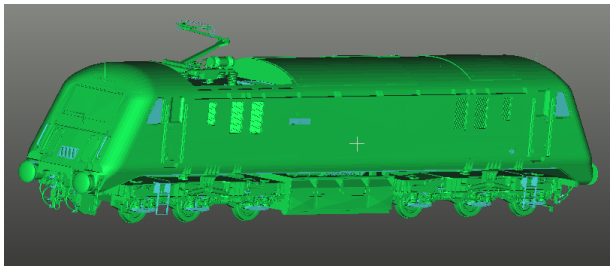
```
1  Function isblocked(start, end, object_bb, voxel_Grid, planes):
       Input: The start point, endpoint, object bounding box, Orthogonal plane
              equations within the bounding box constraints and Voxel grid of the
              environment
       Output: True if the path is blocked; otherwise, False
2      for plane_eq in planes do
3          voxel_found = None
4          intersection_point = get_intersection_point(plane_eq, start, end)
5          if is_in_bounding_box(intersection_point, object_bb) then
6              voxel_found = voxel_Grid.get_voxel(intersection_point)
7              matches = all(voxel_found in obstacle_voxels)
8              if (len(matches) > 0) then
9                  return True
10             end
11         end
12     end
13     return False
```

5 Results and Analysis

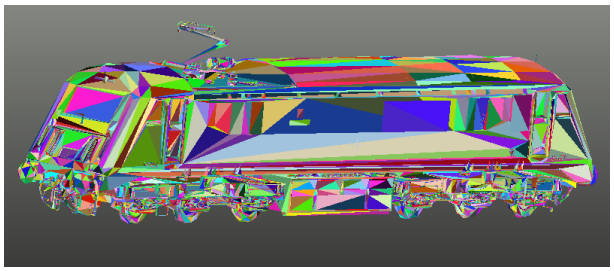
The algorithms described in the previous chapter were implemented on the 3D Mesh Model mentioned in the Problem Statement of the Thesis. The results are produced using the decimated model described in Chapter 3, under Data (3.1). This chapter has two components, the results from the Coverage Calculation algorithm and the results from the Trajectory Generation algorithm.

5.1 Coverage Calculation

The 3D mesh model (decimated version) is tripled to generate random and unique color and color encodings for the triangular faces of the model, as shown in Figure 5.1.



(a) Original Model

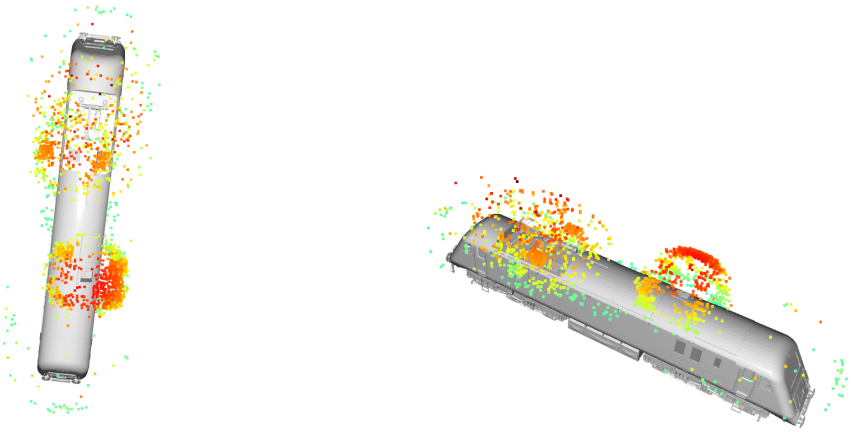


(b) Tripled Model

Figure 5.1: Tripling of the 3D Train Mesh Model

The initial viewpoints were generated for Train Mesh Model, and the selection of viewpoints was performed on the initial set of viewpoints. The initial test revealed a problem with the triangles having a surface area close to zero but having weightage in generating the viewpoints. As a result, over 9107 viewpoints were selected, which also caused clustering over certain parts of the train.

In Figure 5.2, it can be seen that there is more than necessary clustering of viewpoints over two regions of the train. While analyzing the trained model, it can be seen the two



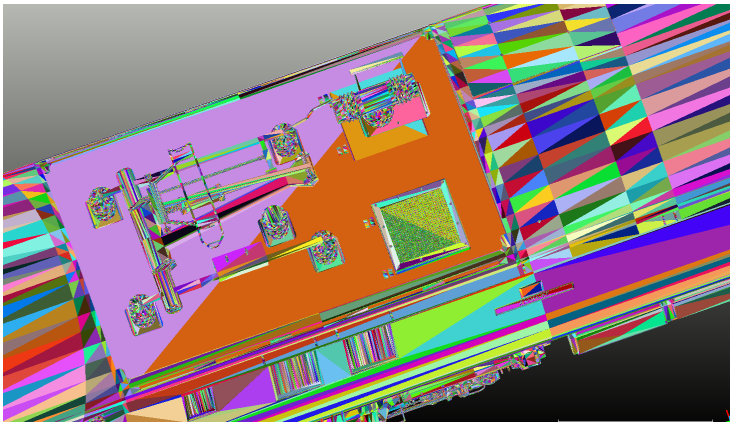
(a) **(b)**

Figure 5.2: Clustering of View Points over small triangle area regions

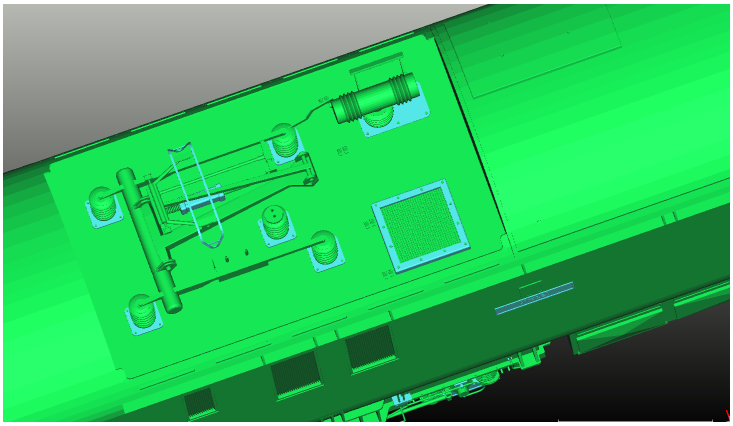
regions contain a large set of triangles that have an area close to zero (as shown in Figure 5.3, due to the presence of a mesh duct on the train roof there was a significantly huge number of triangles with very small surface area as shown in Figure 5.3c) but are having equal weightage in generating viewpoints.

Due to this issue, more than necessary viewpoints are generated (the algorithm considers the triangle with smaller areas with the same precedence as a triangle with a significant area). To resolve this issue, sampling based on the triangle area (as we cannot neglect all triangles with a small area, this might induce some avoidance of critical ROIs) was implemented. The sampling algorithm 6 computes the probability weights for selecting a triangle for the initial view. The higher normalized weight governs a higher probability that the triangle would be considered for generating the initial set of viewpoints. This measure encourages the algorithm to give precedence to the viewpoints generated from triangles with a larger surface area and not ignore the features represented by triangles with a small surface area. A probability for considering triangles with a smaller area as *small _ area _ probability* is set. The *triangle _ in _ consideration _ sampling _ weight* can be set as *small _ area _ probability* if the area of the triangle is very close to zero (as shown in the algorithm 6 line 4). The probability weights could be used as a constraint in algorithm 2, where a triangle could be skipped randomly based on the probability weights generated by function in algorithm 6.

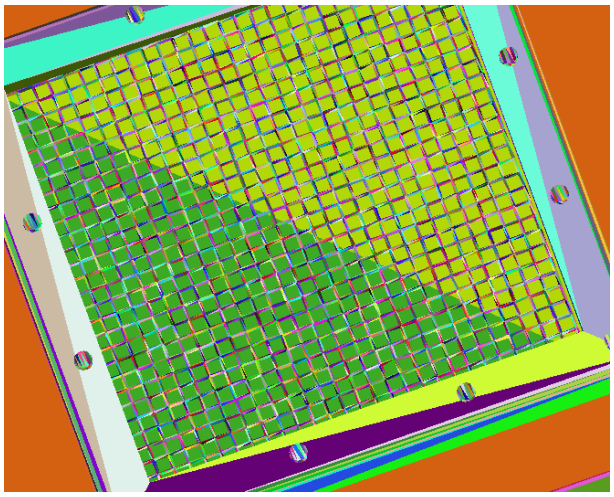
The updated algorithm implemented on the Decimated Model (80097 triangular faces) produces the results shown in Figure 5.4. Out of the initially generated viewpoints, 2568 viewpoints were selected. The ground tolerance is assumed to be 5 units for different experiments.



