Autoregressive 3D Scene Reconstruction with Structured Data Models

Maria Saleem

Master Thesis
Erasmus Mundus Master in
Marine and Maritime Intelligent Robotics
Universitat Jaume I

October 22, 2024

Supervised by:

Prof. Lledó Museros Cabedo (Universitat Jaume I)
Abdelrhman Bassiouny (IAI, University of Bremen)
Prof. Michael Beetz, PhD. (IAI, University of Bremen)

Co-funded by the
Erasmus+ Programme
of the European Union

To my parents for their endless love and support....

Acknowledgments

First, this thesis would not have been feasible without the help, support, guidance, and encouragement of so many remarkable individuals who had been with me throughout this academic journey.

Above all, I would like to express my deepest gratitude to my supervisor Prof. Michael Beetz from the Institute of Artificial Intelligence of the University of Bremen for this great opportunity for a master's thesis in such a stimulating environment. I would like to thank him for his expert guidance and insightful feedback. Above all, his support and constant encouragement toward excellence during the process were priceless.

I also want to convey the same magnitude of appreciation to my advisor from Universitat Jaume I, Prof. Lledó Museros Cabedo, who has always been there with unwavering support and enthusiasm. You've shown me how to navigate not just the technical challenges, but the sometimes equally difficult mental hurdles of research. Your feedback and suggestions have sparked new ideas and improved my thesis study.

I would also like the thank Abdelrhman Bassiouny, from the Institute of Artificial Intelligence, University of Bremen who helped me a lot in the early stages of my Master's thesis and taught me what to do when I was lost. This work would not be possible without his support and enthusiasm.

I also wish to express my profound gratitude to all the great people I got to know during my time at the University of Bremen. From sharing ideas and expertise to sharing coffee and laughs, you turned what could have been a very isolating process into a collaborative and friendly environment. Thank you for always being there, whether troubleshooting a bug or guiding me through a new concept. I truly appreciate all your help.

I am deeply grateful to Prof. Pedro Sanz from UJI for his invaluable advice and consistent guidance throughout my master's journey. His insights and support have been instrumental. I also want to express my sincere thanks to Prof. Ricard Marxer and Prof. Vincent Hugel from UTLN for their consistent support and dedication from the very start of the master's program. Their mentorship and expertise have been invaluable to my growth, both professionally and academically. I appreciate the contributions they have made to my success.

To my fellow students in the MIR master's program, thanks for being my companions along the arduous yet rewarding academic ride. From days of grinding through assignments to days of celebration-however small-your friendship and camaraderie are strengths to me. You have all been part of this journey, big or small.

Finally, to the real stars of this journey: my family. To my parents, for believing in me even when I didn't. Your support has formed the basis of all I have been able to achieve.

Abstract

Reconstructing 3D environments from sparse point cloud data isn't just about filling in the voids —it's about reimagining the whole scene creatively from minimal information. This work takes on that challenge, using an encoder-decoder architecture to turn scattered data into dense, dynamic 3D environments with impressive precision.

The approach used here generates flexible, script-based commands that predict detailed spatial information, using this description, the scene is reconstructed. While traditional models often rely on dense data, which limits their effectiveness in practical scenarios, this method thrives on sparse input. The method uses a Transformer-based decoder which takes the encoded features obtained from sparse input and produces high-level commands and key spatial parameters (e.g., height, width, position). 3D scenes are generated iteratively, learning from a large dataset of indoor environments. The script generation plays a central role in this model, hence acting as a framework to transform sparse point cloud data into structured 3D reconstructions. The ability of the system to achieve flexibility and precision can be attributed to decomposing the process into well-defined, script-based commands. Each command is instructive on how to reconstruct a particular element within the scene in unambiguous terms and, therefore, enables the model to work efficiently even with limited data points. Due to its scalability, this work is quite suitable for urban planning, Virtual Reality (VR), and automation in simulation. More importantly, the model could support real-time geometric prediction and rendering such that the system dynamically evolves the scene without requiring dense data. In this way, the model can prove that 3-D reconstruction may be efficient, reliable, and adaptive operating under the script-driven precision umbrella.

Contents

CHAPTER 1

Introduction

Contents

1.1 Work Motivation

Recent advances in architecture, robotics, and VR have created an increasing demand for detailed, accurate, and efficient reconstructions of real-world environments in three dimensions. The most recent applications rely on dense 3D models capturing not just the coarse structure of an environment but all surfaces, objects, and their spatial relationships in detail. The level of detail here is not an option but an imperative in applications like indoor navigation for autonomous robots or Augmented Reality (AR)/VR systems that need precise spatial awareness. Without such precision, these systems will either result in inefficiencies or, worse still, in errors that impede proper interaction with their surroundings.

1.1.1 Moving Beyond 2D: The Third Dimension

2D data lacks intrinsic depth of information for complete spatial context understanding by definition. On the other hand, 3D reconstruction presents a more holistic virtual representation of an environment, to which a finer level of understanding can be developed through advanced manipulations of virtual objects. For example, in robotics, 3D models present more complete traversal capabilities since the depth information enables a robot to build a map of its surroundings and make real-time decisions using such accurate spatial understanding. Amongst them, and probably most importantly, the work by Beltran [18] shows that 3D data is essential for precise decision-making involved in autonomous systems.

However, virtual objects today are largely hand-designed, which is painfully slow and bounds the scaling of such models. In recent years, there has been progress in indoor data technologies namely mobile scanners like Microsoft Kinect using techniques like [31] and [2].In this regard, automated 3D reconstruction from real-world data bridges the gap from manually designed objects to accurately replicated ones, not only raising efficiency but also enhancing the quality of virtual environments.

Mapping and localization are indispensable in robotics, where continuous depth-aware 3D reconstruction takes place. By conducting the real-time, precise sensing of 3D, a robot will be able to conceive its ambient environment, recognize objects, and manage navigation in space elegantly. Moreover, 3D reconstructions have the advantage of visualizing scenes from several perspectives the bird's-eye view which may also support decision making.

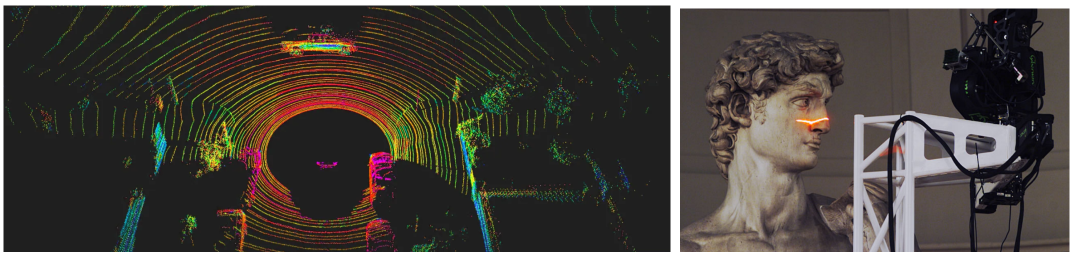


Figure 1.1: Left: A LiDAR system for autonomous cars sensing the environment (reproduced from [9]), and right: a close-up laser scan of the David statue captured under the Digital Michelangelo project (reproduced from [22]).

AR also heavily relies on the accurate creation of 3D reconstructions. For example, to allow virtual objects to be placed convincingly into the real world. If the spatial data is imprecise, the virtual object may not behave well in its environment, thus affecting the user experience.

1.1.2 The Challenge of Sparsity

While point clouds, by their nature, may provide a detailed capture of the representations of the space in an environment, they are inherently sparse, meaning they sample continuous surfaces only at discrete points. This sparsity may be problematic in applications aiming at realistic models. To put this into perspective, suppose that an autonomous vehicle was mapping its surroundings using LiDAR. As a result, partial representations of the objects in their surroundings would remain, and thus gaps in the reconstructed model would persist. This can be observed in Figure: 1.1 which demonstrates how point clouds can be obtained, with the LiDAR scan emphasizing spatial awareness in autonomous vehicles and the detailed laser scan showing the fine details of the David statue. Notice the sparsity and the incomplete scanning of the statue's face [29].

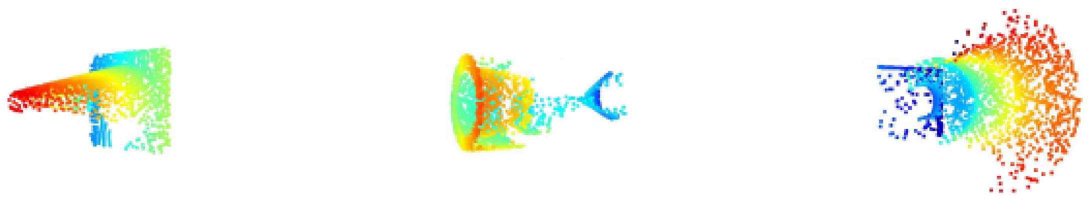


Figure 1.2: (a) Left side of the object captured, (b) Foot partially missing, (c) Overall sparsity and missing sections of the object. This figure shows that a point cloud (obtained from [42]) represents objects as a collection of independent and scattered measurements in 3D space, resulting in sparsity and incompleteness depending on the scanner's position.

This discontinuity is a problem, mainly when the application involves realistic content or decision-making in robotics. In most cases, it is of utmost importance to construct continuous and dense surfaces from sparse point clouds for realistic model generation or to present the robot with enough details about the structure of an object to make realistic decisions. Though the problem may be overcome in controlled environments, real-world conditions very often produce partial scans and information loss, as seen in Figure: 1.2.

1.1.3 From Point Clouds to Dense Representations

Traditionally, dense polygonal meshes have played a central role in creating virtual 3D environments and are capable of providing detailed representations of objects and spaces. They are highly important in content creation industries such as gaming and virtual simulation, where precision and realism are considered crucial elements. However, such sparse point clouds face various challenges while being transformed into dense representations due to several gaps inherent in the data and a lack of continuity therein.

Recent machine learning advances, in particular, data-driven approaches have been seeking to alleviate this issue for the improvement of quality in 3D reconstructions from sparse input data. Indeed, these methods have visibly shown great potential to bridge the gap between sparse point clouds and dense polygonal meshes. However, sparser the data, the limitations are still present in the quest for accuracy.

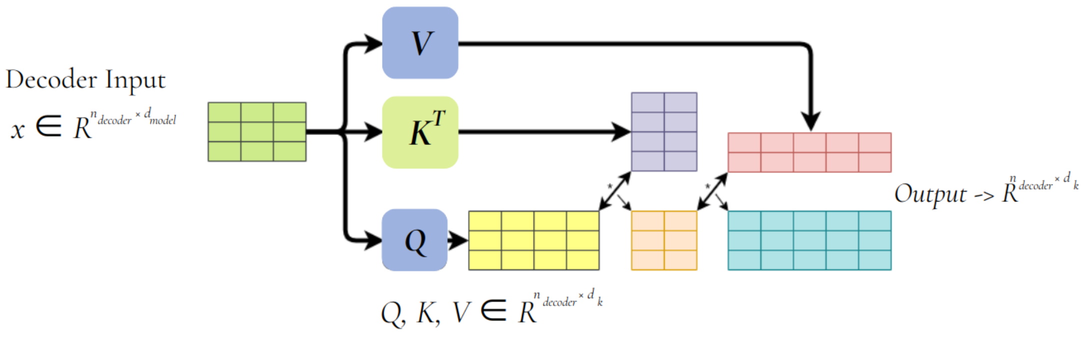
1.1.4 Transformers: Learning Complex Dependencies in 3D Reconstruction

Although transformers were originally proposed for natural language [38] processing, recently it has become a strong solution to learn complicated relations in 3D spatial data. Based on the successful performance of Vision Transformers (ViTs) [12] on processing the visual data, capturing long-range dependencies can be called particularly suitable for the modeling of spatial relationships from sparse point clouds.

Transformers can thus focus on relevant aspects of a scene using multi-head attention mechanisms even when data is incomplete, which is critical when operating on sparse point clouds. The model should hence infer the missing spatial information and create a coherent 3D representation. Attention mechanisms and masking are utilized to enhance transformer models in high-quality 3D reconstructions; these are explained in depth in subsequent sections.

Attention Mechanisms in 3D Reconstruction

Transformers' multi-head mechanism of attention allows it to pay attention to different parts of the 3-D scene at once (see Figure: 1.3). Given that most scene reconstruction involves taking raw point clouds and trying to make sense of them, the spatial relations among sets of different elements are especially crucial [38].



Decoder Input
$x \in R^{n_{decoder} \times d_{model}}$
V
K^T
Q
*
*
Output -> $R^{n_{decoder} \times d_k}$
$Q, K, V \in R^{n_{decoder} \times d_k}$

Figure 1.3: Self-attention head computing relationships within the decoder sequence for capturing context in sequence generation.

Cross-attention will further enable the model to align decoder queries with encoder keys and values, hence enabling the decoder to generate commands containing the most relevant spatial features extracted by the encoder as shown in Figure: 1.4.

Masking for Sequence Generation

Masking ensures that the model only attends to the previously generated commands during the generation of sequences in 3D reconstruction. This mechanism is highly important in keeping the type of sequence autoregressive and, at the same time, it ensures temporally consistent predictions with high accuracy.

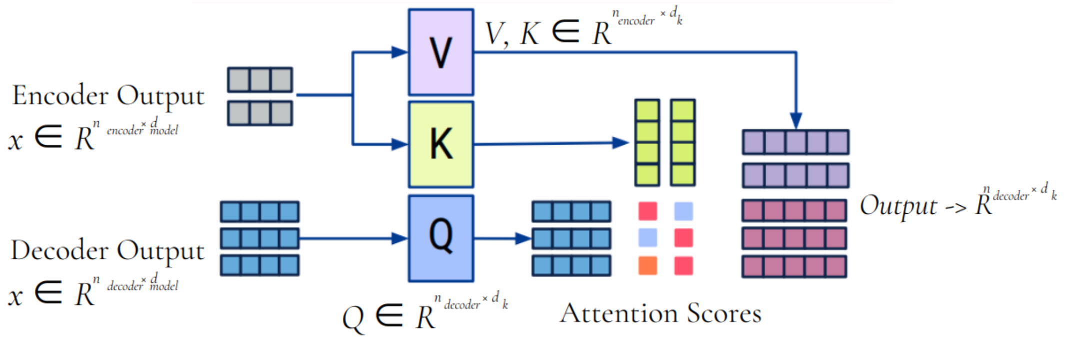

Encoder Output
$x \in R^{n_{encoder} \times d_{model}}$
Decoder Output
$x \in R^{n_{decoder} \times d_{model}}$
V
K
Q
$V, K \in R^{n_{encoder} \times d_k}$
$Q \in R^{n_{decoder} \times d_k}$
Attention Scores
Output -> $R^{n_{decoder} \times d_k}$

Figure 1.4: Cross-attention head aligning decoder queries with encoder keys and values.

$$\text{MaskedAttention}(Q_i, K_i) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}} + \text{mask}\right) V_i \tag{1.1}$$

1.1.5 Why Transformers for 3D Reconstruction?

Volumetric, surface-based, and multi-view point cloud-based 3D reconstruction methods have recently been greatly discussed in the literature. The volumetric methods represent a point cloud as voxel grids but are computationally very expensive, and for larger scenes, because of resolution limits, important detailed information is lost. Surface-based methods, like Poisson Surface Reconstruction, fit the surfaces directly to the point clouds, but when the data is sparse or inconsistent, incompleteness and surface distortion may occur. In multiview stereo methods, consistent viewpoints are required for scene reconstruction, and this does not always hold true for occluded scenes, especially indoors.

Thus, there is a fundamental dependency of these approaches on dense, consistent input that poses quite a great challenge to applying them to real-world point cloud data, considering how real-world data often tends to be sparse, uneven, or incomplete because of sensor limitations relating to occlusions or limitations of capture angles. Another challenge they have is the appropriate representation of complex geometries, to which they usually are bound by dense data to interpolate and fill gaps. Even methods that use occupancy grids or implicit functions, like Signed Distance Fields (SDFs), require heavy pre-processing and face scaling-up issues with computational overhead for detailed reconstructions.

On the other hand, our transformer-based approach will exploit self-attention mechanisms in modeling long-range dependencies to enable sparse data environment reconstructions. This shift lets the model perform inference about spatial relationships on the whole dataset, rather than just depending on local information, effectively filling gaps in point clouds. Furthermore, instead of static 3D models, the model outputs structured commands; thus, the approach is more flexible and adaptable to 3D reconstruction. The command-based generation will not only mitigate the problems related to sparse input but also allow for real-time adjustments and dynamic updates that directly address the limitations found in previous methods depending on complete point cloud coverage.

