

MASTERARBEIT

MULTIVIEW 3D SHAPE RECONSTRUCTION USING DEEP LEARNING

Freigabe:

Der Bearbeiter:

Unterschriften

Moiz Sajid



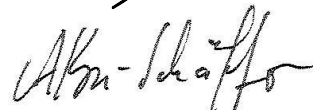
Betreuer:

Maximilian Denninger



Der Institutsdirektor

Prof. Alin Albu-Schäffer



Dieser Bericht enthält 65 Seiten, 25 Abbildungen und 10 Tabellen



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Multiview 3D Shape Reconstruction using Deep Learning

Moiz Sajid





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Multiview 3D Shape Reconstruction using Deep Learning

Multiview 3D Rekonstruktion von Objekten mittels Deep Learning

Author:	Moiz Sajid
Supervisor:	PD Dr. habil. Rudolph Triebel
Advisor:	Maximilian Denninger
Submission Date:	12.11.2021



I confirm that this Master's Thesis in Informatics is my own work and I have documented all sources and material used.

Munich, 12.11.2021

Moiz Sajid

Acknowledgments

First of all, I would like to thank my parents, who supported me in various ways throughout my studies. Their continuous support kept me motivated during my thesis, and I cannot thank them enough. I would also like to thank my grandparents for their prayers and support. The person who deserves the most credit for this thesis is my advisor Maximilian Denninger. Max was always available to answer all my questions, no matter how silly they were. He explained all the complex concepts in such a way that was easy for me to understand. Without his help and encouragement, this thesis would not have been possible. I would also like to thank the whole BlenderProc team for helping me out with different things that ultimately contributed to this thesis. Finally, I would like to thank PD Dr. habil. Triebel for allowing me to do my thesis in the Department of Perception and Cognition at the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), and making all the necessary resources available to me for doing this thesis. I had a great time while working at DLR, and I will definitely miss it.

Abstract

Deep learning has revolutionized computer vision through recent developments on tasks in this field. Although these developments initially started with 2D images, progress has been made recently in 3D computer vision. Tasks such as inferring the 3D shape from multiple images have also gained immense popularity recently due to the breakthroughs in the field of 3D deep learning. These advancements are made possible firstly, by the availability of large 3D object datasets, for example, ShapeNet [4], Pix3D [63], and ModelNet [72], secondly, by network architectures that can better handle 3D data, for example, DeepSDF [50], ShapeHD [70], and PSG [15], and thirdly, by the accessibility of efficient computing resources for processing 3D data.

Humans can actively infer the 3D world around them with just a single view of a scene. However, unlike humans, for computers the same task of estimating 3D information with just a single view becomes challenging because the single view reconstruction problem is generally ill-posed and ambiguous. Instead of perceiving the object of interest from one viewpoint, computers are provided with images from multiple viewpoints so that they can better reconstruct the 3D geometry of the object present in the images.

The goal of this thesis is to present and evaluate a multiview 3D shape reconstruction method for reconstructing the 3D environments better. More specifically, a sparse number of input images are provided to the proposed method to get an object’s representation in 3D. The reconstructions from these methods is crucial in applications such as virtual/augmented reality, autonomous driving, and robotic manipulation and grasping.

To this end, this thesis firstly proposes a large scale multiview dataset with 1,050,816 rendered images and 43,784 3D Truncated Signed Distance Function (TSDF) volumes based upon the ShapeNet [4] dataset, including accurate camera pose and intrinsic parameters. Secondly, a novel 2D-3D end-to-end trainable deep learning-based method for 3D shape reconstruction is presented using images taken from multiple viewpoints and camera parameters. The method maps the 2D features directly into 3D using a backprojection layer. Finally, detailed evaluation studies are conducted using the proposed multiview 3D shape reconstruction approach on the newly introduced dataset.