(a) Tripled Model

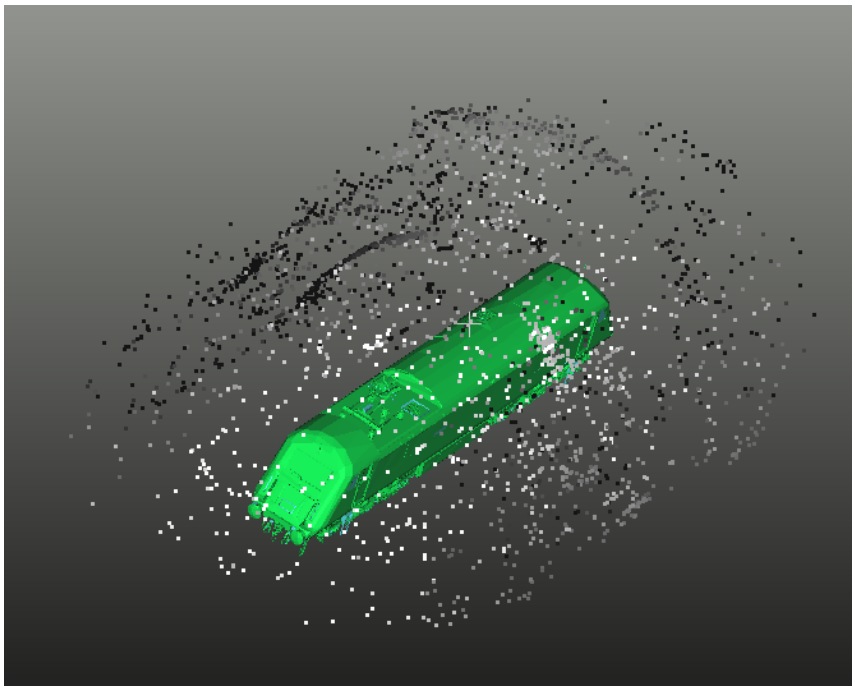


(b) Real Model

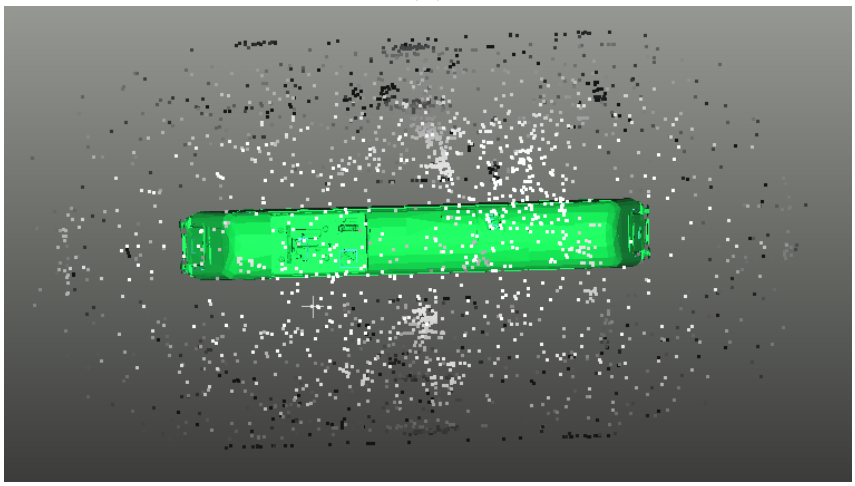


(c) Zoomed-in View of the mesh Duct

Figure 5.3: The meshed Duct on the Train's roof having small triangles



(a)



(b)

Figure 5.4: Selected View Points after Sampling implementation

Algorithm 6: Samling Module for Initial View Points Generation

Data: *Normalized_triangle_area_list*

small_area_probability = probability for sampling a triangle with a triangle face area close to zero

Function `sampling_module(`*triangle*`):`

  Input: Trianlge key from the list of triangles

  Output: Probability for a triangle to be considered

  triangle_in_consideration_sampling_weight = *Normalized_triangle_area_list*[triangle]

  if *triangle_in_consideration_sampling_weight is close to Zero* **then**

    triangle_in_consideration_sampling_weight = *small_area_probability* `# Weights for View Point selection area based on the area of the triangle`

  end

  return *triangle_in_consideration_sampling_weight*

The selection of viewpoints is designed in a way that is greedy in terms of a viewpoint having the most visible viewpoints. Hence the reduction of triangles must reduce with every iteration forward, as shown in Figure 5.5. This shows from approx. 35000 triangles are visible from the selected viewpoints, and while selecting the viewpoints, around 2500 iterations were required. The number of iterations accounts for the number of selected viewpoints.

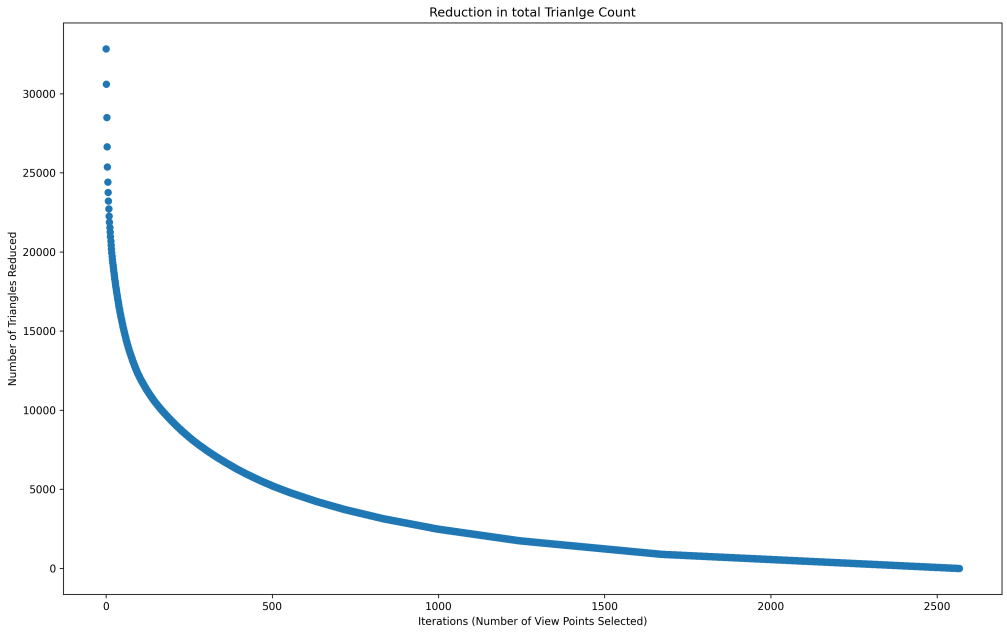


Figure 5.5: Reduction in the number of Triangles

The number of associations of triangles with a viewpoint also decreases with the increasing

number of viewpoints, as shown in Figure 5.6. It can be observed that the vertical axis shows the number of triangles associated with a single iteration, while the horizontal axis represents the number of iterations. It is visible that in the first iteration, more than 2500 triangles were associated with a single viewpoint. Similarly, more than 2250 triangles got associated with the second viewpoint. A greedy approach is followed to maximize the number of triangle associations in the least number of iterations possible. The graph's scale in Figure 5.6 is very high. Hence it cannot show the accurate variation in the number of associations after the steep drop, which is minuscule compared to large values. The variation is on the scale of 100 to 200 triangles, eventually approaching zero associations.

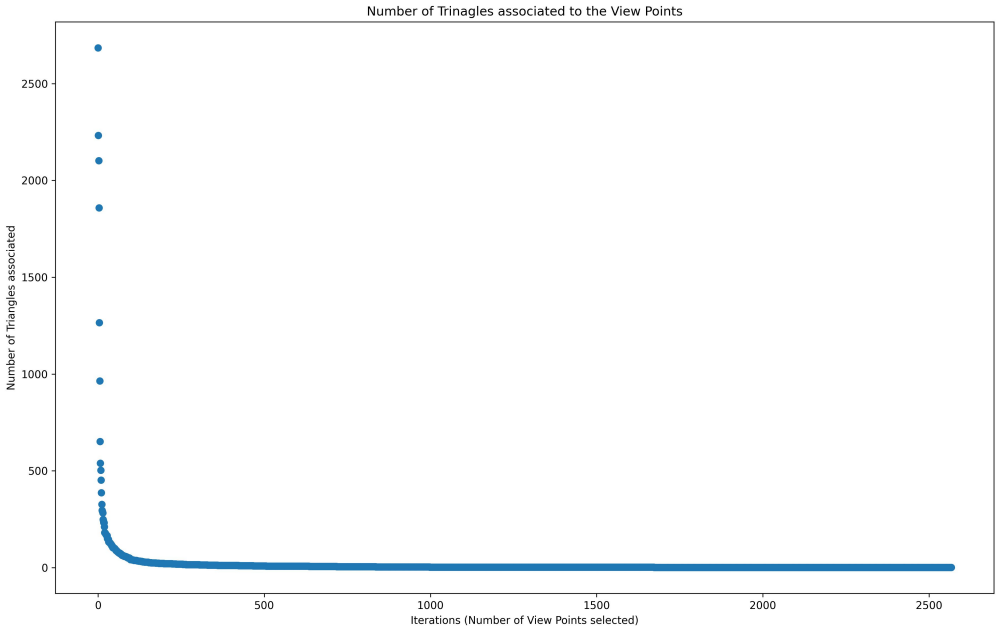
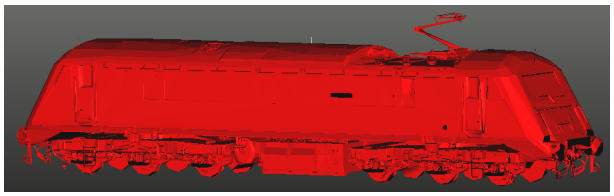