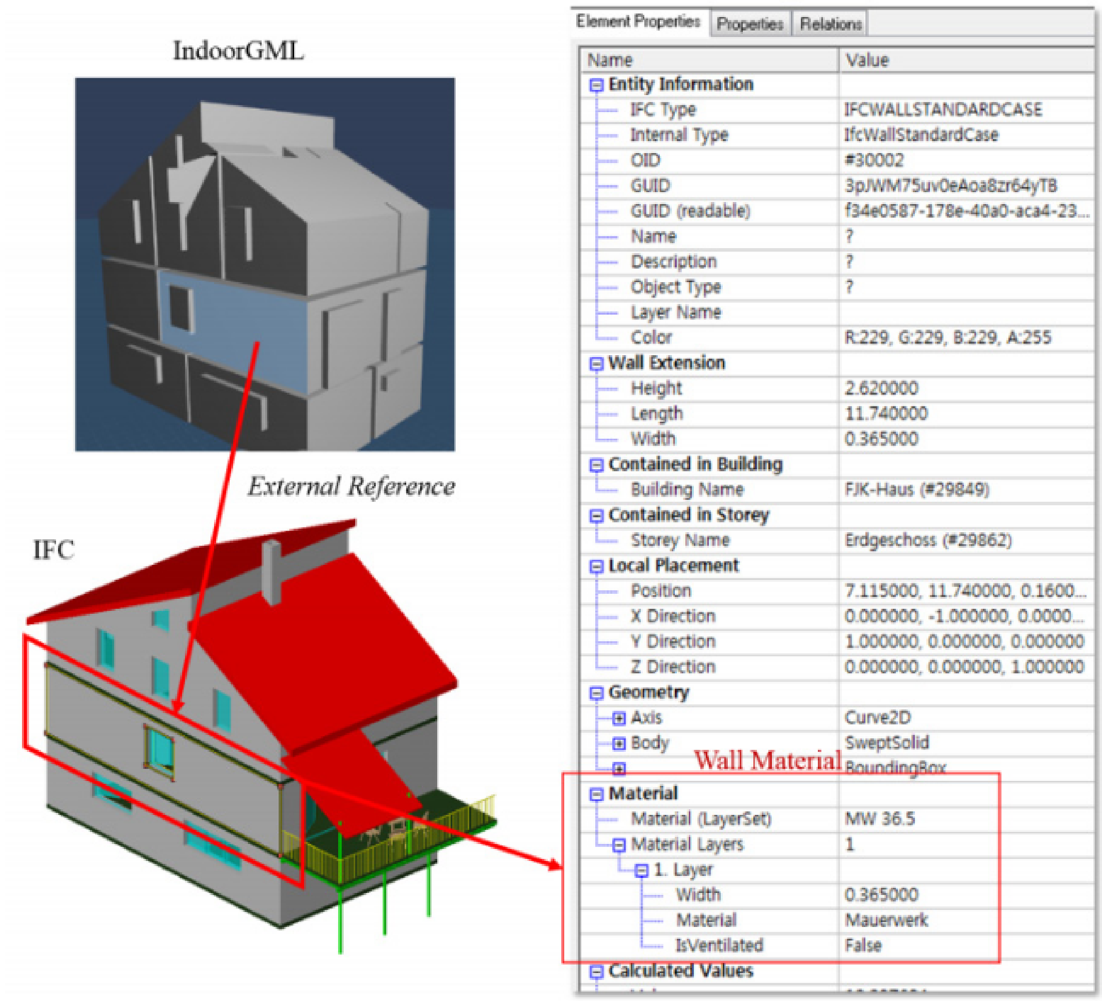
1.2 Related Work

1.2.1 Indoor Data Models and Standards

This section is devoted to discussing indoor data models, important contributors to shaping this work's outcome given their very important applications in in architectural modeling and CAD-based design among others. To create a 3D model compatible with most established models such as Indoor Geographic Markup Language (IndoorGML), Indoor OpenStreetMap (IndoorOSM), and Industry Foundation Classes (IFC) we have to understand the specifications and standards of the representations of indoor space. As indoor mapping and services are becoming more popular, standardization regarding data collection, management, and software development has become vital. Four major models are presently widespread:

- IndoorGML: It is the Open Geospatial Consortium (OGC) standard for indoor navigation and topology. For example, spaces such as rooms and corridors can be modeled, but connectivity is the focus, not architectural details [21].
- City Geography Markup Language, Level of Detail 4 (CityGML, LoD4): This OGC standard is an eXtensible Markup Language (XML)-based standard. It contains detailed interior modeling for LoD4, including rooms, stairs, and doors. It would therefore be good to go for a complete building model [15].
- IFC: A Building Information Modeling (BIM)-based standard focusing on architectural components and building maintenance. Though object-oriented and offering detailed architectural features, IFC lacks spatial and navigational relationships, making it less ideal for navigation-centric models like IndoorGML [17].
- IndoorOSM: An indoor extension of OpenStreetMap, focusing on representing simple indoor spaces (e.g., rooms, corridors) through Volunteered Geographic Information (VGI). While not as comprehensive as the other models, it simplifies data collection and representation [14].

One of the main applications of these models is the transition from a 3D reconstructed model to a Geographic Information System (GIS)/BIM model. These can benefit indoor applications, as expert communities widely support these models. While IndoorGML represents interior spaces using a cellular approach, with the smallest units being cells, IFC models these spaces through architectural components. IndoorGML is useful for navigation, while IFC focuses on building maintenance details. Therefore, IndoorGML can be enhanced by referencing IFC for specific information such as wall materials and thickness (see Figure: 1.5) [21]



Name	Value
Entity Information	
IFC Type	IFCWALLSTANDARDCASE
Internal Type	IfcWallStandardCase
OID	#30002
GUID	3pJWM75uv0eAoa8zr64yTB
GUID (readable)	f34e0587-178e-40a0-aca4-23...
Name	?
Description	?
Object Type	?
Layer Name	
Color	R:229, G:229, B:229, A:255
Wall Extension	
Height	2.620000
Length	11.740000
Width	0.365000
Contained in Building	
Building Name	FJK-Haus (#29849)
Contained in Storey	
Storey Name	Erdgeschoss (#29862)
Local Placement	
Position	7.115000, 11.740000, 0.1600...
X Direction	0.000000, -1.000000, 0.0000...
Y Direction	1.000000, 0.000000, 0.000000
Z Direction	0.000000, 0.000000, 1.000000
Geometry	
Axis	Curve2D
Body	SweptSolid
	BoundingBox
Material	
Material (LayerSet)	MW 36.5
Material Layers	1
1. Layer	
Width	0.365000
Material	Mauerwerk
IsVentilated	False
Calculated Values	

Figure 1.5: The figure demonstrates how IFC can serve as an external reference to populate IndoorGML with missing data about walls, materials, and architectural components [21].

1.2.2 Indoor 3D Reconstruction Methods

Indoor 3D reconstruction methods from point clouds can be classified into four broad categories: Planar-based, Volumetric-based, Mesh-based reconstruction, and Indoor Scene Interpretation and Semantic Labeling. The classification here has put more emphasis on the methodology rather than data acquisition and focused on the process of reconstruction. There is some overlap; for example, semantic labeling may assist in planar or volumetric approaches.

Planar-Based Reconstruction

The planar-based methods identify planar primitives in the point clouds by the least squares, region growing, and Random Sample Consensus (RANSAC). All these techniques deal with polygonal representations of indoor spaces that are particularly suitable for environments that follow the Manhattan World assumption. For example, [10] presented a method for planar region detection in building facades. They reconstructed polyhedrons from the sparse scanned range data by detecting the intersection of planes and applying an edge extraction algorithm to compute the boundary. The work here is outdoors, but its basic principles are also applicable indoors for plane detection.

Following this, [33] introduced an automated planar 3D modeling system for indoor environments. Their approach first classifies points based on their normal vectors to detect floors, ceilings, and walls. RANSAC is then applied to detect planar primitives, which after two-step plane-fitting are used to generate wall polygons and identify such features as staircases. Although effective in detecting staircases as observed in Figure: 1.6, their method is limited to detecting walls aligned with the X and Y axes, and it fails to account for openings like doors or cluttered data, which remains a significant limitation.

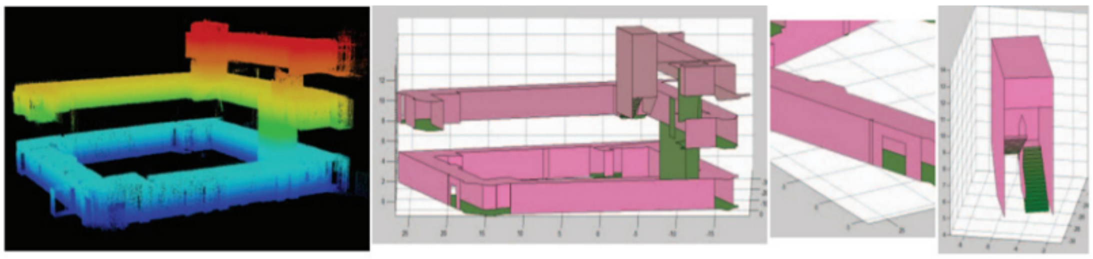


Figure 1.6: Input data on left and reconstructed planar 3D Models of the planes and staircases detected [33]

[8] also proposed a sweeping plane algorithm for detecting vertical and horizontal segments in Manhattan world environments. Here, the density of the point cloud is an input that discretizes and sweeps the environment to detect surfaces such as walls, floors, and ceilings. Later, it uses cell decomposition to identify indoor and outdoor spaces.

However, this cannot be generalized in a non-Manhattan setting and also cannot detect specific interior features such as doors and staircases.

[28] projected the points on the Z-axis to find ceilings and floors and then used Hough transforms to find the walls. The procedure presented would divide point clouds based on height, slicing them and classifying these points into either clutter or permanent structures. While effective for some applications, this method does not infer structural relationships between segments, such as detecting openings, and is better suited to generate a floor plan rather than complete 3D models.

Further building on these approaches, [3] and [41] proposed methods that focused on handling cluttered data and reconstructing occluded walls. Their approach tends to perform ray casting to create the occupancy map, labeling wall surfaces as occupied (walls, for example) or empty-space surfaces (for example, windows). In any case, their model is difficult to handle with complex features such as arched windows or doors with moldings. Mislabeling occluded closed doors as solid walls also happened. Besides occlusion issues in general, indoor environment reconstruction usually shows substantial occluded data due to clutter.

[23] proposed a cell decomposition technique for cluttered point sets in non-Manhattan scenes. They applied ray casting to detect walls and then generated a 2D map by the intersection of wall candidates. This approach managed to compute the polyhedron of each room independently, although it had some problems with openings and low ceilings as shown in Figure: 1.7.

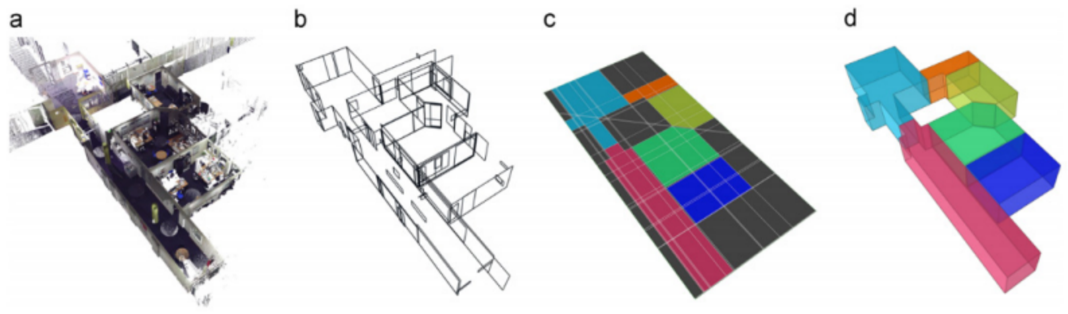


Figure 1.7: The main phases of [23] for cluttered point clouds, showing the final room polyherda

Finally, [6] applied a combined shape grammar approach to reconstruct interiors from point clouds. They divided indoor spaces into rooms and hallways and applied grammar rules to reconstruct the environment. The model benefits from using neighborhood relationships and a priori probabilities for detecting room connections, making it suitable for conversion into BIM models. However, the model lacks specific details about wall openings or areas with sparse data.

While there are many limitations, planar-based approaches have turned out to be powerful tools for indoor reconstructions, especially when the data presents significant noise and occlusions. However, their reliance on surface-based representations makes it difficult to integrate them into BIM models using volumetric representations for indoor navigation and services.

Volumetric-Based Reconstruction

Most of the volumetric-based methods for indoor 3D reconstruction address the partition of space into cells or volumetric primitives, such as cuboids, to represent structures such as walls and rooms.

[19] presented a method for cuboid detection in indoor environments assuming its interior is composed of a set of mutually intersecting cuboids. They first apply the RANSAC algorithm proposed by [34] for detecting planes that define the cuboids with 9 parameters: scale, translation, and rotation. It identifies the optimal cuboids by reconstructing a graph of adjacent planes as displayed in Figure: 1.8. While robust against noise, this method struggles with cluttered data and non-Manhattan world environments. This method also fills the gaps such as doors or windows.

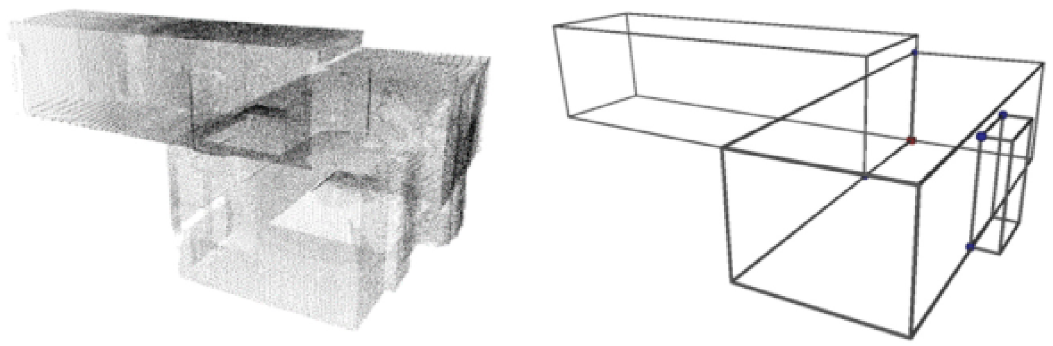


Figure 1.8: Left side: Input data and right side: reconstructed model from cuboid primitives detection and shape graph [19]

[40] represented a Constructive Solid Geometry (CSG) approach. As shown in Figure: 1.9, the model stacks 2D models to 3D by detecting rectangular primitives in point clouds' horizontal slices. Their method, restricted to cuboids, adds textures and reconstructs walls but without detecting openings or doors. Although operates on some non-Manhattan world interiors, this technique does not have any automated quality control, and accuracy has to be checked manually.

[27] used "space partitioning" and "primitive extraction" to reconstruct indoor spaces. Horizontal slices of point clouds are analyzed using Hough transformation, and empty/-solid space labeling is applied through energy minimization, solved by graph cuts. The outcome is a 3D model made up of stacked cells labeled as solid or empty (Figure: 1.10). Their method handles non-Manhattan worlds and clutter but does not detect features like doors or windows. A ray casting method is used which labels solid and empty spaces (Figure: 1.11).

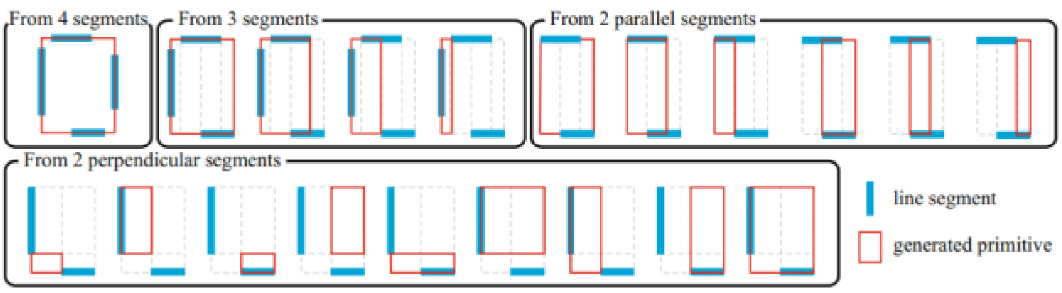


Figure 1.9: Detection of rectangular primitives using line segments [40]

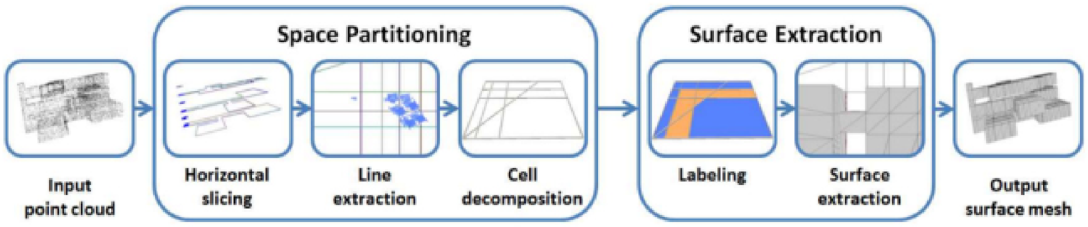


Figure 1.10: Point clouds undergo space partitioning which outputs cell decomposition. These cells are then labeled as empty and solid where solid cells reconstruct permanent structure. Finally, all horizontal slices are stacked to form the 3D model [27].

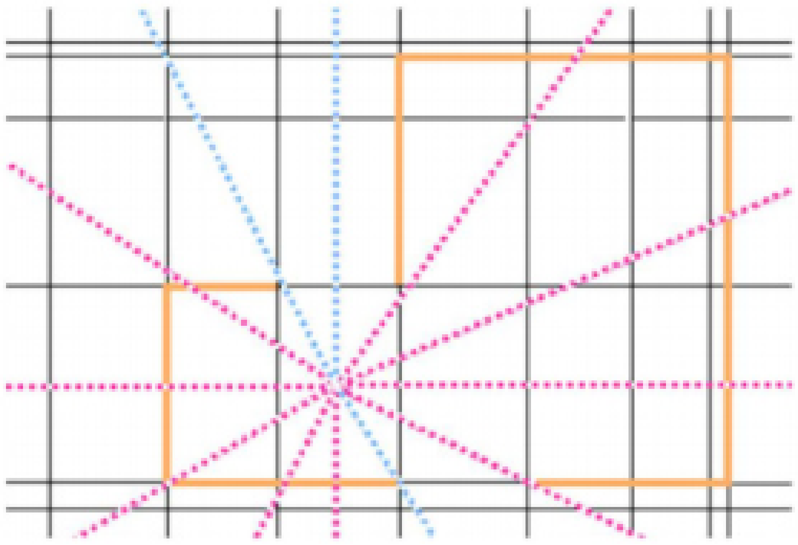


Figure 1.11: Ray-casting method is applied to define empty and solid cells. Walls or edges are marked as orange lines, cells as black lines, and the odd number of intersections as pink lines which represent that the point is in an empty cell while an even number of intersections are represented as blue lines if the point is in a solid space [27].

[20] applied "Palladian Grammar" using cuboids to reconstruct Manhattan world interiors. The method selects the transformation parameters to choose cuboid primitives and grammar rules for placing and merging the cuboids. This still fails when one is dealing with cluttered scenes or those that are not perfectly perpendicular, or even occluded data.

[26] presented an energy-minimization-based approach for detecting volumetric walls and rooms by performing clustering of the input point cloud by room. They detect vertical planes and generate candidate walls to form a 2D planar graph represented in Figure 1.12. This method represents the reconstructed walls as volumetric objects with centerlines, which is helpful for BIM applications. However, their method is sensitive to occluded walls and mobile laser scans when rooms are scanned through windows or corridors. Its detection of wall thickness and openings is an essential ability in the reconstruction of topological relationships.