Abstract - German

Deep Learning hat den Bereich Computer Vision durch seine jüngsten Entwicklung revolutioniert. Obwohl diese Entwicklungen zunächst im 2D, meist auf Bildern begannen, ist es nun möglich diese Fortschritte auch im 3D anzuwenden. Aufgaben wie die Rekonstruktion der 3D-Form eines Objekts aus mehreren Bildern haben in letzter Zeit aufgrund der Durchbrüche auf dem Gebiet des 3D-Deep-Learnings ebenfalls immense Popularität erlangt. Diese Fortschritte werden erstens durch die Verfügbarkeit großer 3D-Objektdatensätze, wie z. B. ShapeNet [4], Pix3D [63] und ModelNet [72], zweitens durch Netzwerkarchitekturen, die 3D-Daten besser verarbeiten können, wie z. B. DeepSDF [50], ShapeHD [70] und PSG [15], und drittens durch den Zugang zu effizienten Rechenressourcen für die Verarbeitung von 3D-Daten ermöglicht.

Der Mensch ist in der Lage die Welt um sich herum mit nur einem einzigen Blick auf eine Szene aktiv zu erfassen. Im Gegensatz zum Menschen ist die gleiche Aufgabe der Schätzung von 3D-Informationen mit einer einzigen Ansicht für einen Computer jedoch eine Herausforderung. Da das Problem der Rekonstruktion aus einer einzigen Ansicht im Allgemeinen nicht eindeutig ist. Anstatt das zu untersuchende Objekt nur aus einem Blickwinkel zu betrachten, werden Computern Bilder aus mehreren Blickwinkeln zur Verfügung gestellt, damit sie die 3D-Geometrie des Objekts besser rekonstruieren können.

Das Ziel dieser Arbeit ist es, eine Multiview-3D-Form-Rekonstruktionsmethode vorzustellen und auszuwerten, um die Rekonstruktion von 3D-Umgebungen zu verbessern. Genauer gesagt, werden der vorgeschlagenen Methode eine geringe Anzahl von Eingabebildern zur Verfügung gestellt, um die Rekonstruktion eines Objekts in 3D zu erhalten. Diese Rekonstruktionen solcher Methoden sind entscheidend für Anwendungen wie virtuelle Realität, autonomes Fahren und robotische Manipulation und Greifen.

Zu diesem Zweck wird in dieser Arbeit zunächst ein groß angelegter Multiview-Datensatz mit 1.050.816 gerenderten Bildern und 43.784 TSDF-Volumen (Truncated Signed Distance Function) auf der Grundlage des ShapeNet-Datensatzes [4] erstellt, einschließlich genauer Kameraposition und intrinsischer Parameter. Zweitens wird ein neuartiges, durchgängig trainierbares 2D-3D-Verfahren auf der Basis von Deep Learning für die 3D-Formrekonstruktion vorgestellt, das Bilder aus verschiedenen Blickwinkeln und Kameraparameter verwendet. Die Methode bildet die 2D-Merkmale mithilfe einer Rückprojektionsschicht direkt in 3D ab. Schließlich werden detaillierte Evaluierungsstudien mit dem vorgeschlagenen Multiview-3D-Form-Rekonstruktionsansatz auf dem neu eingeführten Datensatz durchgeführt.