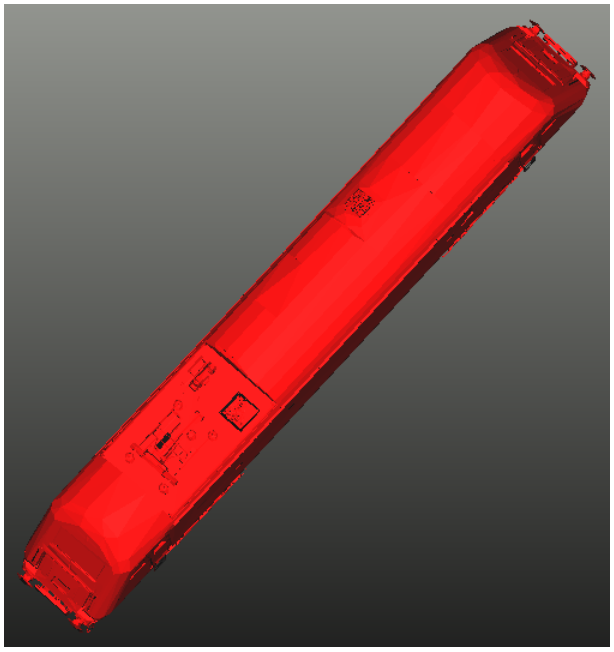
Figure 5.6: Triangle Associations per View Point

Sampling induces a bias in the algorithm to opt for the least number of triangles considered for the viewpoint generation. For this experiment, a probability of 0.5 was chosen for sampling small area triangles, and a ground tolerance limit of 5 was selected. The bias is based on the merit of a triangle (the surface area of the triangle where it is substantial); otherwise, the sampling probability is 0.5. Figure 5.9 shows that more than half of the total triangles, approximately 45000 triangles, were not considered for the viewpoint generation. The non-consideration of a triangle may be due to sampling or the constraint of the ground tolerance limit. Around 35000 triangles were utilized to generate the initial set of viewpoints, and with the further selection process, only around 2500 viewpoints were essential for the coverage of the trained model. Figure 5.7 shows the coverage results

after inducing sampling.



(a) Random Viewing Direction of the Coverage Map



(b) Top View of the Coverage Map

Figure 5.7: Coverage Map of the Decimated Model after implementation of Sampling

Addition of some randomness to the process of generation of the viewpoints, some patches are missing in the coverage map, as shown in Figure 5.8. The missing triangles in the coverage map are because viewpoints generated from the triangles in the vicinity, which passed through the ground tolerance limit constraint and were selected in the sampling module, did not make an angle within the viewing angle threshold with the triangles' normals. As a result, even though they would be visible through any of the viewpoints, due to the angle between the viewpoint's viewing direction and the triangle normal not being within the viewing angle threshold, the triangles were not registered to any of the viewpoints. The other major missing patches close to the bottom of the train are due to the ground tolerance limit of the drone. The viewpoints could not be generated themselves as they lie beneath the limit.

The number of selected viewpoints primarily depends on the ground tolerance limit, distance from the object (hyper-parameter α), and the sampling probability. The ground tolerance limit and α are specific to the drone's flight parameters and the drone's camera's

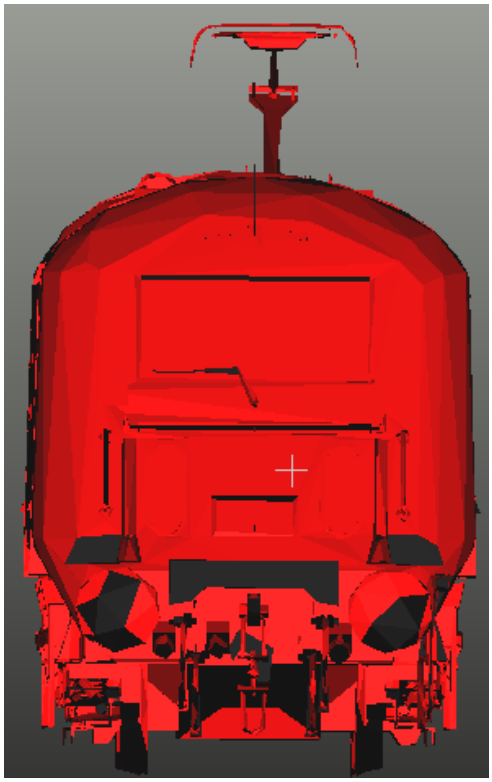


Figure 5.8: Random missing patches in the Coverage Map

parameters. The experiments were made on the variation of the sampling probability for small triangles, as the bigger triangles have the probabilities computed from the normalized areas of the triangles.

The first experiment was done by choosing a small triangle area sampling probability of 0.2, and a ground tolerance of 7 was selected. As shown in Figure 5.10, the number of triangles considered greatly reduced from around 35000 triangles to less than 20000 triangles. As a result, the number of selected viewpoints reduced from more than 2500 to less than 1400. However, the nature of the triangle reduction curve and triangles association curve remains similar, having huge associations and reductions at the start of the iteration and eventually decreasing with an increasing number of iterations. Due to such a low probability of accepting the triangles with an area close to zero, the missing triangle patches in the coverage map increase and are significant, as shown in Figure 5.11. The missing patches in the window panes of the trained model are due to very low sampling probability.

The experiment was done with a small area triangle sampling probability of 0.7, keeping the ground tolerance limit the same. The results are shown in Figure 5.13. The increase in the probability of a triangle with a small surface is to be considered for viewpoint generation resulting in a larger set of initial viewpoints. And subsequently, the number of selected viewpoints also increased, as shown in Figure 5.13b, where if the sampling

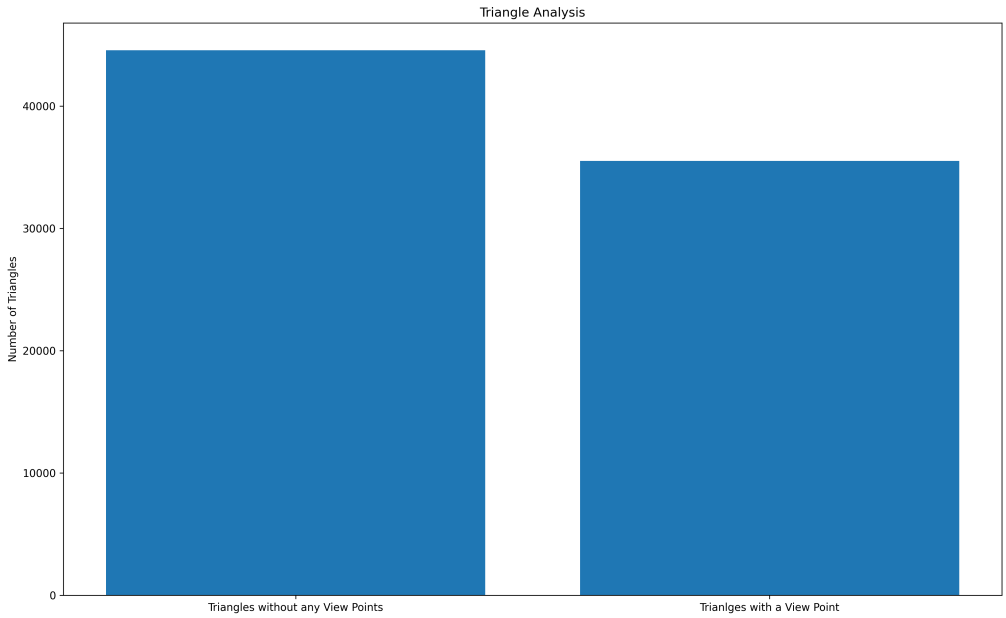
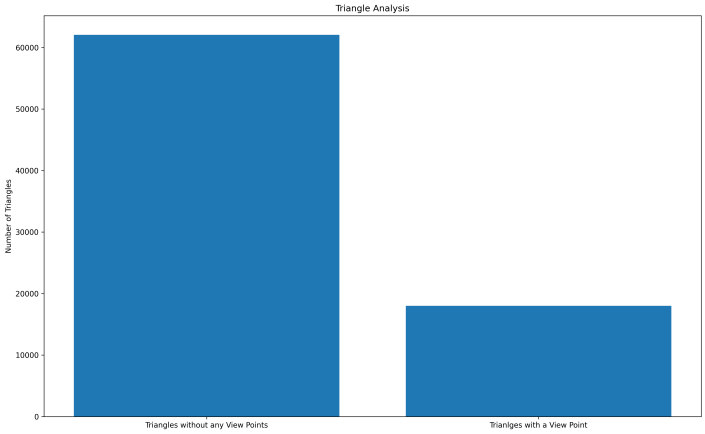


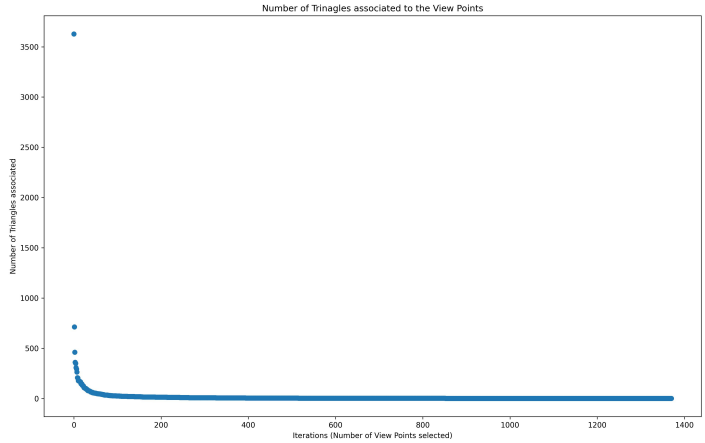
Figure 5.9: Number of Triangles Considered/Not-Considered

probability of 0.2 is considered, less than 1400 viewpoints are obtained (Figure 5.10c). Sampling probability of 0.7 results in more than 2500 viewpoints being selected. However, due to the limit of the viewpoints concerning ground tolerance, there is not much difference in the coverage of triangles observed (as we compare Figure 5.10a and Figure 5.13a). There is only a slight difference between the triangles having a viewpoint if the sampling probability is 0.7 and if triangles having a viewpoint if the sampling probability is 0.2. Of course, the number of triangles having viewpoints would be higher if the sampling probability is higher, but the difference is not substantial. The increased chances of a triangle to generate viewpoints caused some improvements to the issue displayed in Figure 5.11.