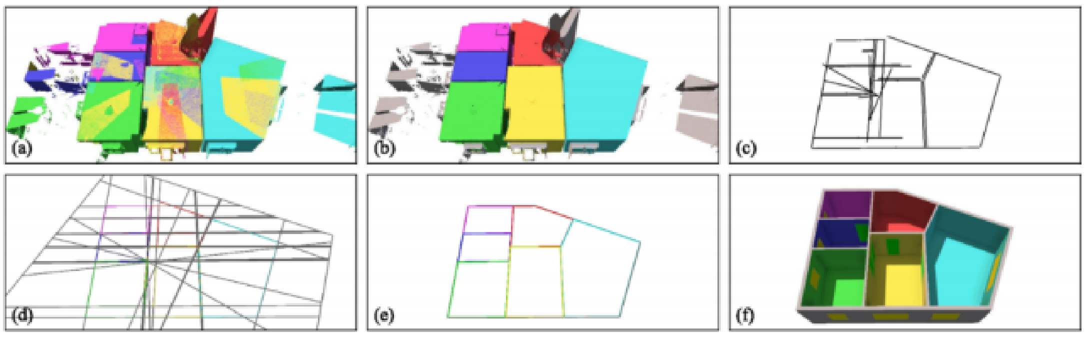


Figure 1.12: (a) The input point cloud is color-coded to show the points assigned to different scans; (b) the segmentation results refine these points per room; (c) vertical planes are projected in 2D; (d) the intersected lines represent potential walls in various colors, dividing space into inside and outside; (e) edges around rooms remain; (f) the final model displays walls, with doors in green and windows in yellow. [26].

Mesh-based Reconstruction

The mesh-based approaches are good in surface reconstruction, object recognition, and finally for rendering in the 3-D environment. These methods range from indoor objects up to building facades [32]; [13]. For example, in [32] a hybrid map approach provided a high accuracy in reconstructing 3D kitchen models for various kinds of surfaces such as walls and furniture. In general, the mesh-based approaches are well-suited for surface modeling and rendering but not practical for semantic labeling and topological relations within complex indoor spaces.

Grammar-Based Reconstruction

Grammar and shape grammar approaches have been widely applied in architecture and computer graphics regarding geometric modeling and reconstruction. They are based on grammar at the level of language, where rules determine the structure of sentences. In this context, "shape grammar" was coined by [35] to produce the 2D and 3D shapes. A highlighted application is the "Palladian Grammar", developed by [36] which models Palladio's architectural designs. It defines rules for symmetric villa plan generation laid out along the Cartesian axis.

[20] have applied this grammar to the reconstruction of Manhattan World interiors using cuboid-based rules without applying rotations, focusing on translation and scaling transformations. They employed connect and merge rules to align the walls and reconstruct the interiors of buildings.

The "Split Grammar" method of Wonka et al. [39] in the "Instant Architecture" paper brings in automation of building design based on enhancements to the parametric grammar approach. This method simplifies shape derivation by splitting basic shapes such as cuboids, cylinders, or prisms into smaller components. "Control Grammar" plays a key role by restricting rule application to ensure consistency with architectural principles. This system follows a three-tiered process in design generation: the inclusion of split grammar, grammar regulation for rule selection, and the employment of an attribute matching system to ensure that randomness is handled according to the users' specifications. The "basic shape" is represented as (b), centered on the origin, and defined by its edges that enable decomposition and transformation. Figure: 1.13 shows how Split Grammar applies to the reconstruction of a building facade. For example, if it is provided with a simple building shape with a certain height, width, and depth, the rules will split the facade into rooms, entrances, and windows. This is done by splitting at predefined axes into different elements constituting a building. The grammar specifies rules such as:

- R1: BUILDING → Subdiv(Z, 0.3, 3.5, 3, 3, 3) {band | GROUNDFLOOR | FLOOR | FLOOR | FLOOR}
- R2: GROUNDFLOOR → Split hoz(X, 4.5, 4, 4.5, 4.5) {ROOM | ENTRANCE | ROOM | ROOM}
- R3: ENTRANCE → Split ver(Z, 0.0, 2.4, 0.6) {∅ | extdoor | wall}
- R4: ENTRANCE → Split hoz(X, 0.5, 3, 0.5) {wall | extdoor | wall}
- R5: ROOM → Split hoz(X, 2.25, 2, 2.25) {wall | WINDOW | wall}
- R6: WINDOW → Split ver(Z, 0.2, 2, 0.2) {frame | glass | frame}
- R7: WINDOW → Split hoz(X, 0.2, 1.60, 0.2) {frame | glass | frame}

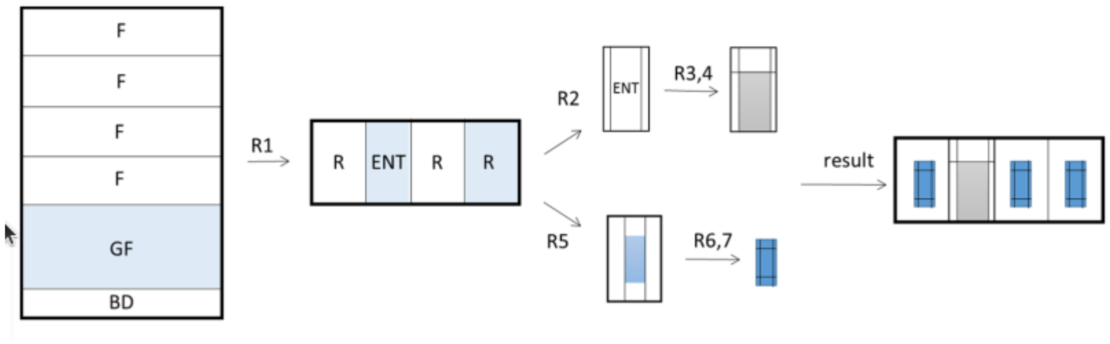


Figure 1.13: Example of Split Grammar applied to a building facade with room and window generation. [39].

Applications of split grammar are found throughout "procedural modeling". [24] applied split grammar to help design complex buildings and city models automatically, while [25] made use of it for facade reconstruction from images.

[5] generalized the previous methods to work with LiDAR data while performing indoor detailed modeling. [7] combined split grammar with Lindenmayer Systems (L-Systems) for hallways and room modeling. Non-corridor areas are split using Split Grammar and L-Systems for generating hallways. Figures 1.13 and 1.14 show some examples of how split and multi-split rules can be utilized to generate realistic structures of rooms from the input point cloud.

[6] expanded on this work by using grammar rules to automatically generate building layouts without predefined input. They derive grammars from the real world directly, rather than manually predefined rules, and that allows flexibility in the approaches for varied building types. However, their approach tends to break when considering non-Manhattan World cases, especially with complex geometries and building axes that aren't well-defined.

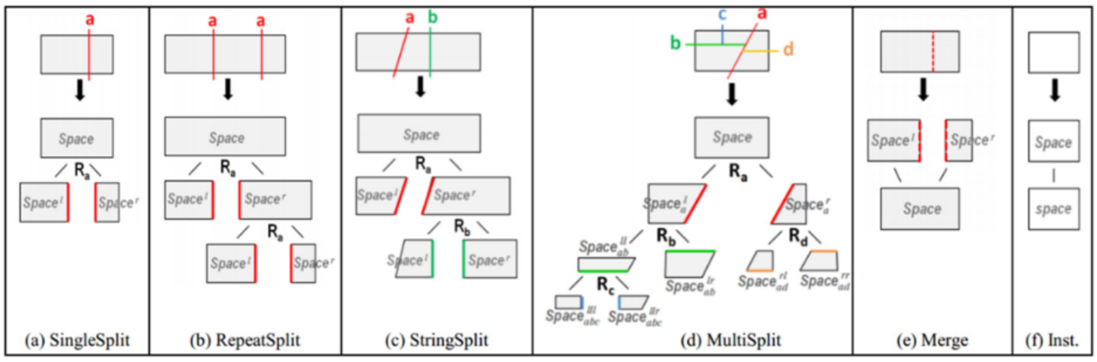


Figure 1.14: Six split rules as shown in [7], demonstrating various partitioning methods like RepeatSplit and MultiSplit.

Detection of Openings

Opening detection is an important task, for doors and windows, in the creation of indoor reconstructions. In fact, for applications that are related to evacuation planning, the identification of these features and the extraction of their geometry can enrich the model semantically, providing information about the escape routes or door functionality. They detect wall surfaces with the analysis of histograms, then they model the surface with voxels. The principal contribution is the "occlusion labeling" step, wherein voxels get labeled as occupied, empty, or occluded as observed in Figure: 1.15.

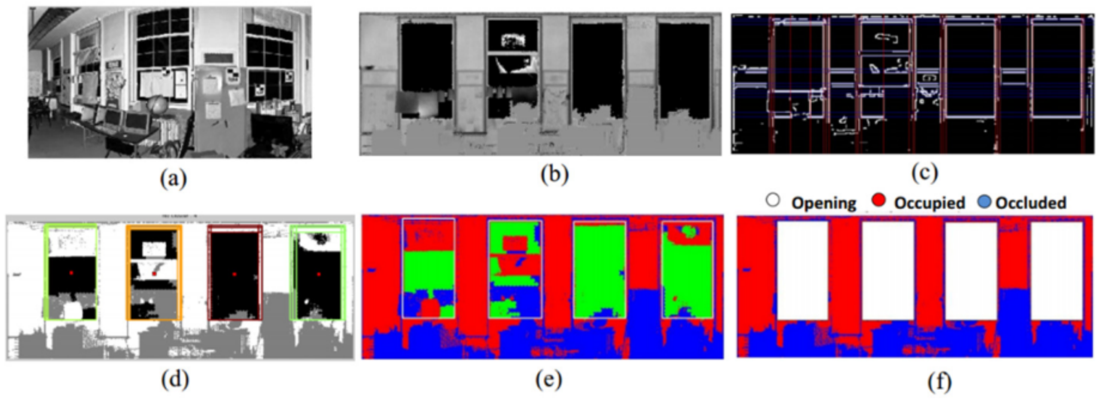


Figure 1.15: Opening detection process. (a) Reflectance image, (b) Depth image, (c) Edge detection, (d) Opening detection using Support Vector Machine (SVM), (e) Prototype openings, (f) Final labeling, where white represents openings, blue is occluded, and red is solid surfaces [3])

Ray tracing allows the detection of voxels that are empty or occluded. After the reconstruction of a range image with the depth values of the surface, Canny's algorithm, and Hough Transform for detecting edges are used, following adjustments in occluded voxels according to detected empty space. Their approach was quite accurate and attained 93.3% of openings correctly identified; however, their method had problems when non-rectangular openings were concerned.

However, this might fail at closed doors, which can be mislabeled as occupied. [11] improved the previous work by using imagery for detecting closed doors through edge detection and a Hough Transform approach. This had an accuracy of up to 95% in some cases; nevertheless, it did not work well where similar textures were available.

Besides, Ray-casting-based techniques for opening detection have been adopted in several works such as [23], [26], and [30] targeting stationary terrestrial laser scanners. In the case of a mobile scanner, where the laser is moving, a more complicated approach should be developed to keep track of which points are measured from which position of the scanner as time progresses.

1.2.3 Open Issues and Conclusion

We can summarize current research issues about 3D indoor reconstruction as follows:

Many of them are restricted to the detection of fundamental shapes such as planes and cuboids, which restricts the reconstruction of more complicated architectural features. Most methods assume horizontal floors and ceilings, and vertical walls. However, these methods are unable to account for non-standard architectural forms, such as curved or sloped surfaces that are frequently found in concert halls, airports, and other locations. While these approaches show promise, there are currently no well-developed solutions for such architectural complexities.

Furthermore, the best approaches usually function in Manhattan-World situations when the walls are perpendicular. It's still difficult to adapt these techniques to non-Manhattan structures, though. While some methods address irregular wall orientations by using primitive detection and cell decomposition, they are still insufficient for complex interiors with irregular geometry. While techniques like Computer Generated Animation (CGA) or split grammar may help alleviate this to some extent, volumetric data or point clouds are difficult to use as direct inputs in current software.

Indoor environments, often have occluded or cluttered data by their very nature, which reduces the accuracy of the reconstruction process. Occlusion is usually handled with techniques like occupancy maps, few techniques focus on reconstructing fully occluded areas [3]. This problem escalates when considering mobile laser scanning since different scanner trajectories add to the already cluttered data [41].

Furthermore, methods based on shape grammar introduce challenges, especially since they often rely on manually defined rules by experts, which is labor-intensive. Though papers like [25] and [7] have tried to find the rules automatically from data, such methods still strongly depend on the quality of the data. Deriving correct geometric relationships like collinearity or parallelism becomes increasingly challenging with cluttered point clouds. Furthermore, the correct assignment of rules to the detected shapes is important. The same shape with different rules can yield different results, which is not acceptable in practical applications like evacuation planning, wherein precision is indispensable.

Model consistency—particularly ensuring both topological and semantic correctness—has not been sufficiently explored. [16] discussed this issue in their research, but more work is needed to guarantee that generated models maintain connectivity and navigability. Consistency control is vital to ensuring that reconstructed spaces, such as rooms and exits, are linked correctly.

Another important challenge is, that accuracy control of the models is difficult since, in most studies, complete ground truth for validation does not exist. Although some manual checks will still be necessary, future research should develop automatic accuracy control methods that verify the accuracy of reconstructed models against real-world data.

These research gaps highlight the need for scalable methods that can effectively address the complexities of various indoor environments. To fill these gaps, my research combines elements of planar- and volumetric-based techniques with grammar-based reconstruction to create a hybrid model that can handle the complexity of interior environments. To further enhance existing approaches, this thesis incorporates script-based commands and transformers. Transformers offer mechanisms of attention that allow the model to record complicated spatial relationships, hence helping with the handling of difficulties brought about by sparse and occluded input. The generation of structured script-based commands for producing intricate, high-fidelity interior reconstructions presents a more scalable and adaptable method. Using such a sequence-to-sequence generating mechanism massively outperforms previous approaches that perform poorly when dealing with complex geometries, occlusions, and sparse data.

1.3 Research Question

Correctly reconstructing indoor places in 3D is difficult, particularly when addressing the essential features of the scene that enable robots to orient themselves and function within the space. There are also differences in the indoor spaces' complexity, dimensions, and design. How can we construct a system that will be able to reconstruct the above-mentioned components efficiently in hundreds of discrete spaces and environments with zero or minimum manual work?

Here, the following research questions are raised:

- There might be a scarcity of 3D data, but how would we make use of this kind of data such that salient components, like walls and doors with windows, all are touched upon?
- How could a reconstruction system of this kind be applied to indoor environments with manifold geometrical settings while remaining within specified accuracy and flexibility?

1.4 Objectives

The main objective of this thesis is to present a system that allows for a fully automated dense 3D reconstruction of indoor scenes based on point cloud data. The specific objectives, in correspondence with the main objective, are listed as follows:

- **Design a dense 3D reconstruction system:** A system able to capture an indoor scene and reconstruct it with high accuracy, mostly the main structural components like walls, doors, and windows. It will, in turn, include detailed 3-D models ready for further processing, modeling, or visualization.
- **Generate human interpretable commands for scene reconstruction:** This algorithm should output high-level, scriptable commands to represent the key architecturally relevant components and their spatial relations.
- **Guarantee Scalability and Automate:** In this project, the system will be trained to cope with a wide variety of indoor locations, from small rooms to the

most complex constructions, using 100,000 different multi-room interior scenes that constitute the Aria Synthetic Environments (ASE) Dataset [4]. Since the reconstruction process was trained on data containing rooms in various configurations, it can work, with few modifications, in a wide range of locations.

- **Create practical 3D models:** The created detailed 3D meshes can be used directly in a virtual model, simulation, and any other application where the interior environment is to be reconstructed.

1.5 Training Dataset

I trained my model on the Meta ASE dataset, which is procedurally generated and consists of 100k unique interior scenes. Originally developed and used by SceneScript, Meta [4] for scene understanding, the ASE dataset is particularly suited to my focus on reconstructing spatial features and generating structured scene scripts. The dataset includes point cloud data and a CAD-like command language designed to describe the structural elements clearly. The language consists of commands such as `make_wall`, `make_door`, among others that perfectly align with the goal of my model. Figure: 1.17 shows language commands used in the scripts.

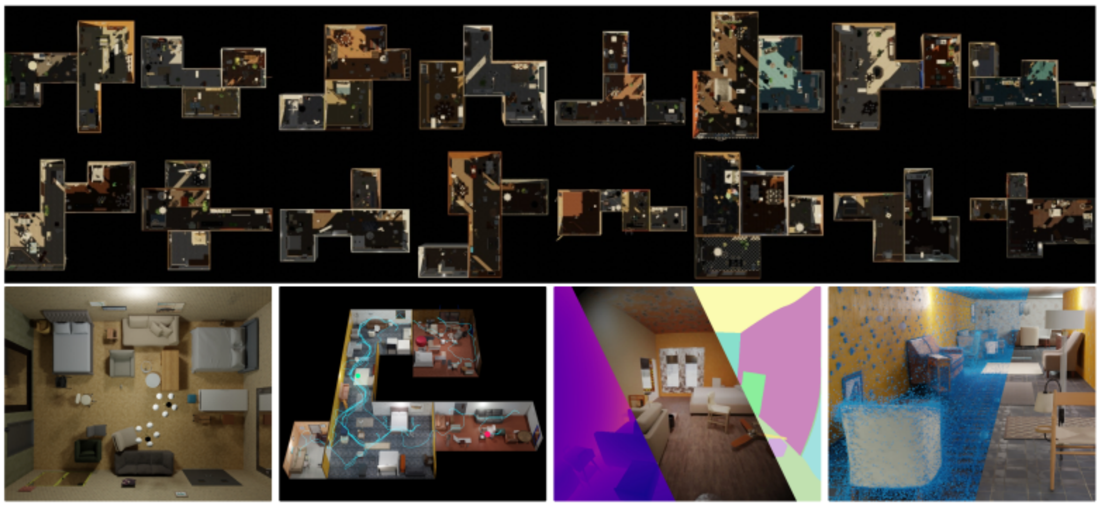
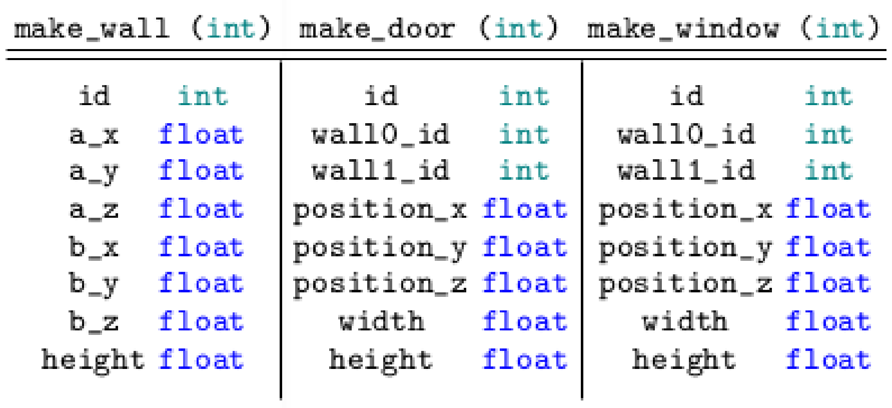


Figure 1.16: Visualization of the ASE Dataset with its many 3D scenes [4].