List of Acronyms

CNN Convolutional Neural Network

DAE Denoising Autoencoder

DBN Deep Belief Network

IoU Intersection over Union

MAE Mean Absolute Error

MSE Mean Squared Error

ResNet Residual Network

RNN Recurrent Neural Network

SfM Structure from Motion

SAE Sparse Autoencoder

TSDF Truncated Signed Distance Function

VAE Variational Autoencoder

vSLAM Visual Simultaneous Localization and Mapping

Contents

Acknowledgments	iii
Abstract	iv
Abstract - German	v
List of Acronyms	vi
1. Introduction	1
1.1. Contributions	1
1.2. Problem Statement and Notation	2
1.3. Thesis Structure	2
2. Related Work	3
2.1. Single-view 3D Reconstruction	3
2.1.1. Shape Reconstruction	3
2.1.2. Scene Reconstruction	3
2.2. Multiview 3D Reconstruction	3
2.2.1. Recurrent Neural Network (RNN) based methods	4
2.2.2. Encoder-Decoder based methods	4
2.2.3. Attention based methods	5
2.3. 3D Shape Completion	5
3. Methodology	7
3.1. Image Formation	7
3.1.1. Pinhole Camera Model	7
3.1.2. 3D Projections	10
3.1.3. View Frustum	11
3.2. 3D Data Representations	12
3.2.1. Point Cloud	12
3.2.2. Binary Occupancy Grid/Voxel Grid	12
3.2.3. Truncated Signed Distance Function (TSDF)	13
3.2.4. Mesh	14
3.3. Truncated Signed Distance Function (TSDF) Generation	14
3.3.1. Problem Statement	14
3.3.2. Methods	15
3.4. Deep Learning	16
3.4.1. Residual Network (ResNet)	16
3.4.2. Autoencoder	18

4. Our Approach	20
4.1. Problem Statement and Notation	20
4.2. Input and Output	20
4.3. Architecture	20
4.3.1. 2D Network	21
4.3.2. Backprojection Layer	22
4.3.3. 3D Network	23
4.3.4. Autoencoder	23
5. Experimental Setup	25
5.1. Dataset	25
5.2. Synthetic Data Generation	25
5.2.1. RGB Images, Camera Intrinsics, and Camera Extrinsics	26
5.2.2. Truncated Signed Distance Function (TSDF) Volumes	27
5.3. Neural Network Training	29
5.3.1. Loss	29
5.3.2. Evaluation	30
5.4. Training Procedure	31
5.4.1. Train, Validation and Test Splits	31
5.4.2. Processing	32
5.4.3. Implementation Details	32
6. Results	34
6.1. Quantitative	34
6.2. Qualitative	35
6.3. Comparison to other approaches	35
6.4. Space Complexity	38
6.5. Compressed Output Visualization	38
6.6. Changing Input Views	39
7. Future Work	41
7.1. Problem Benchmark and Dataset	41
7.2. Uncertainty Estimation	41
7.3. Real World Transfer	41
7.4. 3D Scene Datasets	42
7.5. Camera Intrinsics and Extrinsics	42
7.6. Adversarial Training	42
8. Conclusion	44
A. BlenderProc Config	45
List of Figures	47
List of Tables	50
Bibliography	51

1. Introduction

Nowadays, access to 3D data is possible thanks to not only 3D content creation but also better 3D capture devices, such as stereo cameras, laser scanners, and LiDAR. However, manual 3D content creation by artists is an expensive and time-consuming process since the 3D environments have to set up from scratch. Also, the 3D capture devices are still beyond the reach of most people because of the expensive cost. With each passing year, the demand for 3D data is likely to increase further because of the growing interest in the robotics, autonomous driving, and virtual/augmented reality communities. In order to meet this growing demand, new automatic 3D data generation methods are needed for truly democratizing access to 3D data. The availability of large-scale 3D datasets, like ShapeNet [4] and Pix3D [63], has further supported this mission.

The task of multiview 3D shape reconstruction is crucial in computer vision and robotics for obtaining an accurate 3D representation of an object using just the 2D data. Multiview 3D shape reconstruction methods infer the underlying 3D geometry of an object using RGB images taken from multiple viewpoints. Application areas include virtual/augmented reality, autonomous driving, and robotic manipulation and grasping.

Traditional approaches, such as Structure from Motion (SfM) [49] and Visual Simultaneous Localization and Mapping (vSLAM) [18] use feature matching across images captured from different views plus triangulation to recover the 3D coordinates of the image pixels. These methods can produce semi-dense and dense reconstructions; however, these approaches only work if a specific set of assumptions are satisfied, for example, a wide baseline and textured data.