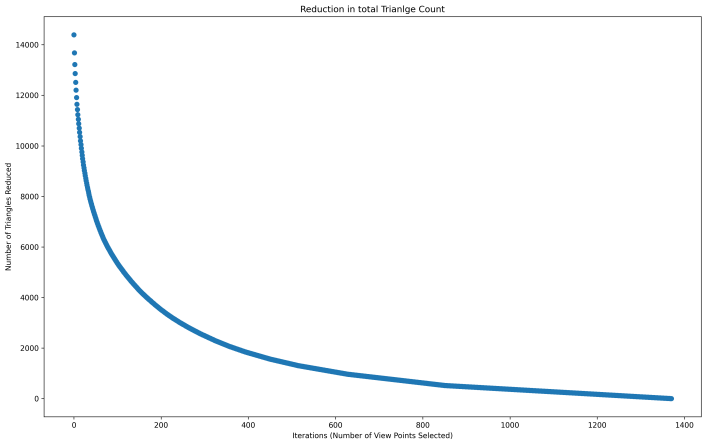
Some of the smaller triangles surrounding the window pane are visible, causing red pigmentation in the black color (the black color's intensity reduces), as shown in Figure 5.12. However, it can be observed that the window panes are primarily black even though there could be some red pigmentations (The pigmentation is due to the coloring scheme of Open3D, where if the sharing vertices of two differently colored triangles exist, then the library tries to find an average color for both the triangles). This is because the window panes are slightly tilted downwards, and due to the higher ground tolerance limit of 7 units, the viewing directions of the viewpoints, how many they may be, cannot make an angle within the threshold angle limit of 45 degrees. This problem gets resolved if we choose a lower ground tolerance limit (as seen in Figure 5.7a, where the ground tolerance



(a) Number of Triangles Considered/Not-Considered



(b) Triangle Associations per View Point



(c) Reduction in the number of Triangles

Figure 5.10: Results for Small Triangle Area Sampling Probability as 0.2

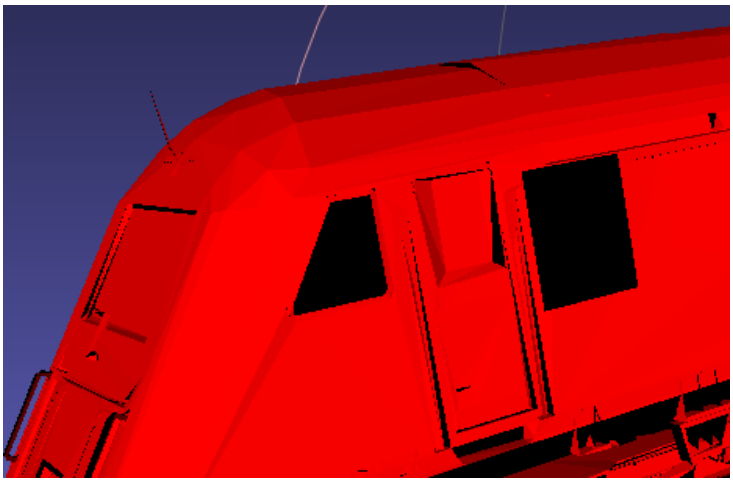


Figure 5.11: Missing patches in the Coverage Map for Sampling Probability of 0.2

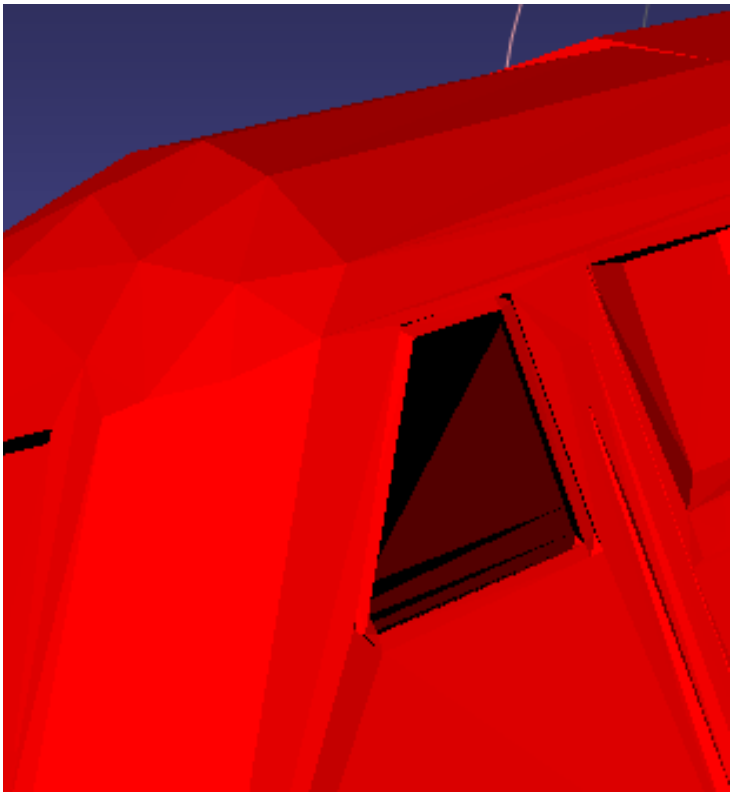
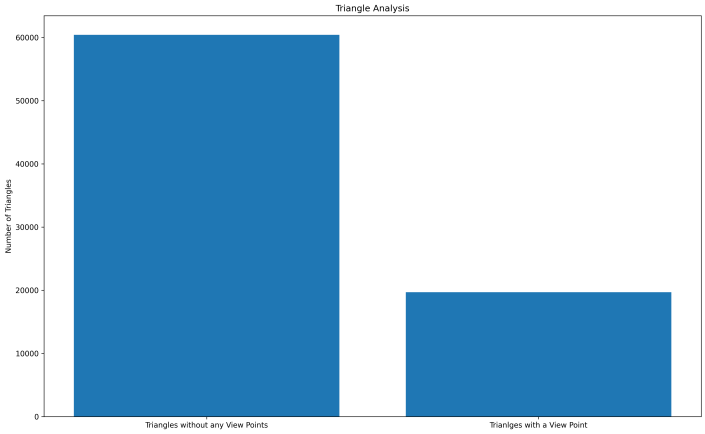


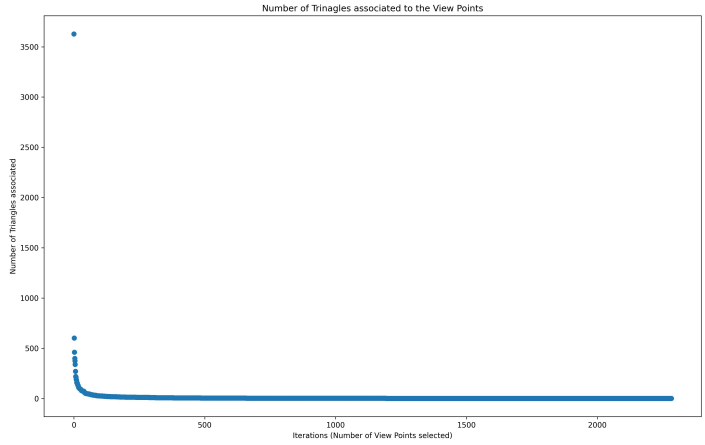
Figure 5.12: Missing patches in the Coverage Map for a Sampling Probability of 0.7

limit is set to 5 units, and the window panes are completely red).