Figure 1.16 offers a visualization of the ASE dataset, showcasing its rich point cloud data and the variety of interior scenes it covers.

1.6 Environment and Initial State

The work presented in this thesis was initiated by building a 3D reconstruction model from scratch, based on the Project Aria ASE dataset [4]. For this work, the dataset



make_wall (int)		make_door (int)		make_window (int)	
id	int	id	int	id	int
a_x	float	wall0_id	int	wall0_id	int
a_y	float	wall1_id	int	wall1_id	int
a_z	float	position_x	float	position_x	float
b_x	float	position_y	float	position_y	float
b_y	float	position_z	float	position_z	float
b_z	float	width	float	width	float
height	float	height	float	height	float

Figure 1.17: Complete set of structured language commands that detail the architectural layouts of the dataset. [4]

provided several input modalities. Among these were point clouds and RGB images. After discussing it with my supervisors, we decided to focus on the latter since they had the richest spatial data to reconstruct the main structural features of interest.

To begin with point cloud processing, I chose TorchSparse [37] for its capability of offering real-time feature extraction. It is the best option for efficient and real-time testing of the model. Using TorchSparse, the point cloud data was processed into sparse tensors so that one can effectively handle big indoor environments. Plotly was used for visualization through which I could view interactively the pre-processed data and the outputs of the model in development.

The model output is the script generated by the model, describing the object placements and dimensions within the scene. Based on discussions with my supervisors, we agreed on the form this output should take such that it would be practical and interpretable for real-world use. The rendering techniques and model architecture are inspired by SceneScript [4], a framework developed by Meta. Decisions regarding the above aspects, including their adaptation in this work, were based on discussions with my supervisors.

Some key decisions were taken at the beginning of the project to include in the main area of focus: indoor environments, where unique challenges lay ahead, including dynamic elements and structural complexity.

The whole development of the model was conducted by me, from data processing to architectural design and rendering techniques, with inputs from the supervisors during the technical discussions on advanced topics like handling point cloud data and performing real-time 3D reconstructions.

CHAPTER 2

Planning and Resources Evaluation

Contents

2.1 Planning

Figure: 2.1 shows a Gantt chart that highlights the timeline of the project. It gives an overview of the flow of all the tasks and how much time each task is consuming. Table 2.1 complements Figure: 2.1 with great detail regarding all the tasks and subtasks and the relationship between them.

As the project developed, different versions of the plan were generated. In fact, from the beginning, the schedule was changed several times to reflect research by SceneScript [4] that revealed limitations in utilizing RGB images alone and the need to switch to point cloud data. Also, issues surrounding voxel size and choices of positional encoding contributed to changing the timeline.

Using point cloud data directly provides 3D spatial representation and avoids the drawbacks associated with RGB images, like poor depth estimation. Changing to this more informative representation involved several adjustments, including the extended time now used in the processing of the point clouds, feature extraction, and creating a sparse tensor representation; The moving from sinusoidal to trainable positional encoding involved additional experimentations and model tunings.

Initial experiments demonstrated that the performance of the model strongly depends on the voxel size: the smaller the voxel size, the more detailed the information; but this

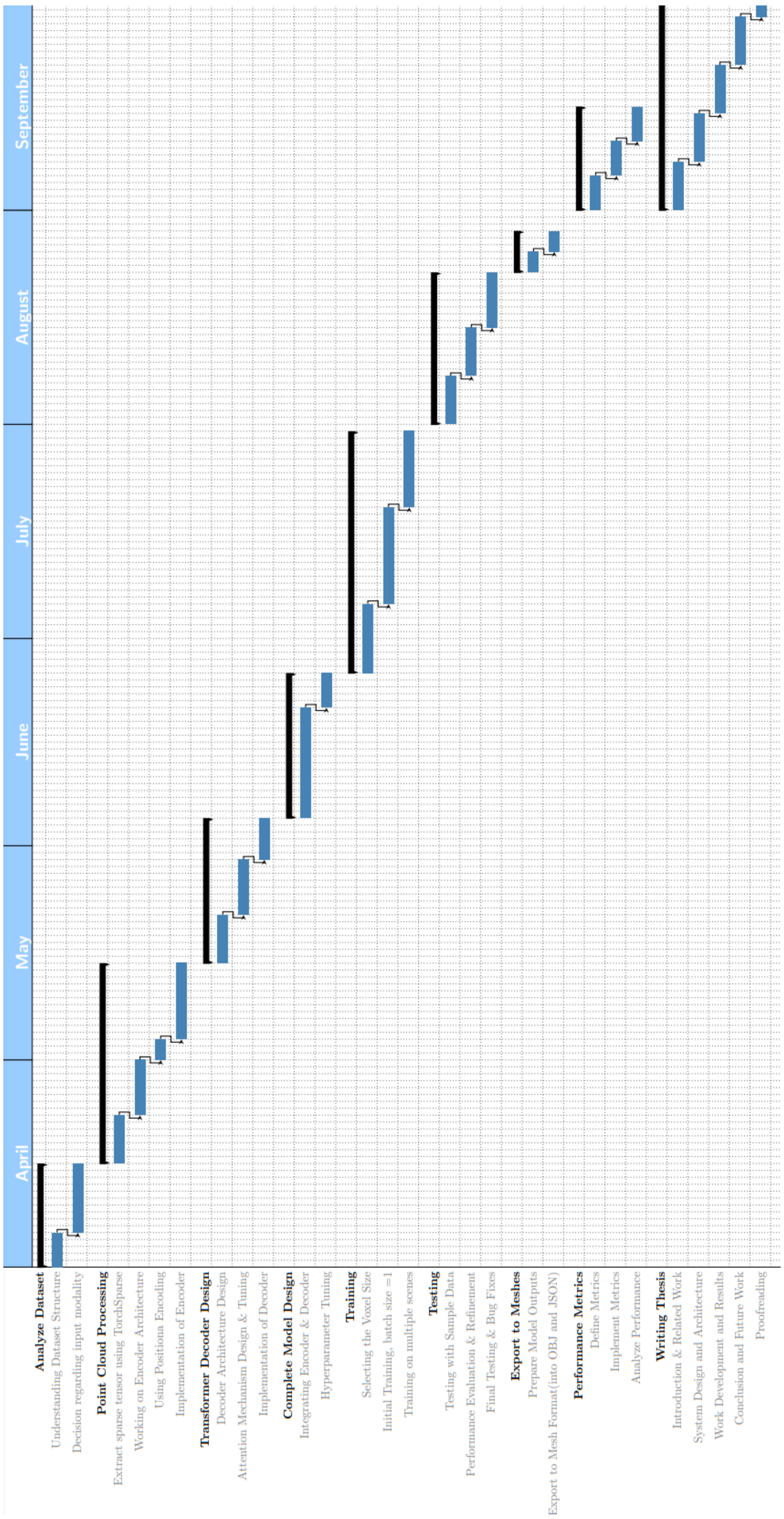


Figure 2.1: Gantt chart representing an overview of project timeline

increases memory consumption and prolongs training. Therefore, another important factor during model development and testing was finding a balance between information detail and computational efficiency.

While considering these changes, the project still had a smooth path because some tasks were being rescheduled and extended when needed. These modifications, though deviating from the original plan, are fundamental for the optimization of the performance of the model in 3D scene reconstruction tasks.

2.2 Resource Evaluation

The resources used during this project, both concern computational infrastructure as well as the dataset used for training. The computation facilities provided by the Institute for Artificial Intelligence at the University of Bremen are:

- **Lab computers**: These have been extensively used for performing software development, debugging, and model refinement in this project.
- **GPU-powered system**: Training the 3D reconstruction model required a Graphics Processing Unit (GPU), as it relies on efficiently handling large-scale point cloud data. The GPU system greatly reduced the training time and allowed faster experimentation during the development process.

Additionally, the **ASE dataset** [4] played a crucial role during training and testing processes. It provided large numbers of point cloud data from synthetic environments with the necessary 3D spatial context for the process of indoor scene reconstruction.

WBS #	Name / Title	Type	Start Date	End Date
1.1	**Analyze Dataset**	Group	2024-04-01	2024-04-15
1.1.1	Understand Dataset Structure	Task	2024-04-01	2024-04-05
1.1.2	Decision regarding input modality	Task	2024-04-06	2024-04-15
1.2	**Point Cloud Processing**	Group	2024-04-16	2024-05-14
1.2.1	Extract sparse tensor using TorchSparse	Task	2024-04-16	2024-04-22
1.2.2	Working on Encoder Architecture	Task	2024-04-23	2024-04-30
1.2.3	Using Positional Encoding	Task	2024-05-01	2024-05-03
1.2.4	Implementation of Encoder	Task	2024-05-04	2024-05-14
1.3	**Transformer Decoder Design**	Group	2024-05-15	2024-06-04
1.3.1	Decoder Architecture Design	Task	2024-05-15	2024-05-21
1.3.2	Attention Mechanism Design & Tuning	Task	2024-05-22	2024-05-29
1.3.3	Implementation of Decoder	Task	2024-05-30	2024-06-04
1.4	**Complete Model Design**	Group	2024-06-05	2024-06-25
1.4.1	Integrating Encoder & Decoder	Task	2024-06-05	2024-06-20
1.4.2	Hyperparameter Tuning	Task	2024-06-21	2024-06-25
1.5	**Training**	Group	2024-06-26	2024-07-30
1.5.1	Selecting the Voxel Size	Task	2024-06-26	2024-07-05
1.5.2	Initial Training, batch size = 1	Task	2024-07-06	2024-07-19
1.5.3	Training on multiple scenes	Task	2024-07-20	2024-07-30
1.6	**Testing**	Group	2024-08-01	2024-08-22
1.6.1	Testing with Sample Data	Task	2024-08-01	2024-08-07
1.6.2	Performance Evaluation & Refinement	Task	2024-08-08	2024-08-14
1.6.3	Final Testing & Bug Fixes	Task	2024-08-15	2024-08-22
1.7	**Export to Meshes**	Group	2024-08-23	2024-08-28
1.7.1	Prepare Model Outputs	Task	2024-08-23	2024-08-25
1.7.2	Export to Mesh Format (OBJ & JSON)	Task	2024-08-26	2024-08-28
1.8	**Performance Metrics**	Group	2024-09-01	2024-09-15
1.8.1	Define Metrics	Task	2024-09-01	2024-09-05
1.8.2	Implement Metrics	Task	2024-09-06	2024-09-10
1.8.3	Analyze Performance	Task	2024-09-11	2024-09-15
1.9	**Writing Thesis**	Group	2024-09-01	2024-09-30
1.9.1	Introduction & Related Work	Task	2024-09-01	2024-09-07
1.9.2	System Design and Architecture	Task	2024-09-08	2024-09-14
1.9.3	Work Development and Results	Task	2024-09-15	2024-09-21
1.9.4	Conclusion and Future Work	Task	2024-09-22	2024-09-28
1.9.5	Proofreading	Task	2024-09-29	2024-09-30

Table 2.1: Project Schedule

Chapter 3

System Analysis and Design

Contents

This chapter presents the requirements analysis, design, and architecture of the proposed work.

3.1 Requirement Analysis

Core functionalities entailed in the system include raw point cloud data processing, transformation of the data into sparse tensors, and generation of final scripts that can be used in rendering 3D scenes. There are various stages involved in realizing the functional and non-functional requirements of the work, which range from data preprocessing to the generation of scripts and assessment of performance.

3.1.1 Functional Requirements

The salient features of the model are reviewed in this section. First is the encoder, which takes the raw point cloud data and transforms it into a sparse tensor using TorchSparse [37]. The encoder further uses this tensor with positional encoding so that the scene's spatial information is preserved. These encoded features do not feed directly into the decoder but rather are first processed through several 3D convolutional layers extracting higher-level geometric features, detailed in Table: 3.2. Such features are used in the second functionality where the decoder autoregressively predicts an output script. It is done by generating each command one at a time to make sure the elements of the structure, like walls, doors, and windows, are correctly reconstructed.

As a pre-processing step for memory efficiency, the point cloud is transformed into a sparse tensor according to Table 3.1; thus, it can facilitate faster computations while retaining the essential details in spatiality. Such an encoding ensures that relationships that may exist between different architectural elements are preserved.

Input:	Raw point cloud data
Output:	Sparse tensor
In this stage, point cloud data is transferred to a sparse tensor with the library of TorchSparse.	

Table 3.1: Functional requirement «Conversion of Sparse Tensor»

This is followed by encoding. The features are positionally encoded, and then some successive 3D convolutional layers extracting geometric details compress the information in space into a high-level feature representation. Explanation in detail is given in Table: 3.2.

Input:	Sparse tensor
Output:	Encoded features
Several layers of 3D convolution are applied on the sparse tensor to extract geometric features. This reduces the point cloud into a deep higher-order representation of features that can maintain the spatial relationships of walls, doors, and windows.	

Table 3.2: Functional requirement «3D Convolution Encoding»

The second functionality is that of script generation, which is a transformer-based decoder making use of the encoded features to generate a structured script. Such commands as "make_wall" and "make_door" are predicted autoregressively based on the previous commands. It allows for accurate and clear scene descriptions, as outlined in Table: 3.3.

3.1.2 Non-functional Requirements

The non-functional requirements for this work are as follows.

- The system should positionally encode features into the system with accurate preservation of the spatial relationships between elements in the 3-D scene.
- The mapping of the vocabulary to the output script should be exact, and command names such as "make_wall", "make_window" and "make_door" should be appropriately generated.

Input:	Encoded point cloud and START token
Output:	Generated script with STOP token
Decoder takes the encoded features from the point cloud and generates a script describing the scene. Starting with the START token, it goes on till the STOP token. Each command includes objects like "make_wall" and their spatial parameters.	

Table 3.3: Functional requirement «Script Generation»

- Parse the script correctly to the renderer to ensure that it renders the 3D scene without errors.
- Operations are to be in the normalized frame of reference so that it has consistent scaling of spatial dimensions between different scenes.
- Voxels extent is to be decided appropriately for point cloud processing with an optimal trade-off between accuracy and computation.
- Causal masking should be performed in such a way that every generated command is conditioned on what has already been predicted at training time.
- Real-time rendering of 3D meshes allows for immediate feedback, capturing all the details in the structure.
- Meshes should be exported in file formats like OBJ and JavaScript Object Notation (JSON) for further use by external rendering engines or simulation systems.

3.2 System Design

The system focuses on the generation of detailed 3D scene reconstructions from sparse point cloud data, driven by a data-driven, autoencoder-like approach. The partial point cloud acts as input, which is further encoded into a latent space vector. Subsequent processing of this latent vector through the decoder results in a scene script that comprises commands for the recreation of the layout of the scene.

The design of the system design follows a well-structured flow represented in Figure 3.1. It first voxelizes point cloud data and feeds them into the sparse 3D encoder. Using the library TorchSparse [37], it will transform data into sparse tensors that ensure computational and memory efficiency. In turn, this encoder uses this sparse representation with some 3D convolutional layers for obtaining high-level geometric features and maps into a latent vector. Positional encodings are applied to retain the most important spatial relationships within the data.

A decoder, inspired by the Meta's SceneScript model [4], subsequently takes encoder outputs and autoregressively generates a script that represents scene layout. The commands, such as `make_wall` or texttmake_door, are predicted one after another while the system keeps coherence and accuracy regarding the structure of the scene.

The system is then rendered and visualized in a dense and interactive environment representation after the processing of the script. Converting the scene to a polygon mesh is followed by converting the scene, which can easily be integrated into different real-time rendering engines and simulation environments.

Each part of the system will be discussed in further detail in Section 3.3.

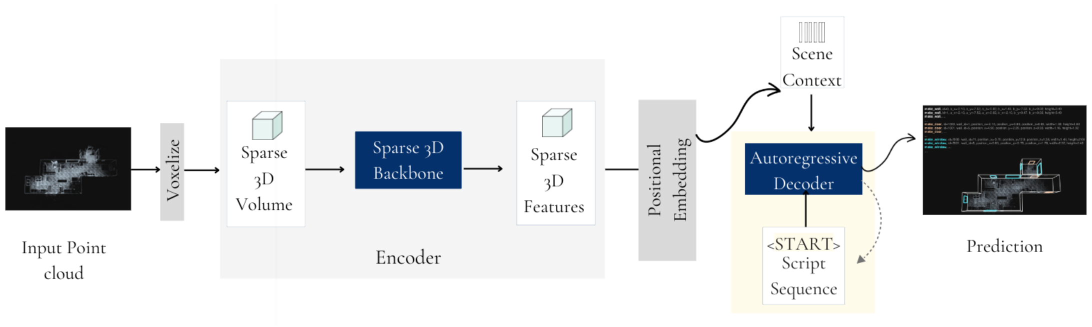


Figure 3.1: High-level overview of the system design.

3.3 System Architecture

The system's architecture is built around an encoder-decoder framework.

3.3.1 Encoder

This is a 5-layer point cloud encoder with 3D convolution, with a kernel size of 3 and a stride of 2. Further, the encoder processes voxelized point cloud data. The point cloud data is first converted to a sparse tensor using the Torchsparse Library [37]. The sparse tensor is then passed through layers of 3D convolutions. After every convolutional layer, the features are downsampled using max-pooling layers. Meanwhile, positional encoding is proposed to keep the spatial information in that process. This architecture is also shown in Figure: 3.2.

3.3.2 Decoder

The autoregressive decoder consists of 6 transformer layers, each of which includes multi-head self-attention, cross-attention, and feedforward networks. This model makes use of 6 multi-head self-attention and cross-attention heads per layer while processing the decoder sequence. The cross-attention is used to align queries in the decoder to outputs from the encoder, ensuring that commands are grounded in the features of the encoded point cloud. Each of these layers is implemented as a feedforward network with a dimensionality of 2048, projecting its inputs from a 512-dimensional space, applying a Rectified Linear Unit (ReLU) activation, and projecting back to 512 dimensions.