The research area of multiview 3D shape reconstruction using deep learning has been studied extensively in the literature [6, 60, 73, 74]. However, most of the previous methods generate a binary voxel grid output of a small resolution which is non-smooth. This thesis, inspired by the previous works in the literature, proposes an end-to-end deep Convolutional Neural Network (CNN) for learning the mapping from the 2D to the 3D domain using a large-scale dataset without any assumptions. The network takes multiview RGB images of the 3D object from different viewpoints and camera parameters, namely camera intrinsics and extrinsics as input. The network outputs an intermediate 3D TSDF representation of resolution 512^3 which is converted into a mesh representation using meshification methods, like Marching Cubes [42]. The network both, during training and testing, does not require image annotations or object class labels. The network also does not make any prior assumptions about the problem, like a large baseline or a lambertian surface.

1.1. Contributions

The key contributions of our work are following:

New Multiview Dataset: A new dataset based upon the ShapeNet dataset [4] with the same categories and data splits as in 3D-R2N2 [6] is proposed that provides RGB images, camera poses and their respective TSDF volumes. Essential information, like textures in the images,

is included from the ShapeNet dataset to make the new dataset as realistic as possible. The RGB renderings and camera poses are generated using BlenderProc [12].

Improved Network Architecture: A novel deep learning-based architecture is proposed that directly associates the 2D features from n RGB images with 3D using camera intrinsics and extrinsics.

Higher Output Resolution: The method proposed in this thesis can generate a 3D volume with a resolution of 512^3 , making it one of the few methods capable of such a high-resolution output.

1.2. Problem Statement and Notation

Given n number of RGB images $I_c : \Omega_c \rightarrow [0, 255]^3$ of dimension $u \times u \times 3, u \in \mathbb{N}$ where $\Omega_c \subset \mathbb{R}^2$ and n number of camera poses as input, the task of multiview 3D shape reconstruction is to generate a mapping from 2D image coordinates $\mathbf{x}_c = (x_c, y_c)$ to 3D object coordinates $\mathbf{x}_s = (x_s, y_s, z_s)$. The output is a high-resolution 3D TSDF $V : \Omega_v \rightarrow [-\sigma_{tsdf}, \dots, \sigma_{tsdf}]$ of dimension $w \times w \times w$ where $\Omega_v = \{0, \dots, 511\}^3$.

1.3. Thesis Structure

[Chapter 2](#) outlines existing work done in literature for the task of 3D shape reconstruction and other related tasks, like 3D shape completion. The strengths and weaknesses of the different 3D shape reconstruction methods are also highlighted here. Plus, this chapter points out how the proposed method handles the shortcomings of the other approaches. [Chapter 3](#) introduces the methodological background of concepts used, namely camera projection, different 3D representations, TSDF generation, and deep learning. [Chapter 4](#) presents the proposed neural network architecture for solving the task of multiview 3D shape reconstruction. The different components of the neural network as well as the different architectural choices are explained in this chapter. [Chapter 5](#) discusses the experimental setup with regards to the dataset creation as well as the training aspects of the proposed deep learning method. In [Chapter 6](#) results from different experiments conducted during the thesis are presented. The insights of the results are also discussed in this chapter. [Chapter 7](#) mentions some further steps that can be investigated as possible future work. Finally, [Chapter 8](#) provides the conclusion of this thesis work.

2. Related Work

In this chapter, an overview of the related 3D reconstruction or similar methods is provided.

2.1. Single-view 3D Reconstruction

Generating 3D reconstruction from just a single image is a challenging task because the single-view 3D reconstruction problem is an ill-posed and ambiguous problem since the partially predicted points can be associated to an infinite number of 3D models as mentioned in Xie et al. [74].