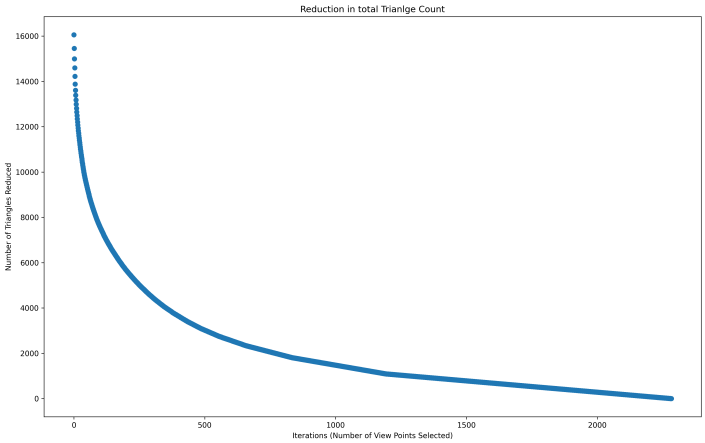
It can be concluded that inducing more randomness to the algorithm in terms of sampling would generally result in fewer viewpoints. But the coverage variation is not significant while doing so, keeping all other parameters constant. Increasing the chances of a triangle generating a viewpoint will only add to the quality of the coverage map, as observed previously that most of the coverage could be taken care of with viewpoints less than 1400 in number. Increasing the acceptance probability just improved the coverage map very slightly with an expense of a substantially higher number of viewpoints (it was observed that with higher sampling probabilities, there were many viewpoints with less than 10 triangles visible).



(a) Number of Triangles Considered/Not-Considered



(b) Triangle Associations per View Point



(c) Reduction in the number of Triangles

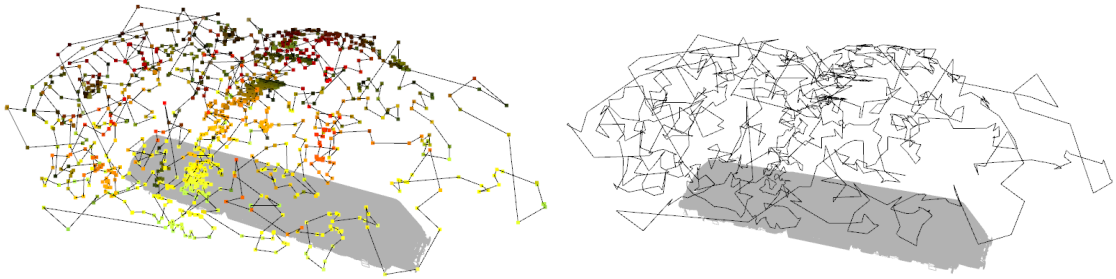
Figure 5.13: Results for Small Triangle Area Sampling Probability as 0.7

5.2 Trajectory Generation

The voxel grid is generated for the decimated model of the train model mesh. The voxel size for this model is chosen as 0.2 units. To voxelize the entire object, 12053 voxels were generated. Figure 5.14 shows the voxel grid for the decimated model.

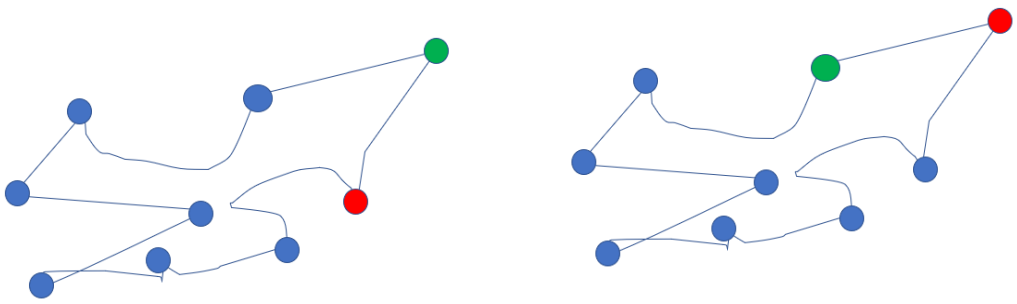
Figure 5.14: Voxelisation of the 3D Mesh Model

The trajectory was generated after sequencing the viewpoints from the chosen Start viewpoint (as shown in Figure 5.15). It is observed that all the viewpoints are covered from the line-segment trajectory after sequencing the viewpoints. Figure 5.17 visualizes the start and endpoints.



(a) Visualization with View Points **(b)** Visualization without View Points

Figure 5.15: Trajectory Generation from a Random Initial View Point



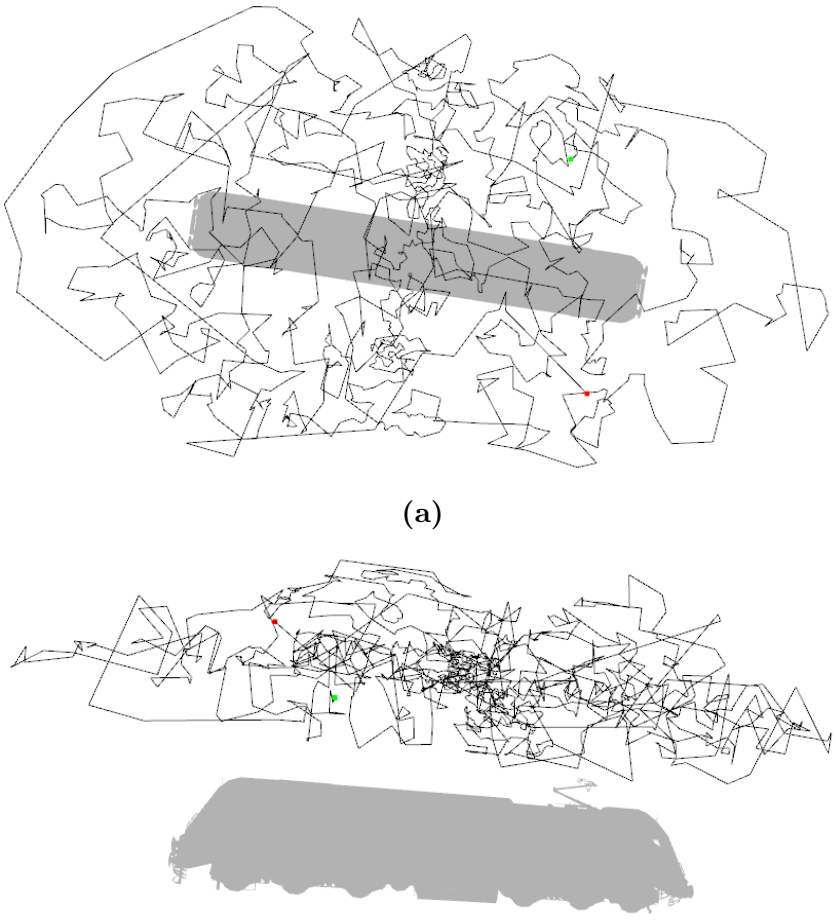
(a) Start View Point (Green) and End View Point (Red) **(b)** Start View Point (Green) and End View Point (Red)

Figure 5.16: Illustration showing Loop Formation with absence of Obstacles

The results show that the trajectory is generated for the decimated model, the obstacle management was not required as all the viewpoints are situated at a certain altitude (as shown in Figure 5.17b), where there is no component of the object in between any of the two random viewpoints. The consistency of the solution was analyzed by computing the distance covered in a generated trajectory by sampling random viewpoints from the set of selected viewpoints and setting them as starting viewpoints for the Trajectory Generation Algorithm. In Figure 5.18, it can be seen that the distance covered for all the sampled viewpoints is the same represented by a single bar in the graph. The graph describes that around 682 viewpoints were sampled to be the starting point of the trajectory, generating the trajectory with a covering distance of values ranging between 821.8 to 822 units. The consistency in the value of the distance traveled is due to the absence of any obstacle between the possible paths in the viewpoint. The Distance Matrix **D** contains the distances between all consecutive viewpoints, which in turn is a symmetric matrix, causing the consistency in the additive values per column. This causes a loop formation, where the start and end points become consecutive. And distance computed is the length of the loop for any starting position. As explained in Figure 5.16, if (as shown in Figure 5.16a) there is a set of sequenced viewpoints, with the Start viewpoint and the End View Point, the distance covered is the same if the next consecutive viewpoint becomes the Start View Point (as shown in Fig 5.16b). For every case, the distance covered will be the length of the curve.

The distance variation covered by a trajectory in case an obstacle exists in the proximity of the selected viewpoints can be seen in Figure 5.19. In the case of the trajectory generated for the Toy Train model, the viewpoints were in proximity to the obstacle (the object and the dummy cube). This causes no loop formation due to skipping in the scheme of viewpoints based on the distance values in the Distance Matrix **D** when an obstacle is encountered. In this case, the symmetry of the Distance Matrix **D** does not play a role. Although the variation in the distance covered is insignificant (as shown in Figure 5.19, the variation is from 200 units to 220 units), the distance value is inconsistent.

On the contrary, when the trajectories were generated for the Toy Train Model without implementing the Obstacle Management module, consistency in the distance value was achieved (as shown in Figure 5.20). Due to no obstacle management implemented, it can be shown that all 220 viewpoints have the same trajectory distance of approximately 216 units, which is entirely different from the results obtained in Figure 5.19 where there are obstacles in the generated trajectory path. This scenario is what is being observed in the case of the decimated model (as shown in Figure 5.18) where there are no obstacles effectively.