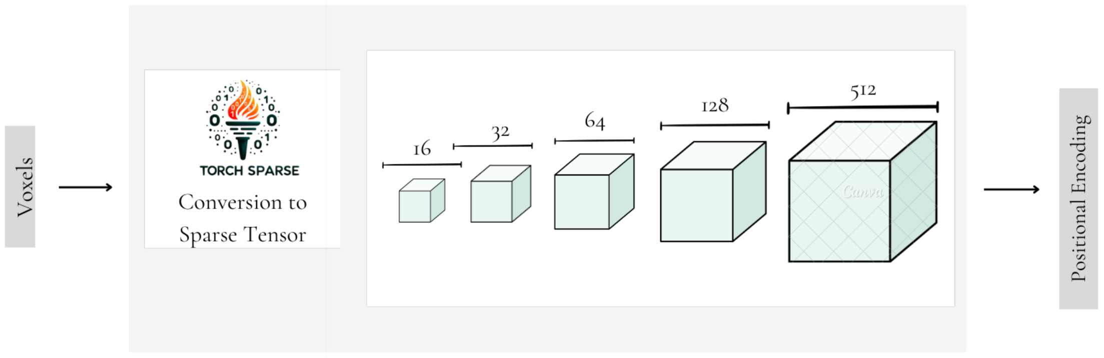


Figure 3.2: Encoder Architechture

The decoder outputs structured commands like make_wall, make_door, and make_window, where each line in the output predicts both the type of command involved and its geometric parameters, which include position, width, and height.

The model used here for the decoder architecture is very much similar to that in [38], except that it involves both multi-head as well as cross-attention mechanisms while the one mentioned in the paper [38] relies purely on multi-head attention shown in Figure: 3.3.

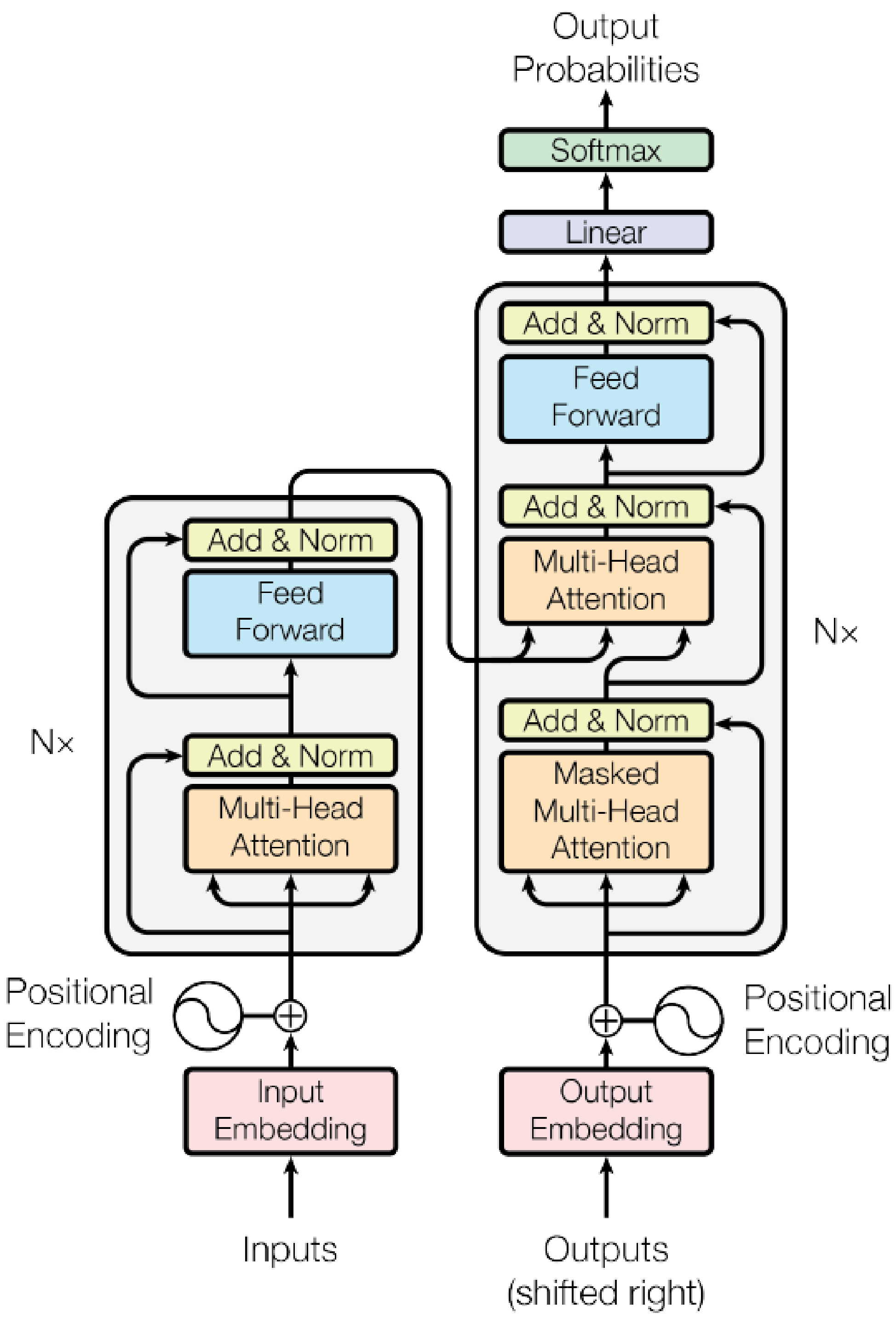


Figure 3.3: Decoder Architecture of the System [38]

Chapter 4

Work Development and Results

Contents

The developed work and the obtained results are made explicit in this chapter. All possible deviations from the initial planning are detailed here and justified.

4.1 Point Cloud Processing

I first focused on RGB images as the input modality because they encoded rich visual and texture information, promising for 3D environment reconstruction. However upon observing the performance of the model using this modality from "Scenescript" paper [4], where the performance was much below expectations, especially concerning depth estimation and spatial accuracy. The limitation is very much due to the inherently 2D nature of RGB images, providing just the projection of 3D objects onto a plane.

The problem with using RGB information at the core was that it required the model to infer depth from visual cues such as shading and occlusion, which, though possible, provided ambiguities of the scene. Mapping from 2D mathematically poses an under-constrained problem: for RGB images, we solve for three spatial dimensions based on only two observable coordinates. That has inherently noisy and uncertain captures of exact structural relationships between objects.

Having observed these limitations, I moved on to point cloud data. This provided 3D coordinates directly and avoided indirect depth inferences.

4.1.1 Sparse Tensor Representation

The system begins its processing by taking input in the form of a point cloud $P = \{p_1, p_2, \ldots, p_N\}$, where each point $p_i \in R^3$ is a 3D coordinate. The point cloud data is first discretized by voxelizing the points onto a 3D grid with a given voxel size v (see Figure: 4.1). This voxelization of points is calculated as:

$$V = \left\lfloor \frac{p_i}{v} \right\rfloor \tag{4.1}$$

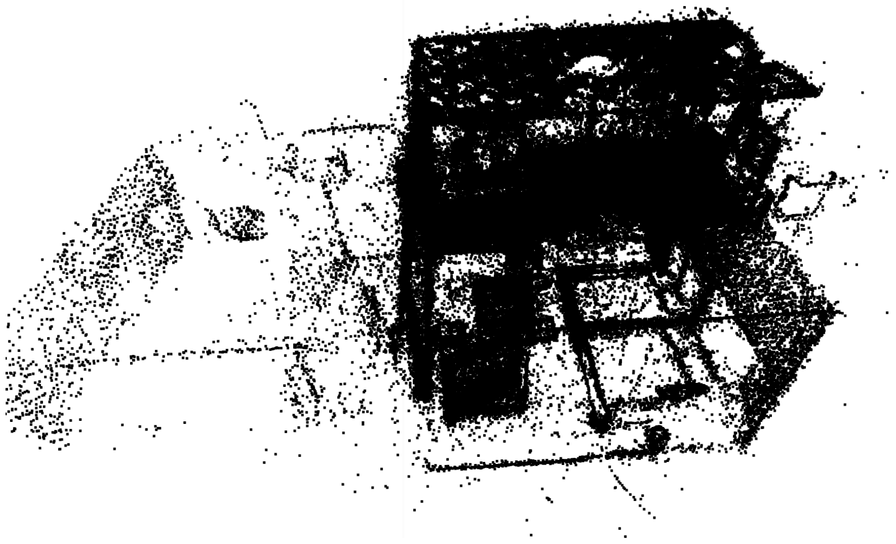


Figure 4.1: After voxelization, the challenge of sparsity of the data can be observed on the left side of the scene as parts of the layout appear to be missing

The features, such as the mean distance inside each voxel, are calculated after voxelization. These features are aggregated in the form of a sparse tensor $S = (C, F)$, where $C \in Z^{K\times 3}$ is the coordinate of the non-empty voxels, and $F \in R^{K\times d}$ is the feature for each voxel. Here, d denotes the feature dimension, including the statistical properties of the points within each voxel.

$$S = \text{SparseTensor}(F, C) \tag{4.2}$$

For the sparse tensor processing, I decided to use the Torchsparse library [37]. Different from the standard dense tensor libraries, operating on large sparse point clouds with Torchsparse will only be applied on voxels that are not empty, hence efficiently reducing computational overhead.

Moreover, the optimized sparse convolution operations in Torchsparse are memory- and speed-efficient, especially for 3D point cloud data where most of the volume in a grid would be empty. This capability of dynamic sparsity, where there could be variation in the density over regions, is very effective for real-time 3D scene reconstruction tasks as well.

4.1.2 Encoder with Convolutional Layers

First, the sparse tensor undergoes a process of feature extraction via a set of 3D convolutional layers. These layers increase the depth of features while gradually reducing the spatial resolution of the input. The convolution process enriches the features in such a way that the number of channels increases after passing through each layer, for example, 16, 32, 64, 128, and 512. Each convolutional block applies a 3D convolution followed by max pooling and can be represented by the following equation:

$$S_{l+1} = \text{MaxPool3D}(\text{Conv3D}(S_l, W_l, \text{stride} = 2, \text{kernel} = 3)) \tag{4.3}$$

The convolution operation over multiple layers can be summarized as:

$$S_{\text{encoded}} = \text{Conv3D}^n(S_0, \{W_1, W_2, \dots, W_n\}) \tag{4.4}$$

Where n is the number of convolutional layers, and W_i denotes the convolutional weights for layer i.

4.1.3 Positional Encoding

Initially, I used sinusoidal positional encoding to encode the spatial information in the point cloud. This method encodes the coordinates of voxels C with a sum of sinus and cosines of different magnitudes, providing a non-trainable yet consistent way of infusing positional awareness. These coordinates are encoded by the equation below:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{4.5}$$

However, I switched to the trainable positional encoding embedding layer. That is mainly because the trainable embeddings can learn positional representations concerning the geometry of the point cloud data. Sinusoidal encoding enforces a hard-wired and predefined structure that cannot adapt to the unique geometric characteristics in the dataset, such as point densities and intricate local structures typically found in indoor environments.

In voxelized point clouds, the data is inherently irregular and sparse, with significant variation in spatial structure. Because of this, the static nature of sinusoidal encoding couldn't capture the details of the local geometry. Switching to trainable positional encoding has been a better adaptation of the model to these variations as the model can encode and make use of the positional information more correctly relevant to each element in the scene.

Therefore, after feature extraction by convolutional layers, I apply learned positional encoding. Voxel coordinates (C) are concatenated with the feature vectors as:

$$F_i \leftarrow \mathrm{cat}(F_i, C_i) \tag{4.6}$$

A learned positional embedding layer is applied subsequently to map the voxel coordinates into a higher-dimensional space as:

$$P_i = W \cdot C_i \tag{4.7}$$

Thus, the final encoded feature vectors, which include both feature and learned positional information, can be represented as:

$$F_{\mathrm{geo}} = F + P \tag{4.8}$$

The output will be a set of feature vectors $F_{\mathrm{geo}} \in R^{K\times 512}$, with each vector retaining the encoded feature and the learned positional information that will be used in subsequent steps.

4.2 Language Decoder

4.2.1 Tokenization

The input tensor $x \in R^{n_{\mathrm{decoder}}\times d_{\mathrm{model}}}$ in the decoder part of the model is tokenized and processed into a sequence of embeddings. These commands are encoded as an enumeration; thus vocabulary is given by:

- `START` $= 1$
- `STOP` $= 2$
- `MAKE_WALL` $= 3$
- `MAKE_WINDOW` $= 4$
- `MAKE_DOOR` $= 5$

This gives a total vocabulary size of 5. For each instruction, an embedding is generated by concatenating a one-hot encoded vector that represents the type of command with its parameters; the resulting input size becomes 1×11. Positional encoding, as explained in the previous section, is added to incorporate spatial information about the sequence.

The model's command generation is built upon three basic layout elements: `MAKE_WALL`, `MAKE_DOOR`, and `MAKE_WINDOW` shown in Figure: 1.17. All three commands have associated parameters that define the geometry of these elements. For example, the complete set of parameters for `MAKE_WALL` defines a gravity-aligned 2D plane while `MAKE_DOOR` and `MAKE_WINDOW` define cutouts within walls including their respective sizes and positions.

Unlike the approach followed in Meta's SceneScript [4], which predicts the command structure parameter by parameter, my model predicts the whole script line. In this line-based prediction approach, the model generates a complete and coherent command, including all its parameters, in a single output. This method not only accelerates the process but also ensures consistency across the generated commands.

4.2.2 Implementation of Attention Mechanisms in 3D Reconstruction

This 3D reconstruction model uses the multi-head attention mechanism in both self-attention and cross-attention to predict command sequences based on point cloud data. Using learned weight matrices W_Q, W_K, W_V, the multi-head attention projects the input tensor $x \in R^{n_{\text{decoder}} \times d_{\text{model}}}$ into queries, keys, and values.

These matrices are used in the scaled dot-product attention mechanism. The query Q_i is multiplied by the transpose of the key K_i, scaled by the square root of the key dimension d_k, and the softmax function is applied.

An output of size $R^{n_{\text{decoder}} \times d_k}$ is produced by each attention head. By concatenating and transforming these outputs into a tensor of size $R^{n_{\text{decoder}} \times d_{\text{model}}}$, the model can handle various segments of the input sequence concurrently.

Self-Attention

My approach applies self-attention to the decoder inputs, enabling it to attend to distinct segments of the input sequence and discern correlations between previous commands and the one it is predicting at that moment. The decoder's self-attention mechanism is calculated as follows:

$$Q_{\text{decoder}} = W_Q \cdot x_{\text{decoder}}, \quad K_{\text{decoder}} = W_K \cdot x_{\text{decoder}}, \quad V_{\text{decoder}} = W_V \cdot x_{\text{decoder}} \tag{4.9}$$

The attention weights are computed as:

$$\text{SelfAttention}(Q_{\text{decoder}}, K_{\text{decoder}}, V_{\text{decoder}}) = \text{softmax}\left(\frac{Q_{\text{decoder}} K_{\text{decoder}}^\top}{\sqrt{d_k}}\right) V_{\text{decoder}} \tag{4.10}$$

This mechanism allows the decoder to understand how previous commands influence upcoming ones in the autoregressive generation process.

Cross-Attention

The decoder can attend to features encoded by the encoder from the point cloud data by using cross-attention in addition to self-attention to align decoder queries with encoder keys and values. The key and value matrices from the encoder output, K_{encoder} and V_{encoder}, are combined with the decoder's query matrix, Q_{decoder}:

$$Q_{\text{decoder}} = W_Q \cdot x_{\text{decoder}}, \quad K_{\text{encoder}} = W_K \cdot x_{\text{encoder}}, \quad V_{\text{encoder}} = W_V \cdot x_{\text{encoder}} \tag{4.11}$$

The cross-attention is then computed as:

$$\text{CrossAttention}(Q_{\text{decoder}}, K_{\text{encoder}}, V_{\text{encoder}}) = \text{softmax}\left(\frac{Q_{\text{decoder}}K_{\text{encoder}}^{\top}}{\sqrt{d_k}}\right)V_{\text{encoder}} \tag{4.12}$$

This ensures that the decoder generates commands that are grounded in the 3D spatial context of the scene by focusing on relevant features from the point cloud data. Figure 1.3 and Figure 1.4 illustrate the workings of both self-attention and cross-attention in my model.

Masking

Masking is used in my 3D reconstruction model to provide appropriate sequential production of commands and relevant feature selection for both self-attention and cross-attention mechanisms.

Masking in self-attention guarantees that the decoder does not pay heed to subsequent tokens when making an autoregressive prediction about the next instruction. The following is how the masking operation is used:

$$\text{MaskedSelfAttention}(Q_{\text{decoder}}, K_{\text{decoder}}) = \text{softmax}\left(\frac{Q_{\text{decoder}}K_{\text{decoder}}^{\top}}{\sqrt{d_k}} + \text{mask}\right)V_{\text{decoder}} \tag{4.13}$$

In cross-attention, masking ensures that only relevant parts of the encoded point cloud data are attended to. By doing this, the model is kept from concentrating on portions of the input that are not relevant, guaranteeing precise command creation based on spatial properties. The calculation of the masked cross-attention is:

$$\text{MaskedCrossAttention}(Q_{\text{decoder}}, K_{\text{encoder}}) = \text{softmax}\left(\frac{Q_{\text{decoder}}K_{\text{encoder}}^{\top}}{\sqrt{d_k}} + \text{mask}\right)V_{\text{encoder}} \tag{4.14}$$

These masking mechanisms help ensure the model attends to the correct tokens and features at each step of the generation process.

4.2.3 Feedforward Network and Final Output

In the model, after the attention layers, a feedforward network projects the input tensor from R^{512} to R^{2048}, applies ReLU activation, and then projects it back to R^{512}:

$$\text{FeedForward}(x) = W_2(\text{ReLU}(W_1x)) \tag{4.15}$$

Each transformer layer normalizes and applies residual connections between the input and the outputs of the self-attention, cross-attention, and feedforward layers.