2.1.1. Shape Reconstruction

Methods have been introduced recently with new data representations for the task of 3D shape reconstruction. These data representations include point clouds [15], meshes [67] and signed distance fields [75]. The PSG method [15] recovers a point cloud from a single RGB image. The method of Pixel2Mesh [67] is the first method in literature for generating a triangular mesh from a single RGB image. The approach of DeepSDF [50] provides the SDF representation of a set of points provided as an input. However, this approach will not work for reconstruction from just an RGB image. The proposed method in this thesis also generates encoded TSDF volume in the end. However, the method is not a generative model, unlike DeepSDF, with no probabilistic interpretation. The OGN [64] method uses an octree for handling the memory constraints of large 3D resolutions. Matryoshka Networks [55] decompose the 3D shape into nested shape layers. The method can outperform octree-based reconstruction methods, and it can generate output resolution as high as 256^3 .

2.1.2. Scene Reconstruction

The work of Denninger et al. [14] proposes not only an efficient method for generating TSDF volumes but also a tree net architecture that solves the scene reconstruction task by splitting channel-wise. This method uses an autoencoder for efficiently compressing TSDFs of a resolution of 512^3 to $32^3 \times 64$. The decoder part of the autoencoder is used to return to the original resolution of 512^3 , which means it is one of the only methods out there that can generate this high of a resolution. Furthermore, the method also proposes a custom loss shaping function, which penalizes the loss around the surface of an object and the free space before an object more. This thesis makes use of not only the autoencoder for compression and decompression but also a modified version of the TSDF generation pipeline as proposed in Denninger et al.

2.2. Multiview 3D Reconstruction

Traditional dense 3D reconstruction methods, for example, SfM and vSLAM require a dense number of RGB images with a certain set of assumptions. These traditional methods involve

feature extraction and matching [49] or minimizing reprojection errors [3, 18]. Firstly, the feature matching process can be slow especially if, for example, SIFT features are calculated, and secondly, the extracted features should cover the whole surface of the 3D object. Otherwise, there may be occlusions or holes in the final 3D reconstruction.

One of the first multiview deep learning based methods from literature is the MVCNN [62] network. In MVCNN, 3D geometry is rendered into 2D after which the 2D features are calculated, followed up by max pooling. This approach works suitably well for the task of classification; however, it is not suitable for other upstream 3D tasks, like reconstruction.

2.2.1. Recurrent Neural Network (RNN) based methods

The 3D-R2N2 [6] method proposed an RNN for multiview 3D shape reconstruction where the authors for each multiview image, use an RNN module. However, this approach suffers from several issues. Firstly, the approach is order variant meaning that the generated results depend on the order in which the images of the different viewpoints are given to the network. Secondly, the approach suffers from long-term memory-related issues common in RNNs, which means that the features learned from the initial images might be forgotten. Finally, the approach is not parallelizable and hence time-consuming since the images are processed sequentially. The LSM [33] method also uses an RNN for fusing 3D features from different views. However, it addresses the RNN related problems identified in the approach of 3D-R2N2. The LSM approach also uses feature projection and unprojection along the viewing rays for which it needs the camera intrinsic and extrinsic parameters. As reported in Xie et al. [74], LSM performs better with more than one view as compared to other methods. They argue that for more than one view, the camera intrinsics and extrinsics help to align the 2D features of multiview images better. Our proposed approach is firstly not dependent on the view order since the images are processed spatially instead of being processed temporally. Additionally, the proposed approach makes use of camera intrinsics and extrinsics, similar to LSM, which are generated, along with the 2D renderings, using BlenderProc [12].

2.2.2. Encoder-Decoder based methods

The Pix2Vox [73] method uses an encoder-decoder based architecture alongside a context aware module for fusion and a refiner module for correcting wrongly recovered reconstructions. Even the network produce impressive results, the training process is not end-to-end where the modules are tried separately. The authors tried to improve their work in a follow-up method named Pix2Vox++ [74] that generates better reconstructions due to improved architectural choices. They also propose a large scale multiview 3D shape reconstruction dataset named Things3D, based upon the SUNCG [59] dataset, which unfortunately is no longer available. The work of Spezialetti et al. [60] proposed to do multiview 3D shape reconstruction with the added task of estimating the relative pose image pairs used for reconstruction. Unlike the encoder-decoder based approaches, our approach uses a 2D network that calculates 2D features, which are directly associated with the 3D reconstruction using camera intrinsics and extrinsics parameters. Furthermore, a new dataset is proposed with the output target having a TSDF representation with the same categories and data splits as in 3D-R2N2 [6].