(a)

(b)

Figure 5.17: Trajectory with Start Point (Green) and End Point (Red)

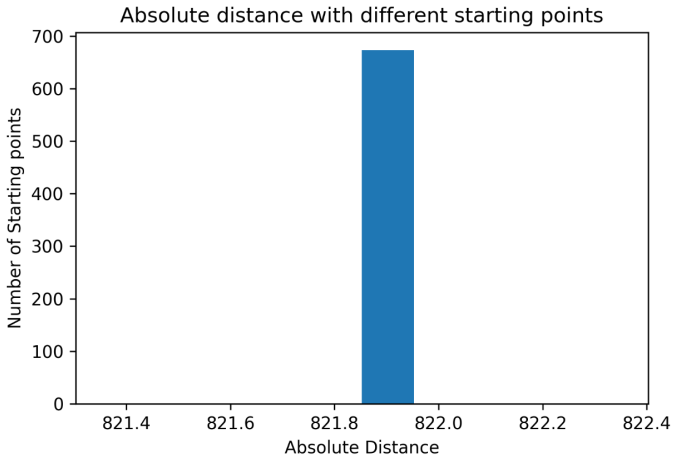


Figure 5.18: Distance covered by the Trajectory of Decimated Model (682 viewpoints sampled for trajectory generation have the same trajectory distance of 821.9 units because no obstacle was found)

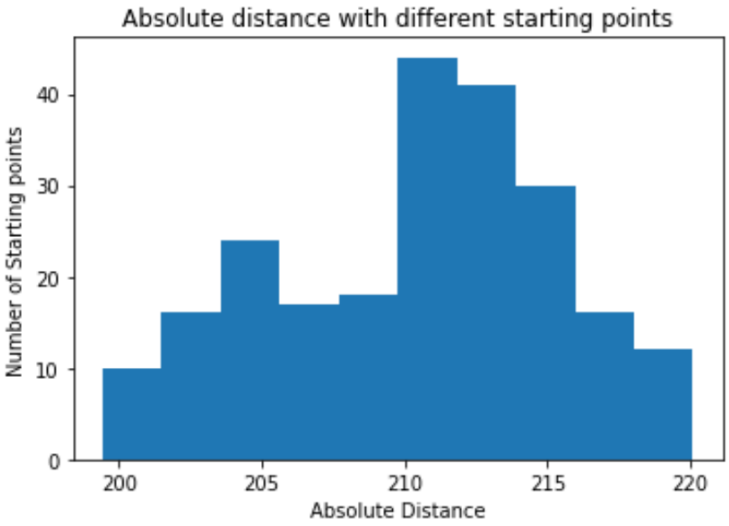


Figure 5.19: Distance covered by the Trajectory of Toy Train Model with Obstacles in the Trajectory Path

This can be concluded from the above results, if there is a sufficient ground tolerance for the selected viewpoints that gives a good coverage map, then no matter from which viewpoint trajectory generation starts, the distance covered following a generated trajectory would be consistent, assuming that there is no other obstacle apart from the object itself. The variation would be observed if an obstacle is introduced to the environment or the ground tolerance limit is low enough such that some of the selected viewpoints are in proximity of the object (proximity to the object increases the chances of possible line-segment intersecting part of the object).

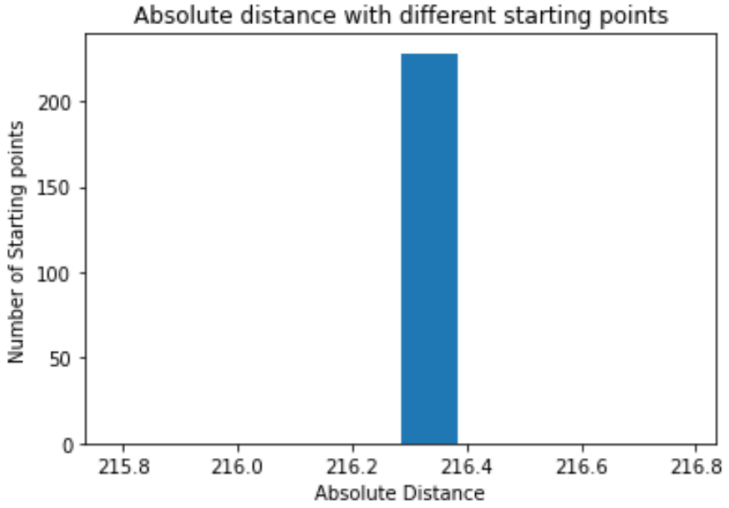


Figure 5.20: Distance covered by the Trajectory of Toy Train Model (no Obstacle Management)

6 Conclusion

In the previous chapters, we discussed the algorithms and the results for generating viewpoints and planning the corresponding trajectory. The thesis concludes with a summary of results, addressing individual research questions and an outlook for future work.

6.1 Summary

The overall task of the thesis was to generate a dynamic trajectory for a drone from a dynamic start point. The trajectory must ensure full coverage of all ROIs. The thesis focuses on one component of the entire Rail Environment. In the following, the defined research questions are discussed:

1. **Is there a possibility to generate a trajectory that ensures the coverage of all ROIs considering a ground tolerance of how close a drone can fly near the ground?**

   This work shows that a feasible trajectory can be generated to ensure all the ROIs' coverage. The trajectory's length depends entirely on the number of viewpoints generated. The generation of viewpoints is based on three hyper-parameters. The distance from the object or the focal length depends entirely on the drone camera's configuration. The ground tolerance of the drone is a feature specification of the drone construction and the industry safety standards. We saw the variation in the sampling probability, causing different results. The computation of a minimum number of viewpoints converges after the introduction of sampling. A higher probability of sampling only increases the number of viewpoints with less improvement in the coverage. Multiple iterations are required to find the sub-optimal sampling ratio.

2. **Is there any efficient way to generate the shortest trajectory visiting all the essential viewpoints?**

   We followed a heuristic approach to the Travelling Salesman Problem with multiple goals into consideration. The heuristics were defined as the distance matrix for all the viewpoints. The correlation between the viewpoints was expressed in terms of distances among the viewpoints. The heuristics were followed in a greedy approach. This produces a sub-optimal trajectory. Global Optimality is not guaranteed. However, the exact method ensures global optimality, but the computational

time increases exponentially. We saw the effect of obstacles in the generation of trajectory. While if no obstacle is found, the trajectories generated for each viewpoint are consistent. The trajectory generation can be done in real-time using a greedy approach. This gives us a very good trade-off between optimality and computational time.

3. **How are the obstacles managed when generating the trajectory?**

   While there are many techniques to manage obstacles in real-time, if we know the environment before the computation, voxelization and bypassing the voxels containing the obstacle content is the best option. In this case, the obstacle environment could be extended to the whole global environment, where many obstacles apart from the object itself could be accommodated. The voxel size governs how sensitive the algorithm is to the obstacle. If the voxel size is big, many features of the object would be lost in discretization. This boils down to adding boundary conditions over the object. This would lead to the spaces where the viewer could exist but would not be considered due to the inflated boundary conditions.

6.2 Future Work

Coverage Path Planning has many more variations. The method discussed in the thesis is primarily offline path planning. Here we assume the environment is known, and no stochastic behavior is accommodated. At the same time, the method used to generate the viewpoints ensures the coverage of all the critical regions of interest. This work can be extended to online trajectory planning. While knowing necessary viewpoints as keyframes, Reinforcement Learning can be implemented to train a model to find the most appropriate heuristics to follow to obtain the trajectory. This work focuses on exploiting the knowledge base instead of exploring it. If Reinforcement Learning is implemented, the model could choose between exploration and exploitation. This way, a more versatile system can be developed that satisfies Rail Industry standards and can be used for a variety of models without doing repetitive computations. Online Coverage Path Planning could be more feasible in dynamic environment situations where there is variation in the obstacles.

Moreover, a real-world demonstrator would be the next extension to this work, which would help in running simulations and training a smart model to make intelligent decisions for the maneuvers of the drone. The next step would be integrating object scanning and detection to reduce prior information and knowledge reliability. SLAM operations are vital in generating knowledge and conducting the desired task. This way, operating requires less

human effort and helps generate a knowledge base for further analysis and computations. There is one more direction where progress could be made: fitting a smooth curve along the sequenced viewpoints, such as B-Splines that take care of the velocity and acceleration variation of the drone.

Bibliography

[AFM00] Arkin, E. M.; Fekete, S. P.; Mitchell, J. S.
Approximation algorithms for lawn mowing and milling A preliminary version of this paper was entitled "The lawnmower problem" and appears in the Proc. 5th Canad. Conf. Comput. Geom., Waterloo, Canada, 1993, pp.461 466.
In: Computational Geometry, 17 (2000) 1, DOI https://doi.org/10.1016/S0925-7721(00)00015-8, https://www.sciencedirect.com/science/article/pii/S0925772100000158, pp. 25–50.

[ATC17] Alzugaray, I.; Teixeira, L.; Chli, M.
Short-term UAV path-planning with monocular-inertial SLAM in the loop
In: ISBN 9781509046331, pp. 2739–2746.

[BA94] Balch, T.; Arkin, R. C.
Communication in reactive multiagent robotic systems
In: Autonomous Robots, 1 (Mar. 1994), DOI 10.1007/BF00735341/METRICS, https://link.springer.com/article/10.1007/BF0073%205341, pp. 27–52.

[Blea] Blender
Blender: Decimation
https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/decimate.html?highlight=decimation.