The final output of the decoder consists of two linear transformations: one for predicting command probabilities and one for predicting associated parameters. The command layer calculates the probability distribution over the command vocabulary:

$$\text{command\_probs} = \text{softmax}(W_{\text{command}} \cdot x) \tag{4.16}$$

The parameter layer calculates the associated parameters of the architectural elements:

$$\text{parameter\_probs} = \tanh(W_{\text{param}} \cdot x) \tag{4.17}$$

Finally, the output is mapped from 512 to 11 (5 commands + 6 parameters) using a linear layer:

$$\text{final\_output} = \text{Linear}(W_{\text{final}} \cdot x) \tag{4.18}$$

4.2.4 Optimized Parameter Prediction

The model was initially designed to predict, for each command, seven parameters comprising identifiers and spatial properties. For every command, such as `make_wall`, `make_door`, and `make_window`, it was predicting parameters of the form `wall0_id` and `wall1_id` of doors and windows. This wasn't necessary, since wall identifiers can always be gathered from the spatial relations between different architectural elements.

The strategy was then optimized to predict six key parameters for every architectural feature except the identifiers of the wall for doors and windows. In this setup, the model focuses on:

- `make_wall`: $\{id, \text{pos\_x}, \text{pos\_y}, \text{angle}, \text{width}, \text{height}\}$
- `make_door` and `make_window`: $\{id, \text{pos\_x}, \text{pos\_y}, \text{pos\_z}, \text{width}, \text{height}\}$

The parameters for walls define the essential spatial and structural properties:

- id: A unique identifier for the wall.
- pos_x, pos_y: The x and y coordinates of the wall's position, defining the horizontal location of the wall in the scene.
- angle: The orientation angle of the wall relative to the coordinate system, defining the wall's rotation.
- width: The width of the wall, representing its horizontal extent.
- height: The height of the wall, with the assumption that walls are straight and aligned with gravity (i.e., vertical).

The parameters of each command specify their spatial properties and dimensions:

- id: A unique identifier for the door or window.
- pos_x, pos_y, pos_z: The x, y, and z coordinates represent the center of the door or window, defining its exact location in the 3D space.
- width: The width of the door or window, defining its horizontal extent.
- height: The height of the wall, door, or window, specifying its vertical extent.

The model uses spatial proximity to infer the relationship between doors, windows, and walls, hence making the process of prediction more effective. The wall association for doors and windows is done based on their predicted positions and their distances to the walls, without explicit prediction of wall IDs. For example, given the predicted coordinates (x_d, y_d, z_d) of a door or window, select the closest wall based on the minimum distance $d_{\min}$.

This optimization reduces redundancy to its minimum and the prediction mechanism is simplified.

4.2.5 Output Format

Scenes are stored as JSON files, where each object is saved in OBJ format. This streamlines the process, enabling other simulation platforms to directly include the scenes in their environment with ease as shown in Figure: 4.2.

4.3 Training

4.3.1 Voxel Size

Voxel size is a significant hyperparameter when training. The voxel size decides the resolution at which point cloud data is processed. First, I tried using the voxel size values 0.03, 0.04, and 0.05 for 20 epochs each with a batch size of 1. The rates of convergence were about the same for all these values of voxel size. As can be seen from Figure: 4.3, the loss function has similar converging rates for all the tested voxel sizes. However, voxel size 0.04 gives an optimal tradeoff between keeping sufficient resolution and saving memory consumption.

Voxel size controls the amount of detail the model can capture in the point cloud. Smaller voxel sizes increase resolution, but demand higher memory and computation since the number of voxels N_{vox} grows exponentially with smaller voxel sizes:

$$N_{\text{vox}} = \frac{1}{\text{voxel size}^3} \tag{4.19}$$

For example, a reduction of the voxel size from 0.05 to 0.04 increases the number of voxels by approximately 50

$$N_{\text{vox}}(0.05) = \frac{1}{0.05^3} = 8,000, \quad N_{\text{vox}}(0.04) = \frac{1}{0.04^3} = 15,625 \tag{4.20}$$

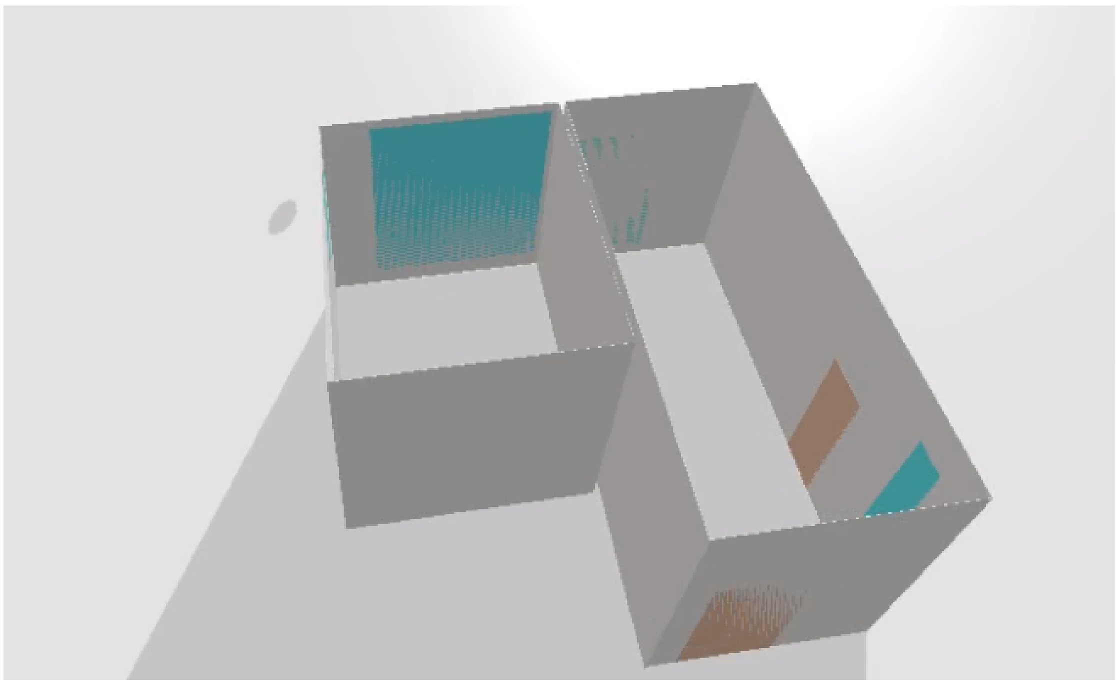
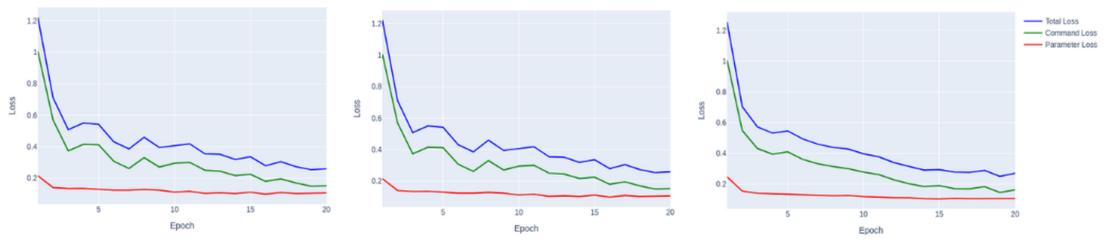


Figure 4.2: Meshes rendered in PyCram, enabling smooth integration into simulation environments [1].



a) 0.03 b) 0.04 c) 0.05

Figure 4.3: Loss convergence rates for three different voxel sizes, 0.03, 0.04, 0.05 over 20 epochs. All of them have a similar convergence rate.

For example, further reducing it to 0.03 would have an exponential growth of usage in memory and very long training time without yielding proportional benefits in scene reconstruction accuracy.

A voxel size of 0.04 offered a good tradeoff: the algorithm preserved enough geometric detail from the point cloud while maintaining reasonable memory and computational requirements. Using smaller voxel sizes allows for the capture of finer details; however, this comes at a higher computational cost that is not justified by the minimal improvements in model performance.

Thus, 0.04 was chosen to strike the best balance between capturing detail during training and computational efficiency for the voxel size.

4.3.2 Model Training

The model was first fit too closely to one scene to ensure it could learn and reconstitute the spatial relationships and commands specific to that particular scene. This overfitting experiment offered verification that the architecture worked the way it should: the model performed satisfactorily in the prediction of commands and their relative parameters across the scene it had been trained on. The ground truth sequence of commands was very similar, and the structural information of the scene was captured well by the 3D reconstruction from these predictions. Figure: 4.4 consists of a graph showing loss during overfitting along with the rendered ground truth and predicted scenes.

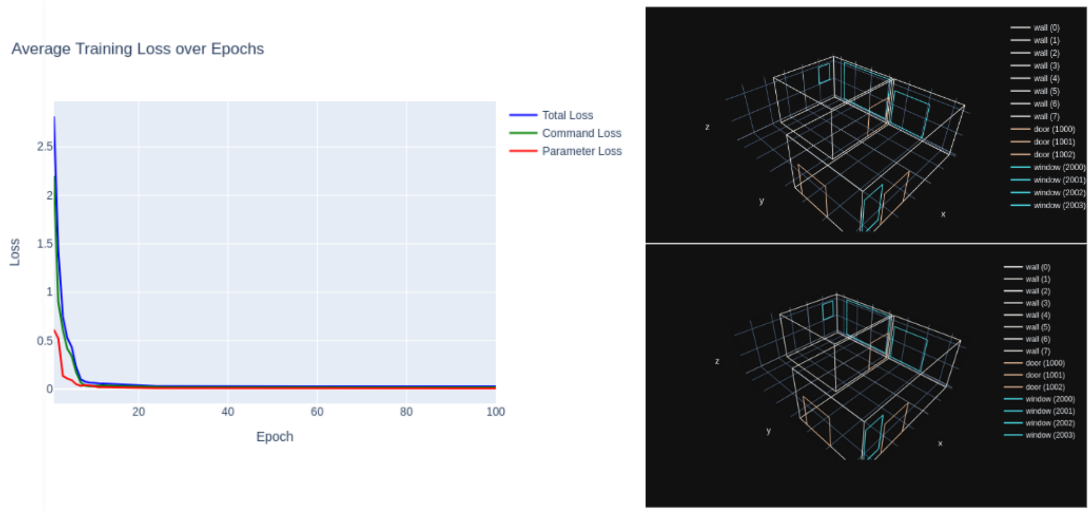


Figure 4.4: Plot showing the model's loss during overfitting on a single scene, along with the rendered ground truth (top) and prediction (bottom)

Validated through this overfitting test, the model was trained for a dataset of 4000 scenes both in point clouds and scene script format. Training on batch size 32 with the Adam optimizer, and a learning rate of 0.0001, a step learning rate scheduler that reduced the learning rate by half every 10 epochs. The loss function consisted of two parts: cross-entropy loss for predicting the categorical commands and mean squared error (MSE) for the continuous parameters. The training time for the model was 6 days 13 hours.

4.3.3 Testing

During testing, the model is evaluated on 50 unseen scenes, inputting only the point cloud without its corresponding scene script. A `START` token is given first to initiate an autoregressive generation process of a sequence of commands, one step at a time. In each step, the model predicts a command with its corresponding parameters; the process continues until a `STOP` token is predicted, which signals the end of the scene description.

Each predicted command will correspond to six parameters, which include spatial position, dimension, and orientation of the architectural elements. The object ID is the first among these six parameters and is predicted as a `float` then converted into an `int` for use in the scene reconstruction. These are the parameters that will describe the geometric properties of objects. Since the model was trained on normalized scripts, the output script that will be predicted would also be normalized thus keeping the parameters all consistent in scale.

Once generated, the scene is rendered by taking as input the predicted commands and normalized parameters with the object ID converted to an `int`. In this manner, it will be possible to evaluate whether the model can infer complete and accurate scene descriptions solely from point cloud input in the absence of supervision from scene scripts at test time. The quality of the model's prediction is considered by the similarity of the generated sequence to the structure of the real scene.

The testing duration varies significantly based on scene complexity, ranging from approximately 17 seconds for simple scenes to around 54 seconds for more complex indoor layouts.

4.4 Results

4.4.1 Quantitative Evaluation

To test the accuracy of the layout estimation, I used the entity distance metric (d_E) using Hungarian matching and then compared the performance of my model to SceneScript. The d_E is defined as the maximum Euclidean distance between the corners of corresponding entities (walls, doors, and windows) from the ground truth and predictions:

$$d_E(E, E') = \max\{||c_i - c'_{\pi(i)}|| : i = 1, \ldots, 4\} \tag{4.21}$$

Where $\pi(i)$ is the permutation found via Hungarian matching.

The metrics used to evaluate the model's performance are:

- **F1 Score @ Threshold**: The F1 score at a fixed entity distance threshold.

- **Average F1 Score**: Computed across multiple thresholds, averaged over the dataset.

- **PSNR (Peak Signal-to-Noise Ratio)**: Measures reconstruction quality by comparing the maximum signal to error ratio.

- **LPIPS (Learned Perceptual Image Patch Similarity)**: Assesses perceptual differences based on deep features.

The table below compares the results of the model against those reported for SceneScript [4]:

Method	**F1 @ 5cm**	**Wall (F1 Avg)**	**Door (F1 Avg)**	**Window (F1 Avg)**
SceneScript	0.930	0.843	0.811	0.692
My Model	0.815	0.815	0.698	0.610

Table 4.1: Comparison of F1 scores between SceneScript and my model using point cloud data.

According to Table: 4.2, the model's PSNR score of 27.65 dB suggests that it can accurately recreate scenes with an acceptable level of visual fidelity. A higher PSNR indicates less noise and a closer match to the ground truth, indicating that the model accurately captures and predicts spatial elements like windows, doors, and walls. This degree of accuracy suggests that the model works well in applications like interior navigation systems and architectural modeling, where upholding precise and unambiguous geometry is essential.

Also, the LPIPS score of 0.2895, while acceptable, highlights perceptual differences that can arise in areas with complex geometry or cluttered data, such as overlapping entities or densely packed spaces. This score shows that, although the model performs well overall, there are still subtle inconsistencies in fine structural details that need refinement.

Metric	**Value**
PSNR (Point Cloud)	27.6541 dB
LPIPS (Point Cloud)	0.2895

Table 4.2: Performance metrics of the model using point cloud data.

These results point to how well the model performs on robust 3D reconstruction even with sparse data input. That is to say, the model generalizes well to different indoor scenes, predicts accurate spatial layouts, and may also maintain important features despite inherent challenges like occlusion and clutter. Further tuning and refinement are necessary, especially for elaborate and densely packed areas, to achieve higher fidelity reconstructions.

These accuracy metrics demonstrate both the possibility of the model for practical applications and further work, especially on the accuracy of fine-grained structural details. It is also anticipated that these insights will form a guide for further improvement and enhancement of the model in the future.

4.4.2 Qualitative Results

The model-generated 3D scenes, which are constructed using point cloud inputs, range in complexity. Figure 4.5 depicts simple room layouts with a single door or window, proving the model's accuracy in basic architectural setups.

In Figure: 4.6, the model successfully reconstructs rooms with multiple doors and windows, showing its adaptability to more intricate setups.

More complex configurations are illustrated in Figure: 4.7, revealing the model's capability to reconstruct scenes with several spatial relationships and objects positioned closely together.

While the model accurately depicts the overall structure, tightly packed or small places pose difficulties. Overlaps between things such as doors and windows as seen in Figure: 4.8 indicate limitations in voxelization. Although this procedure is efficient, it may reduce spatial precision when numerous objects are closely aligned. The voxel size utilized may be insufficient to distinguish between these near objects, resulting in overlap mistakes. This constraint highlights the necessity for a more flexible voxel size that may change dynamically depending on object density in the scene.

Moreover, the dataset's diversity plays a crucial role. A lack of varied examples of densely packed configurations may limit the model's learning capability, preventing it from accurately distinguishing adjacent entities. For instance, the model may fail to generalize effectively in scenes where architectural elements are tightly positioned, resulting in misalignments.

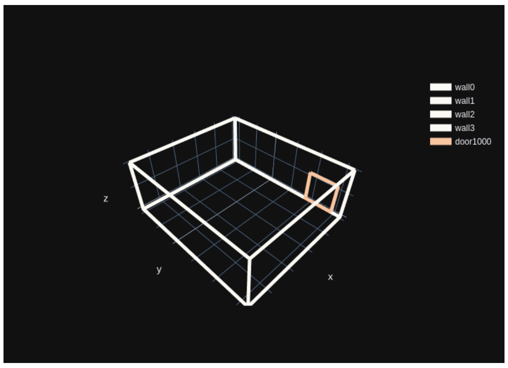
The attention mechanism, which detects spatial correlations, may also underperform in such instances. If the attention focus is too broad, it can obscure those small distinctions required to recreate tight spaces accurately. In certain cases, the attention may fail to detect precise boundaries, leading to overlapping predictions or inaccurate spatial relationships.

These findings highlight areas for future research, such as increasing the dataset to include more examples of complicated layouts and fine-tuning the attention mechanism to balance broad spatial context with localized precision. Addressing these difficulties could significantly improve the model's accuracy in reconstructing dense and complex indoor environments.

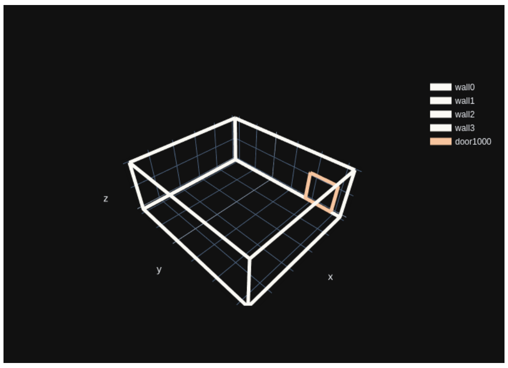
These findings highlight possibilities for development, such as expanding the dataset to include additional examples of complex layouts and fine-tuning the attention mechanism to balance broad spatial context with localized precision. Addressing these concerns could greatly improve the model's accuracy in reconstructing dense and complicated indoor environments.

Despite these minor flaws, the visual results demonstrate the model's ability to generate coherent and visually realistic images from minimal input data. The model's capacity

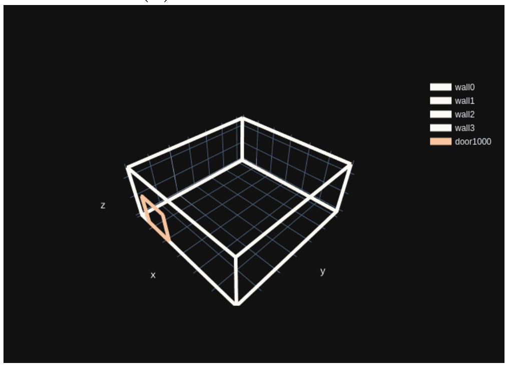
Ground Truth **Prediction**



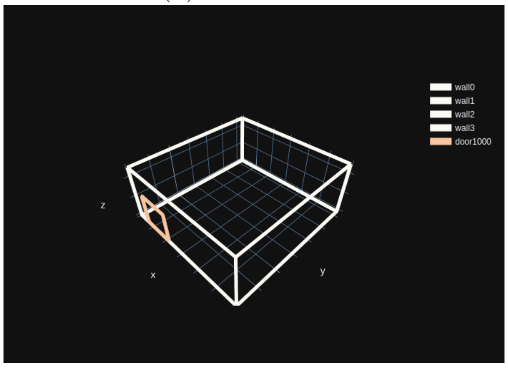
(a) Ground Truth 1



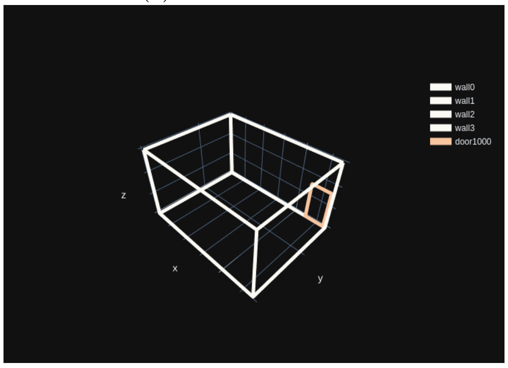
(b) Prediction 1



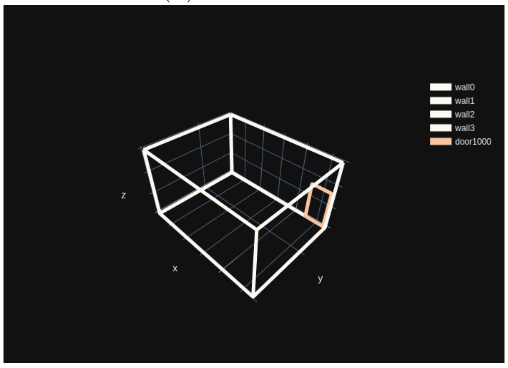
(c) Ground Truth 2



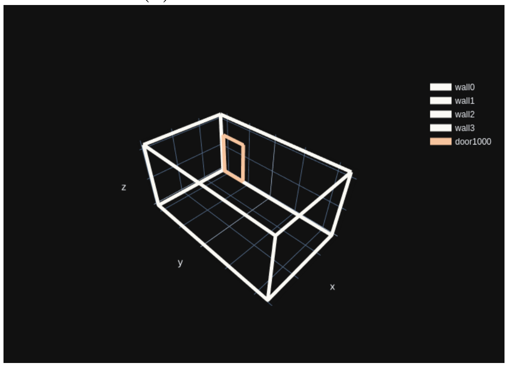
(d) Prediction 2



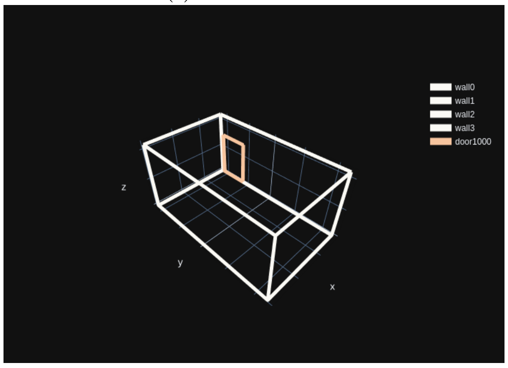
(e) Ground Truth 3



(f) Prediction 3



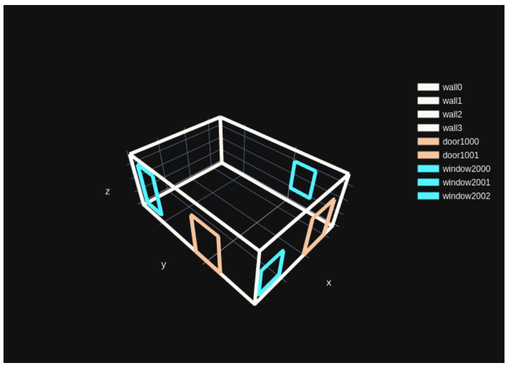
(g) Ground Truth 4



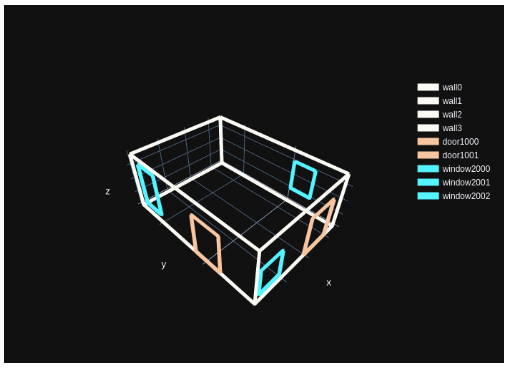
(h) Prediction 4

Figure 4.5: Predicted vs. ground truth layouts for simple room configurations.

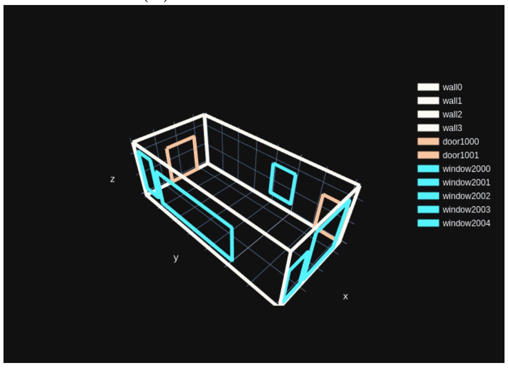
Ground Truth **Prediction**



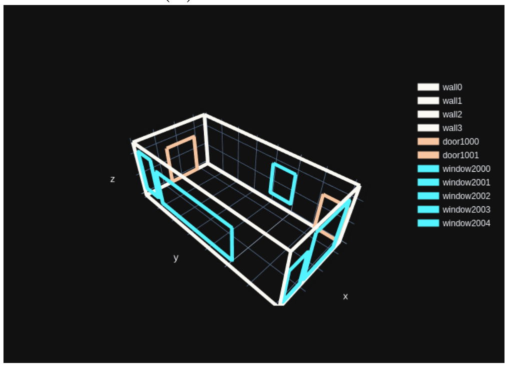
(a) Ground Truth 1



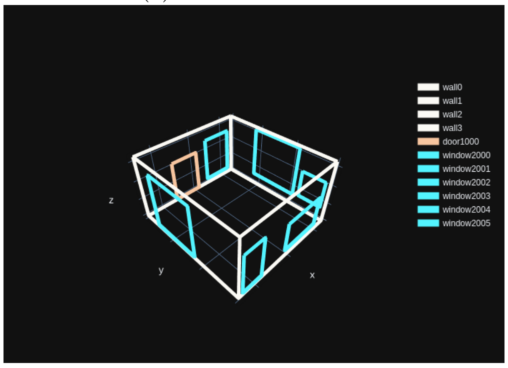
(b) Prediction 1



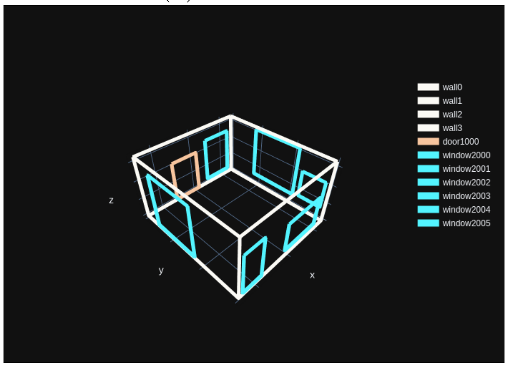
(c) Ground Truth 2



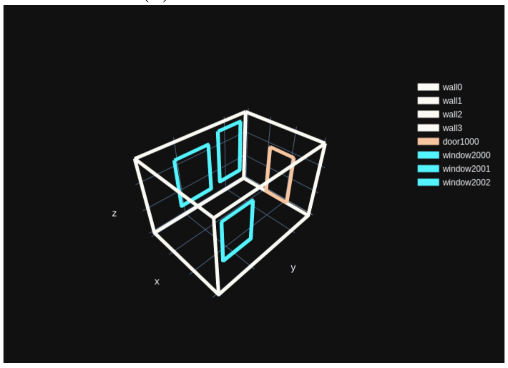
(d) Prediction 2



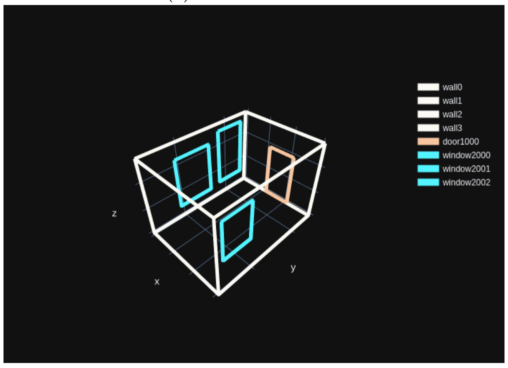
(e) Ground Truth 3



(f) Prediction 3



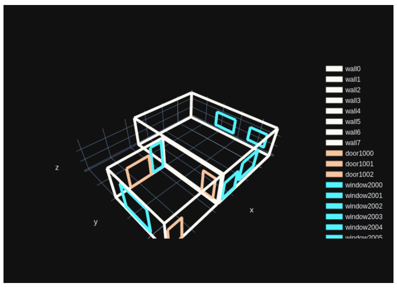
(g) Ground Truth 4



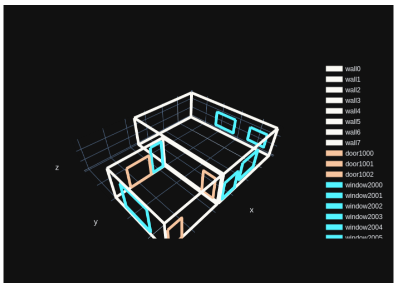
(h) Prediction 4

Figure 4.6: Predicted vs. ground truth for moderately complex room configurations.

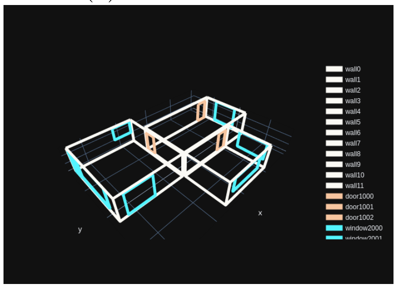
Ground Truth **Prediction**



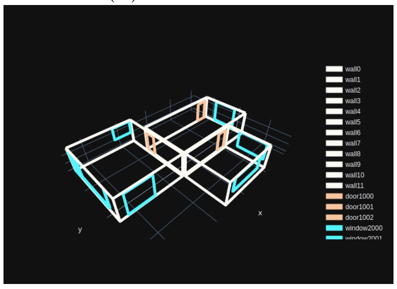
(a) Ground Truth 1



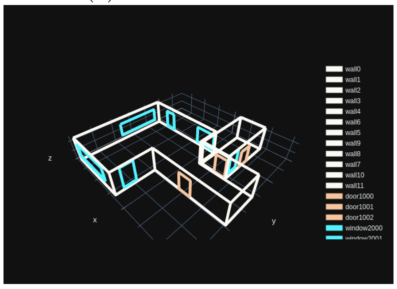
(b) Prediction 1



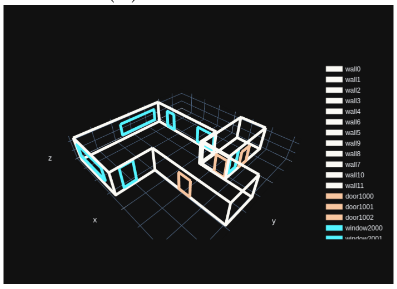
(c) Ground Truth 2



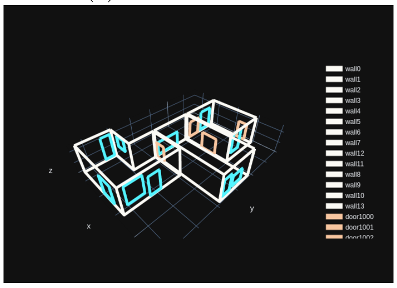
(d) Prediction 2



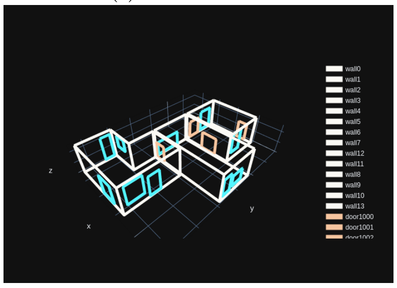
(e) Ground Truth 3



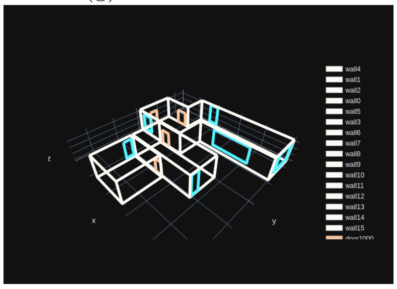
(f) Prediction 3



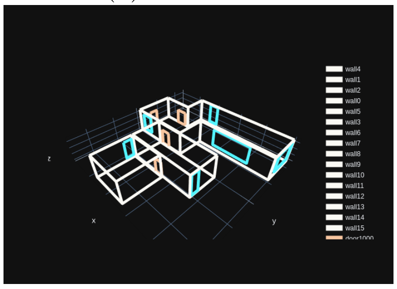
(g) Ground Truth 4



(h) Prediction 4



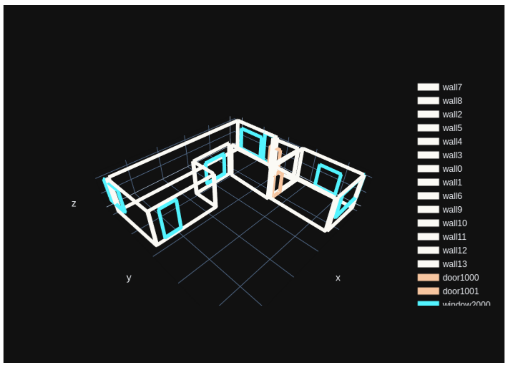
(i) Ground Truth 5



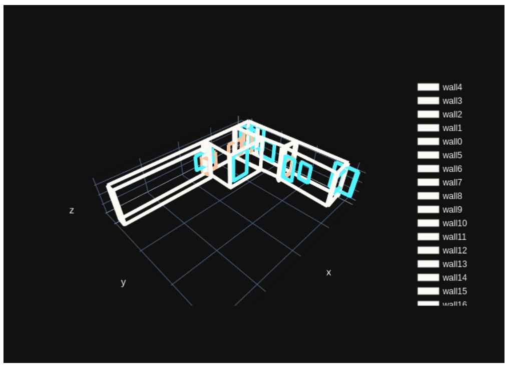
(j) Prediction 5

Figure 4.7: Qualitative results of complex room layouts showcasing effective scene reconstruction.

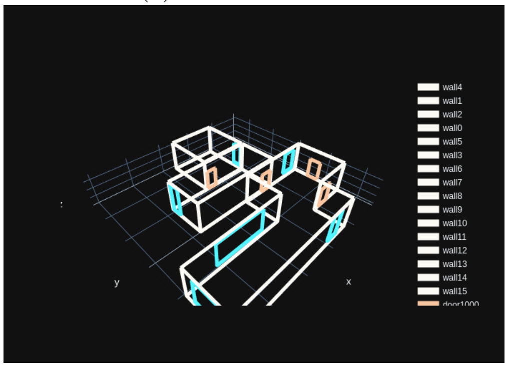
Ground Truth **Prediction**



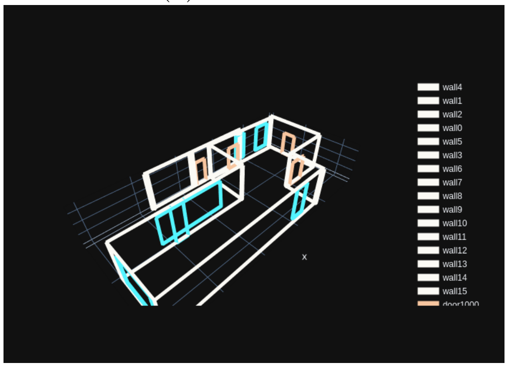
(a) Ground Truth 1



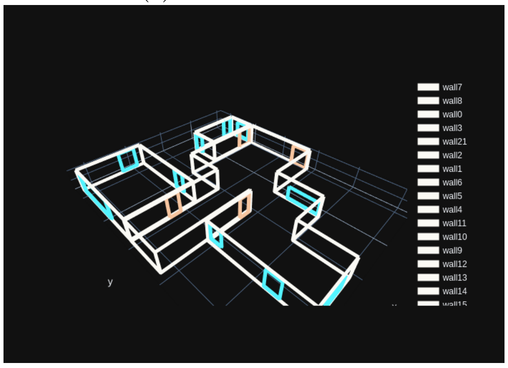
(b) Prediction 1



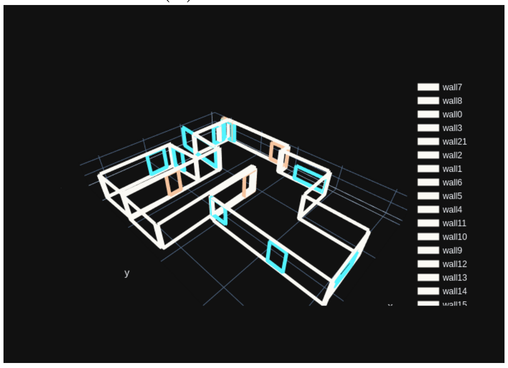
(c) Ground Truth 2



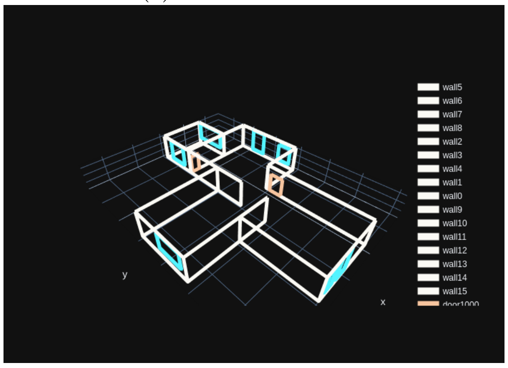
(d) Prediction 2



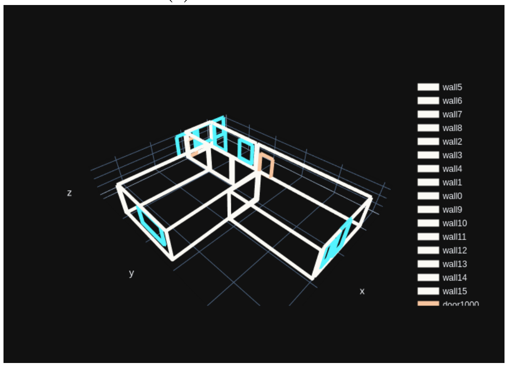
(e) Ground Truth 3



(f) Prediction 3



(g) Ground Truth 4



(h) Prediction 4

Figure 4.8: Examples of overlapping objects in complex scenes.

to infer spatial relationships and structure 3D spaces with great fidelity demonstrates its suitability for a wide range of indoor layouts. These qualitative findings support the model's strengths while also highlighting areas for improvement in complicated spatial configurations.