[Bleb] Blender
Blender: Introduction
https://docs.blender.org/manual/en/latest/getting_started/about/introduction.html.

[Bra00] Bradski, G.
The openCV library.
In: Dr. Dobb's Journal: Software Tools for the Professional Programmer, 25 (2000) 11, pp. 120–123.

[Bro86] Brooks, R.
A robust layered control system for a mobile robot
In: IEEE Journal on Robotics and Automation, 2 (Mar. 1986) 1, DOI 10.1109/JRA.1986.1087032, pp. 14–23.

[CBP19] Cabreira, T. M.; Brisolara, L. B.; Paulo, R. F.
Survey on Coverage Path Planning with Unmanned Aerial Vehicles
In: Drones 2019, Vol. 3, Page 4, 3 (Jan. 2019), DOI 10.3390/DRONES3010004, https://www.mdpi.com/2504-446X/3/1/4/htm%20https://www.mdpi.com/2504-446X/3/1/4, p. 4.

[CBS15] Coutinho, W.; Bandeira, T.; Subramanian, A.; Brito, A.
Analysis of Path Planning Algorithms Based on Travelling Salesman Problem Embedded in UAVs
In:

[CG12] Crassin, C.; Green, S.
Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer
Mar. 2012, ISBN 978-1-4398-9376-0, DOI 10.1201/b12288-26.

[CGT19] Cicek, C. T.; Gultekin, H.; Tavli, B.; Yanikomeroglu, H.
UAV Base Station Location Optimization for Next Generation Wireless Networks: Overview and Future Research Directions
In: 2019 1st International Conference on Unmanned Vehicle Systems-Oman, UVS 2019 (, Mar. 2019), DOI 10.1109/UVS.2019.8658363, https://squ.pure.elsevier.com/en/publications/uav-base-station-location-optimization-for-next-generation-wirele, p. 8658363.

[Cho00] Choset, H.
Coverage of known spaces: The boustrophedon cellular decomposition
In: Autonomous Robots, 9 (2000), DOI 10.1023/A:1008958800904/METRICS, https://link.springer.com/article/10.1023/A:1008958800904, pp. 247–253.

[Cho01] Choset, H.
Coverage for robotics-A survey of recent results
In: Annals of Mathematics and Artificial Intelligence, 31 (2001), pp. 113–126.

[CK21] Cheikhrouhou, O.; Khoufi, I.
A Comprehensive Survey on the Multiple Travelling Salesman Problem: Applications, Approaches and Taxonomy
In: Computer Science Review, 40 (Feb. 2021), DOI 10.1016/j.cosrev.2021.100369, http://arxiv.org/abs/2102.12772%20http://dx.doi.org/10.1016/j.cosrev.2021.100369.

[CNL09] Crassin, C.; Neyret, F.; Lefebvre, S.; Eisemann, E.
GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering
In:

[CSu09] Cormen, T. H.; Sussman, J.; [u.a.]
Introduction to algorithms [Hauptbd.]
In: (2009), https://katalog.ub.uni-paderborn.de/records/PAD_ALEPH001247720, p. 1180.

[ČVU18] ČVUT, F.
Components of a 3D Mesh Model
https://courses.fit.cvut.cz/BI-3DT/@B161/en/tutorials/mesh/index.html.

[CWL09] Chen, H.; Wang, X. M.; Li, Y.
A Survey of Autonomous Control for UAV
In: 2009 International Conference on Artificial Intelligence and Computational Intelligence, 2 (2009), DOI 10.1109/AICI.2009.147, pp. 267–271.

[DGB06] Dorigo, M.; Gambardella, L. M.; Birattari, M.; Martinoli, A.; Poli, R.; Stützle, T.
Ant Colony Optimization and Swarm Intelligence
In: 4150 (2006), DOI 10.1007/11839088, http://link.springer.com/10.1007/11839088.

[Dor] Dorméus, E.
Cooperative Path Planning of Unmanned Aerial Vehicles Cooperative Path Planning of Unmanned Aerial Vehicles
https://www.academia.edu/40817398/Cooperative_Path_Planning_of_Unmanned_Aerial_Vehicles_Cooperative_Path_Planning_of_Unmanned_Aerial_Vehicles.

[EBM15] Evers, L.; Barros, A. I.; Monsuur, H.; Wagelmans, A.
UAV Mission Planning: From Robust to Agile
In: Operations Research/ Computer Science Interfaces Series, 56 (2015), DOI 10.1007/978-3-319-12075-1_1, pp. 1–17.

[ED08] Eisemann, E.; Décoret, X.
Single-pass GPU solid voxelization for real-time applications
In: ISBN 978-1-56881-423-0, pp. 73–80.

[FCS10] Furukawa, Y.; Curless, B.; Seitz, S. M.; Szeliski, R.
Towards Internet-scale multi-view stereo
In: pp. 1434–1441.

[FL14] Feng, B.; Liu, Y.
An improved RRT based path planning with safe navigation
In: Applied Mechanics and Materials, 494-495 (2014), DOI 10.4028/WWW.SCIENTIFIC.NET/AMM.494-495.1080, https://www.researchgate.net/

publication/269368835_An_Improved_RRT_Based_Path_Planning_with_Safe_Navigation, pp. 1080–1083.

[GC13] Galceran, E.; Carreras, M.
A survey on coverage path planning for robotics
In: Robotics and Autonomous Systems, 61 (Dec. 2013), DOI 10.1016/j.robot.2013.09.004, pp. 1258–1276.

[HWZ] Hoppe, C.; Wendel, A.; Zollmann, S.; Pirker, K.; Irschara, A.; Bischof, H.; Kluckner, S.
Photogrammetric Camera Network Design for Micro Aerial Vehicles

[HYF98] Huang, J.; Yagel, R.; Filippov, V.; Kurzion, Y.
An accurate method for voxelizing polygon meshes
In: IEEE Symposium on Volume Visualization (Cat. No.989EX300) (, 1998), pp. 119–126.

[Kar72] Karp, R. M.
Reducibility among Combinatorial Problems
In: Complexity of Computer Computations (, 1972), DOI 10.1007/978-1-4684-2001-2_9, https://link.springer.com/chapter/10.1007/978-1-4684-2001-2_9, pp. 85–103.

[Law85] Lawler, E. L.
The Traveling salesman problem : a guided tour of combinatorial optimization
In: (1985), https://www.wiley.com/en-us/The+Traveling+Salesman+Problem%3A+A+Guided+Tour+of+Combinatorial+Optimization-p-9780471904137, p. 465.

[LEW19] Li, Y.; Eslamiat, H.; Wang, N.; Zhao, Z.; Sanyal, A. K.; Qiu, Q.
Autonomous waypoints planning and trajectory generation for multi-rotor UAVs
In: DESTION 2019 - Proceedings of the Workshop on Design Automation for CPS and IoT (, Apr. 2019), DOI 10.1145/3313151.3313163, https://dl.acm.org/doi/10.1145/3313151.3313163, pp. 31–40.

[Lin65] Lin, S.
Computer solutions of the traveling salesman problem
In: The Bell System Technical Journal, 44 (Dec. 1965) 10, DOI 10.1002/j.1538-7305.1965.tb04146.x, pp. 2245–2269.

[LK10] Laine, S.; Karras, T.
Efficient Sparse Voxel Octrees-Analysis, Extensions, and Implementation
2010, http://code.google.com/p/efficient-sparse-voxel-octrees/.

[LWW13] Lin, Y.; Wu, D.; Wang, X.; Wang, X.; Gao, S.
Improving rrt-connect approach for optimal path planning by utilizing prior information
In: International Journal of Robotics and Automation, 28 (2013), DOI 10.2316/Journal.206.2013.2.206-3781, pp. 146–153.

[MB96] Mackenzie, D. C.; Balch, T. R.
Making a Clean Sweep: Behavior Based Vacuuming
1996.

[MCD05] Merino, L.; Caballero, F.; Dios, J. M.-d.; Ollero, A.
Cooperative Fire Detection using Unmanned Aerial Vehicles
In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation (, Jan. 2005), https://www.academia.edu/8173251/Cooperative_Fire_Detection_using_Unmanned_Aerial_Vehicles.

[MMP04] Maddula, T.; Minai, A. A.; Polycarpou, M. M.
Multi-Target Assignment and Path Planning for Groups of UAVs
In: (2004), DOI 10.1007/978-1-4613-0219-3_15, https://link.springer.com/chapter/10.1007/978-1-4613-0219-3_15, pp. 261–272.

[MN19] Maghazei, O.; Netland, T.
Drones in manufacturing: Exploring opportunities for research and practice
In: Journal of Manufacturing Technology Management, Forthcoming (Sept. 2019), DOI 10.1108/JMTM-03-2019-0099.

[MNF21] Maghazei, O.; Netland, T. H.; Frauenberger, D.; Thalmann, T.
Automatic Drones for Factory Inspection: The Role of Virtual Simulation
In: ISBN 9783030859091, pp. 457–464.

[Num] Numpy
Numpy: The fundamental package for scientific computing with Python
https://numpy.org/doc/stable/user/whatisnumpy.html.