4.5 Limitations

Despite the model's success in 3D reconstruction from point clouds, several limitations need to be highlighted:

4.5.1 Trade-offs with Sparse Input Data

The reliance on point cloud data, while advantageous for direct 3D spatial representation, has problems. Sparse input data lacks detail in particular places, resulting in missing or incorrectly represented elements during reconstruction. For instance, when areas of the point cloud have fewer points due to occlusions or incomplete scanner trajectories, the model fails to properly infer information about missing parts. This sparsity creates gaps or inconsistencies in the results, especially over minor features such as door frames or narrow corridors.

Furthermore, sparse data requires a trade-off between detail and computing efficiency. Higher voxel resolution captures more detail, but it also consumes more memory and takes longer to process. Finding the proper voxel size was crucial, as higher resolutions became impracticable for real-time applications, while lower resolutions reduced detail fidelity.

4.5.2 Assumption of Orthogonal Structures

The model operates under the assumption that walls and ceilings are aligned orthogonally to the z-axis. While this works well for standard architectural designs, it may limit the model's generalization to non-standard features such as curved walls or sloped ceilings, which are common in modern architecture. Although this particular limitation has not been tested, reliance on voxelization processes that convert point clouds into grid structures would insinuate that the model might over-simplify such non-rectilinear forms and hence could be inaccurate for such scenarios.

4.5.3 Dependence on Normalization

Normalization of input values is essential for model stability and accuracy. Early experiments with unnormalized scripts revealed that parameter loss spikes significantly when values aren't scaled appropriately. As shown in Figure: 4.9, the model's architecture, which handles command prediction effectively through discrete classifications, struggles with continuous parameter values unless they are normalized to a consistent scale. This indicates a limitation in handling variability and suggests that, without normalized inputs, the model's robustness is compromised.

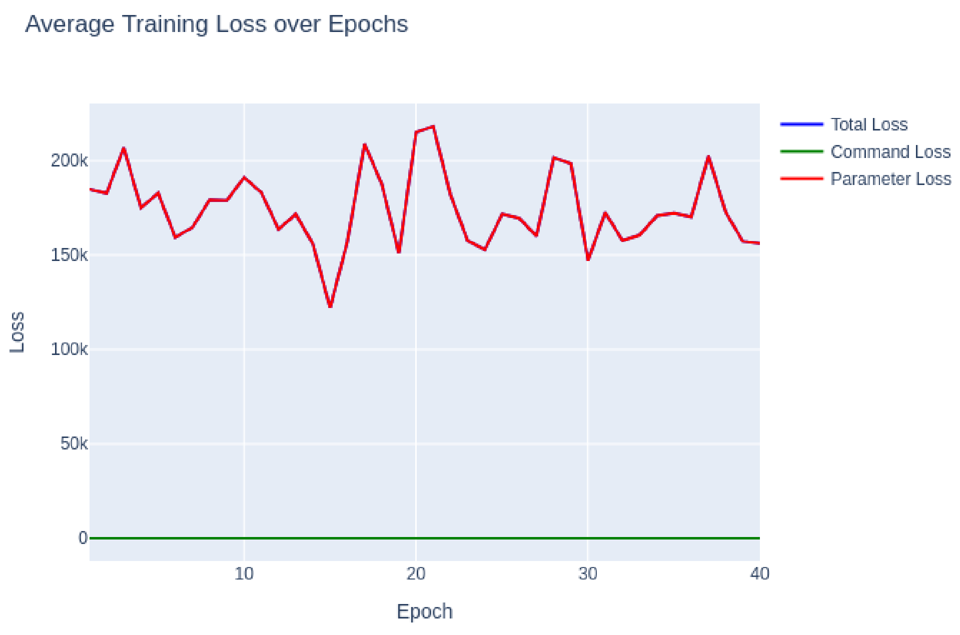


Figure 4.9: Plot showing the spike in loss when training with unnormalized scripts.

4.5.4 Generalization to Complex Geometries

The dataset used to train the model lacks diverse examples of non-orthogonal and complex geometries, such as arches or curved surfaces. Consequently, the model's ability to generalize beyond standard, rectilinear architectures is limited. For scenes with intricate layouts or varying angles, the attention mechanism may fail to capture the necessary spatial relationships accurately, leading to oversimplifications in the output. Enhancing the dataset with diverse architectural features and expanding the training scope could potentially mitigate this limitation.

4.5.5 Overlapping and Densely Packed Features

The model shows difficulty in accurately reconstructing scenes with closely positioned or overlapping features. As demonstrated in earlier results, when multiple elements like doors and windows are packed in a small area, voxelization resolution may not be sufficient to distinguish between these objects precisely, resulting in overlaps or misalignments. This issue is also related to the shortage of diverse training data for densely packed environments, whereby models do not see enough examples to learn from and generalization is poorer in such cases.

The limitation of the model, in turn, points out that it is hard to find a good balance between voxel resolution and sparsity of data and generalize on complicated architectural aspects. Addressing these concerns would include improving both the dataset and the model's design to boost resilience and flexibility.

Chapter 5

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions are shown.

5.1 Conclusion

This thesis proposes a methodology for 3D scene reconstruction utilizing point cloud data, proving the efficacy of neural network-based, organized command generation in producing correct architectural layouts. The model efficiently converts point clouds into a series of commands, recreating critical features like walls, doors, and windows with excellent structural accuracy.

The key findings reveal that the model achieves great perceptual and structural accuracy over a wide range of indoor settings, as evidenced by quantitative measurements. The effective reconstruction of large-scale architectural components demonstrates the model's ability to generalize across contexts while minimizing error rates. The suggested approach is also scalable, as it automates the reconstruction process for complicated situations without requiring considerable manual intervention.

The model effectively solves the research challenges by utilizing sparse 3D data to recreate major structural features such as walls, doors, and windows across a variety of contexts. It meets the goals of scalability and automation by managing a wide range of architectural challenges with minimal involvement. This feature proves the model's applicability in a wide range of complicated circumstances.

Despite these achievements, constraints persist. The model struggles with fine-grained geometric details, especially in densely packed places, and its current reliance on voxelized input restricts its capacity to handle non-standard, curved geometries. These limitations point to future work in two directions: improving the capabilities of the model in capturing fine details and being adaptable to various input formats for practical applicability.

This thesis presented a robust, scalable system for fully automated 3D indoor reconstruction; its promising applications include architectural modeling, robotics, and VR. Future enhancement is intended to increase the model's flexibility and accuracy to be helpful in a real-world application.

5.2 Future Work

Future work can take this model and its capabilities further in a number of directions:

One of the key directions is auto-structured command generation. It is noted that in the current ASE dataset, language commands are pre-defined by humans. In future work, it may apply self-supervised learning or reinforcement learning to enable the model to learn on its own the relations of objects concerning each other and generate structured commands. This will lower the dependency on predefined rules and make the model adaptable under diverse conditions.

The model can also be extended to dynamically identify and predict additional object classes. For instance, categories present in the ASE dataset, which includes a total of 20 classes, can be seamlessly integrated into the model's prediction framework. It will then be able to generate additional commands such as:

```
make_chair(position_x, position_y, position_z, width, height, length)
```

This could be extended by adding an extra layer of commands in the decoder that could predict furniture-like objects, adding these commands to both the predicted and ground truth scripts when training. That way, this extension allows the model to generate indoor scenes with more detail, complete with reconstructions beyond mere architectural elements of walls, windows, and doors.

Being able to handle more complex structures like a curved wall and sloped ceilings would greatly enhance the usability of the model. This is not explicitly tested, but adding new command types like:

```
make_curved_wall(a_x, a_y, a_z, b_x, b_y, b_z, c1_x, c1_y, c2_x, c2_y, height, thickness)
```

With parameters such as Bezier control points, this will allow the model to rebuild the surface correctly, be it a curve or non-planar surface. Future datasets with such structures would be needed in training the model on how to identify and process such features effectively in increasing the flexibility of the model and expanding the range of applications.

Another promising direction is to develop a model that directly predicts un-normalized values, which essentially enables the model to operate on an absolute scale without requiring normalized input data. This would enable the model to directly produce real-world dimensions and positions and, therefore, be applicable on a wide range of scales without any further post-processing. The normalization included as part of learning by the model would, therefore, bridge gaps between scaled and unscaled data and enable the model to intrinsically understand and predict real-world measurements.

Another important future direction is the application of the model to real-time 3D reconstruction in robotics. This includes the optimization of the model for real-time inference and the integration with robotic systems that may enable an on-the-fly 3D scene understanding and navigation, which is a core factor in autonomous robots operating within dynamic environments. This would involve incorporating real-time data streaming and improving computational efficiency by allowing the model to operate within hardware constraints on robotic platforms.

Moreover, this can also be used for interactive scene reconstruction with AR, enhancing its usability. By allowing real-time interaction with users, it can adapt the model to modify and refine the reconstructions of users while they are being generated. Such integration of AR technologies allows various applications in the fields of interior design, architectural visualization, and robotics, where there is a need for user input and immediate feedback.

These potential future works will further extend the model's accuracy and functionality for tremendous opportunities for both practical and theoretical advancement of the technologies involved in 3D reconstruction.

Bibliography

[1] Pycram. Available at: Link.

[2] M. Zollhöfer S. Izadi A. Dai, M. Nießner and C. Theobalt. Bundlefusion - real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. ACM Transactions on Graphics (2017). DOI: 10.1145/3054739.

[3] A. Adan and D. Huber. 3d reconstruction of interior wall surfaces under occlusion and clutter, 2011. DOI: 10.1109/3DIMPVT.2011.42.

[4] A. Avetisyan and C. et al. Xie. Scenescript: Reconstructing scenes with an autoregressive structured language model, 2024. DOI: 10.48550/arXiv.2403.13064.

[5] S. Becker. Generation and application of rules for quality dependent façade reconstruction, 2009. DOI: 10.1016/j.isprsjprs.2009.06.002.

[6] S. Becker, M. Peter, and D. Fritsch. Grammar-supported 3d indoor reconstruction from point clouds for "as-built" bim, 2015. Vol. 1, pp. 17–24, DOI: 10.5194/isprsannals-II-3-W4-17-2015.

[7] S. Becker, M. Peter, D. Fritsch, D. Philipp, P. Baier, and C. Dibak. Combined grammar for the modeling of building interiors, 2013. DOI: 10.5194/isprsannals-II-4-W1-1-2013.

[8] A. Budroni and J. Boehm. Automated 3d reconstruction of interiors from point clouds, 2010. pp. 55–73, DOI: 10.1260/1478-0771.8.1.55.

[9] O. Cameron. An introduction to lidar: The key self-driving car sensor., 2017. Link.

[10] J. Chen and B. Chen. Architectural modeling from sparsely scanned range data., 2008. DOI: 10.1007/s11263-007-0105-5, pp. 223–236.

[11] L. Diaz-Vilarino, K. Khoshelham, J. Martínez-Sánchez, and P. Arias. Modeling of building indoor spaces and closed doors from imagery and point clouds, 2015. 3491–3512, DOI: 10.3390/s150203491.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, and Gelly et. al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. DOI: 10.48550/arXiv.2010.11929.

[13] C. Frueh, S. Jain, and A. Zakhor. Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images, 2005. DOI: 10.1023/B:VISI.0000043756.03810.dd.

[14] M. Goetz and A. Zipf. Indoor route planning with volunteered geographic information on a (mobile) web-based platform, in: Krisp, j.m. (ed.), progress in location-based services, lecture notes in geoinformation and cartography. springer berlin heidelberg, pp. 211–231. DOI: 10.1007/978-3-642-34203-5$_1$2.

[15] Claus Nagel Karl-Heinz Häfele Gröger, Thomas H. Kolbe. Geography markup language (citygml) encoding standard. Link.

[16] G. Gröger and L. Plümer. How to achieve consistency for 3d city models, 2009. 15, 137–165, DOI: 10.1007/s10707-009-0006-7.

[17] ISO. Iso 16739-1:2018. Link.

[18] F. M. Moreno-D. Cruzado F. García J. Beltrán, C. Guindel and A. de la Escalera. A 3d object detection framework from lidar information. ITSC (2018). pp. 3517–3523, DOI: 10.1109/itsc.2018.8569311.

[19] P. Jenke, B. Huhle, and W. Straßer. Statistical reconstruction of indoor scenes, 2009. Available at: Link.

[20] K. Khoshelham and L. Diaz-Vilarino. 3d modeling of interior spaces: Learning the language of indoor architecture, 2014. DOI: 10.5194/isprsarchives-XL-5-321-2014.

[21] Li K.-J. Zlatanova S.-Kolbe T.H. Nagel C. Becker T. Lee, J. Indoorgml [www document]. Link.

[22] M. Levoy and K. Pulli. The digital michelangelo project: 3d scanning of large statues (presentation slides)., 2000. Link.

[23] C. Mura, O. Mattausch, A. Jaspe Villanueva, E. Gobbetti, and R. Pajarola. Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts, 2014. Vol. 44, pp. 20–32, DOI: 10.1016/j.cag.2014.07.005.

[24] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings, 2006. DOI: 10.1145/1179352.1141931.

[25] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades, 2007. DOI: 10.1145/1276377.1276484.

[26] S. Ochmann, R. Vock, R. Wessel, and R. Klein. Automatic reconstruction of parametric building models from indoor point clouds, 2016. pp. 94–103. DOI: 10.1016/j.cag.2015.07.008.

[27] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut, 2014. pp. 68–82, DOI: 10.1016/j.isprsjprs.2014.02.004.

[28] B. Okorn, X. Xiong, B. Akinci, and D. Huber. Toward automated modeling of floor plans, 2010. DOI: 10.3390/app10082817.

[29] B. Curless-S. Rusinkiewicz D. Koller L. Pereira M. Ginzton S. E. Anderson J. Davis J. Ginsberg J. Shade P. M. Levoy, K. Pulli and D. Fulk. The digital michelangelo project - 3d scanning of large statues. pp. 131-144. DOI: 10.1145/344779.344849.

[30] M. Previtali, L. Barazzetti, R. Brumana, and M. Scaioni. Towards automatic indoor reconstruction of cluttered building rooms from point clouds, 2014. 281–288, DOI: 10.5194/isprsannals-II-5-281-2014.

[31] O. Hilliges-D. Molyneaux D. Kim A. J. Davison P. Kohli J. Shotton S. Hodges R. A. Newcombe, S. Izadi and A. W. Fitzgibbon. Kinectfusion - real-time dense surface mapping and tracking. ISMAR (2011). pp. 127–136, DOI: 10.1109/ISMAR.2011.6092378.

[32] R.B. Rusu, Z.C. Marton, N. Blodow, A. Holzbach, and M. Beetz. Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments, 2009. DOI: 10.1109/IROS.2009.5354759.

[33] V. Sanchez and A. Zakhor. Planar 3d modeling of building interiors from point cloud data, 2012. pp. 1777–1780, DOI: 10.1109/ICIP.2012.6467225.

[34] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection, 2007. pp. 214–226, DOI: 10.1111/j.1467-8659.2007.01016.x.

[35] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture, 1971. DOI: https://dblp.org/rec/conf/ifip/StinyG71.bib.

[36] G. Stiny, W.J. Mitchell, et al. The palladian grammar, 1978. DOI: 10.1068/b050005.

[37] H. Tang, Z. Liu, X. Li, Y. Lin, and S. Han. Torchsparse: Efficient point cloud inference engine, 2022. Available at: https://torchsparse.mit.edu/.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. DOI: 10.48550/arXiv.1706.03762.

[39] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture, 2003. DOI: 10.1145/1201775.882324.

[40] J. Xiao and Y. Furukawa. Reconstructing the world's museums, 2014. DOI: 10.1007/s11263-014-0711-y.

[41] X. Xiong, A. Adan, B. Akinci, and D. Huber. Automatic creation of semantically rich 3d building models from laser scanner data, 2013. pp. 325–337, DOI: 10.1016/j.autcon.2012.10.006.

[42] Aditya Khosla Fisher Yu Linguang Zhang Xiaoou Tang Jianxiong Xiao Zhirong Wu, Shuran Song. 3d shapenets: A deep representation for volumetric shapes. DOI: 10.48550/arXiv.1406.5670.

Appendix A

List of Abbreviations

Abbreviation	**Full Term**
AR	Augmented Reality
ASE	Aria Synthetic Environments
BIM	Building Information Modeling
CGA	Computer Generated Animation
CityGML, LoD4	City Geography Markup Language, Level of Detail 4
CSG	Constructive Solid Geometry
d_E	Entity distance metric
GIS	Geographic Information System
GPU	Graphics Processing Unit
IFC	Industry Foundation Classes
IndoorGML	Indoor Geographic Markup Language
IndoorOSM	Indoor OpenStreetMap
JSON	JavaScript Object Notation
LPIPS	Learned Perceptual Image Patch Similarity
L-systems	Lindenmayer Systems
OGC	Open Geospatial Consortium
PSNR	Peak Signal-to-Noise Ratio
RANSAC	Random Sample Consensus
ReLU	Rectified Linear Unit
SDFs	Signed Distance Fields
SVM	Support Vector Machine
ViTs	Vision Transformers
VGI	Volunteered Geographic Information
VR	Virtual Reality

XML	eXtensible Markup Language

Appendix B

Source Code

The code for this project can be found at the following GitHub repository: `https://github.com/MariaSaleem6571/3d_SceneScript.git`.

The TorchSparse Library can be accessed here: `https://github.com/mit-han-lab/torchsparse.git`