[OLM05] Ollero, A.; Lacroix, S.; Merino, L.; Gancet, J.; Wiklund, J.; Remuss, V.; Perez, I. V.; Gutiérrez, L. G.; Viegas, D. X.; Benítez, M. A. G.; Mallet, A.; Alami, R.; Chatila, R.; Hommel, G.; Lechuga, F. J. C.; Arrue, B. C.; Ferruz, J.; Dios, J. R. M.-d.; Caballero, F.
Multiple eyes in the skies: Architecture and perception issues in the COMETS unmanned air vehicles project
In: IEEE Robotics and Automation Magazine, 12 (June 2005), DOI 10.1109/MRA.2005.1458323, pp. 46–57.

[Ope] Open3D
Open3D: A Modern Library for 3D Data Processing
http://www.open3d.org/docs/release/index.html.

[OSK15] Oh, H.; Shin, H.-S.; Kim, S.; Tsourdos, A.; White, B. A.
Cooperative Mission and Path Planning for a Team of UAVs
In: Handbook of Unmanned Aerial Vehicles (, 2015), DOI 10.1007/978-90-481-9707-1_14, https://link.springer.com/referenceworkentry/10.1007/978-90-481-9707-1_14, pp. 1509–1545.

[OY21] Osaba, E.; Yang, X.-S.
Applied Optimization and Swarm Intelligence: A Systematic Review and Prospect Opportunities
In: May 2021, ISBN 978-981-16-0661-8, pp. 1–23.

[Pan11] Pantaleoni, J.
VoxelPipe: A Programmable Pipeline for 3D Voxelization
In: Dachsbacher, C.; Mark, W.; Pantaleoni, J. (eds.), ISBN 978-1-4503-0896-0.

[Pau03] Pauly, M.
Point primitives for interactive modeling and processing of 3D-geometry
In:

[Peh12] Pehlivanoglu, Y. V.
A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV
In: Aerospace Science and Technology, 16 (Jan. 2012), DOI 10.1016/J.AST.2011.02.006, pp. 47–55.

[PP21] Pehlivanoglu, Y. V.; Pehlivanoglu, P.
An enhanced genetic algorithm for path planning of autonomous UAV in target coverage problems
In: Applied Soft Computing, 112 (Nov. 2021), DOI 10.1016/J.ASOC.2021.107796, p. 107796.

[sel10] selvi, vetri
Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques
In: International Journal of Computer Applications (, Jan. 2010), https://www.academia.edu/7993888/Comparative_Analysis_of_Ant_Colony_and_Particle_Swarm_Optimization_Techniques.

[Sko07] Skoglar, P.
Technical report from Automatic Control at Linköpings universitet UAV Path and Sensor Planning Methods for Multiple Ground Target Search and Tracking-A Literature Survey
2007.

[SMD12] Strasdat, H.; Montiel, J. M.; Davison, A. J.
Visual SLAM: Why filter?
In: Image and Vision Computing, 30 (Feb. 2012), DOI 10.1016/J.IMAVIS.2012.02.009, pp. 65–77.

[SS10] Schwarz, M.; Seidel, H.-P.
Fast Parallel Surface and Solid Voxelization on GPUs
In:

[SSS08] Snavely, N.; Seitz, S. M.; Szeliski, R.
Skeletal graphs for efficient structure from motion
In: 26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR (, 2008), DOI 10.1109/CVPR.2008.4587678, https://www.researchgate.net/publication/221362579_Skeletal_graphs_for_efficient_structure_from_motion.

[ST19] Shen, F.; Tarbutton, J.
A voxel based automatic tool path planning approach using scanned data as the stock
In: pp. 26–32.

[Teo09] Teodorović, D.
Bee Colony
In: Innovations in Swarm Intelligence, 248 (2009), DOI 10.1007/978-3-642-04225-6_3, pp. 39–60.

[THI16] THILAKANATHAN, D.
3D MODELING FOR BEGINNERS, LEARN EVERYTHING YOU NEED TO KNOW ABOUT 3D MODELING!
Thilakanathan Studios., 2016.

[VFS01] Vazquez, P.-P.; Feixas, M.; Sbert, M.; Heidrich, W.
Viewpoint Selection using Viewpoint Entropy
In:

[XLO22] Xiang, D.; Lin, H.; Ouyang, J.; Huang, D.
Combined improved A and greedy algorithm for path planning of multi-objective*

mobile robot
In: Scientific Reports, 12 (Aug. 2022), DOI 10.1038/s41598-022-17684-0.

[YFG22] Yang, F.; Fang, X.; Gao, F.; Zhou, X.; Li, H.; Jin, H.; Song, Y.
Obstacle Avoidance Path Planning for UAV Based on Improved RRT Algorithm
In: (2022), DOI 10.1155/2022/4544499, https://doi.org/10.1155/2022/4544499.

[ZCZ18] Zhao, B.; Chen, X.; Zhao, X.; Jiang, J.; Wei, J.
Real-Time UAV Autonomous Localization Based on Smartphone Sensors
In: Sensors 2018, Vol. 18, Page 4161, 18 (Nov. 2018), DOI 10.3390/S18124161, https://www.mdpi.com/1424-8220/18/12/4161/htm%20https://www.mdpi.com/1424-8220/18/12/4161, p. 4161.

[ZML15] Zhang, B.; Mao, Z.; Liu, W.; Liu, J.
Geometric Reinforcement Learning for Path Planning of UAVs
In: Journal of Intelligent and Robotic Systems: Theory and Applications, 77 (Feb. 2015), DOI 10.1007/S10846-013-9901-Z/METRICS, https://link.springer.com/article/10.1007/s10846-013-9901-z, pp. 391–409.

[ZPK18] Zhou, Q.-Y.; Park, J.; Koltun, V.
Open3D: A Modern Library for 3D Data Processing
In: (Jan. 2018), https://arxiv.org/abs/1801.09847v1.

[ZTZ13] Zienkiewicz, O.; Taylor, R.; Zhu, J.
Automatic Mesh Generation
In: The Finite Element Method: its Basis and Fundamentals (, 2013), DOI 10.1016/B978-1-85617-633-0.00017-4, pp. 573–640.

List of Algorithms

1 3D mesh tripling and unique Face Color and Keys Generation 29

2 Initial View Points Generation 31

3 Selection of ViewPoints 36

4 Generation of the Order of the ViewPoints 40

5 Obstacle Management using Voxelisation 45

6 Samling Module for Initial View Points Generation 50

List of Figures

1.1 Test Case 3D Mesh Model . . . 3

2.1 Components of a 3D Mesh Model . . . 6
2.2 Camera Position Computation . . . 8
2.3 Check-Point-based Path Planning . . . 9
2.4 Boustrophedon Path . . . 10
2.5 Thin Voxelization . . . 12
2.6 Fully Conservative Voxelization . . . 12
2.7 Simple Voxelization Pipeline . . . 13
2.8 Bounding polygon of a triangle used for conservative rasterization . . . 14
2.9 Implementation of our voxelization pipeline on top of the GPU rasterization pipeline . . . 14
2.10 Memory Organization in Octree Representation . . . 15
2.11 RRT algorithm expansion process . . . 16

3.1 Toy Train Model . . . 20
3.2 Decimated Train Model . . . 21

4.1 A Cluster of Camera View Points surrounding the Object . . . 26
4.2 Camera View Point for a Trianlge in a 3D Mesh . . . 27
4.3 Mesh Tripling . . . 30
4.4 Viewpoint Entity Structure . . . 30
4.5 Trianlge Entity Structure . . . 32
4.6 Rendered Images . . . 33
4.7 Selected View Points around the Object . . . 35
4.8 Visible Triangles . . . 37
4.9 Camera View Points Trajectory covering an Object . . . 39
4.10 Trajectory Generation using Line Segments . . . 41
4.11 Line-Segment Trajectory around the Object . . . 42
4.12 Voxelised Model . . . 43
4.13 Line Segment Trajectory with a Dummy Obstacle . . . 44

5.1 Tripling of the 3D Train Mesh Model . . . 46
5.2 Clustering of View Points over small triangle area regions . . . 47
5.3 The mesh Duct on the Train's roof having small triangles . . . 48
5.4 Selected View Points after Sampling implementation . . . 49

5.5 Reduction in the number of Triangles 50
5.6 Triangle Associations per View Point 51
5.7 Coverage Map of the Decimated Model after implementation of Sampling . 52
5.8 Random missing patches in the Coverage Map 53
5.9 Number of Triangles Considered/Not-Considered 54
5.10 Results for Small Triangle Area Sampling Probability as 0.2 55
5.11 Missing patches in the Coverage Map for Sampling Probability of 0.2 . . . 56
5.12 Missing patches in the Coverage Map for Sampling Probability of 0.7 . . . 56
5.13 Results for Small Triangle Area Sampling Probability as 0.2 58
5.14 Voxelisation of the 3D Mesh Model 59
5.15 Trajectory Generation from a Random Initial View Point 59
5.16 Illustration showing Loop Formation with absence of Obstacles 59
5.17 Trajectory with Start Point (Green) and End Point (Red) 61
5.18 Distance covered by the Trajectory of Decimated Model 61
5.19 Distance covered by the Trajectory of Toy Train Model 62
5.20 Distance covered by the Trajectory of Toy Train Model (no Obstacle Management) 62