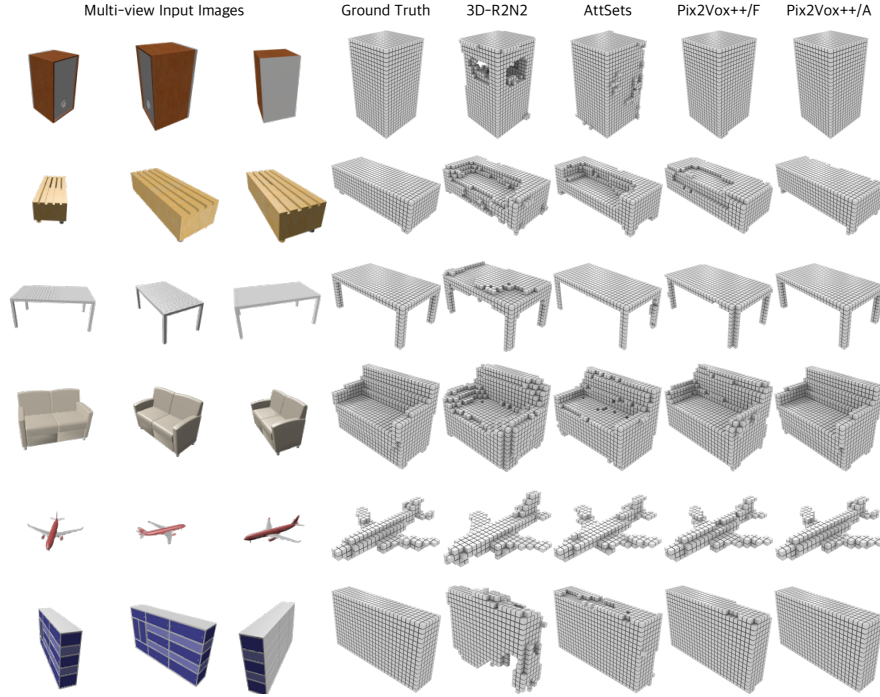


Figure 2.1.: Figure shows the binary occupancy grid output of resolution 32^3 of the different multiview 3D shape reconstruction methods on the dataset introduced in 3D-R2N2 from Choy et al. [6]. The image is taken from Xie et al. [74].

2.2.3. Attention based methods

The work of Yang et al. [77] proposed an attention aggregation module named AttSets and a training algorithm named FASet. The work claims to have an aggregation approach comparable to pooling-based approaches, such as average and max pooling. Most recently, Transformer Networks have been used for the task of multiview 3D shape reconstruction in the work of Yagubbayli et al. [76] and Wang et al. [66]. The Transformer Networks again have the advantage of using attention for view aggregation. However, our approach uses 3D voxel-based max pooling for view aggregation to avoid the dependency on the number of views.

All these approaches use occupancy grid representation with a resolution of 32^3 , except for the work of Xie et al. [74] that also presented some results for an output resolution of 128^3 . With this small resolution, objects with fine details cannot be represented. Additionally, the surfaces are not smooth in occupancy grid representation as shown in Figure 2.1. The proposed approach instead uses a TSDF based representation which is a more dense and smooth representation compared to an occupancy grid representation.

2.3. 3D Shape Completion

The task of 3D shape completion is also closely related to the task of 3D shape reconstruction. Shape completion can be divided into two types, namely direct methods and data-driven methods. Data-driven shape completion methods usually use depth images or 3D data in different representations,

like point clouds or voxel grids directly. Extensive work has been done in the literature on shape completion, and the reader is requested to refer to the latest literature on shape completion. The three seminal works are from Wu et al. [72], Dai et al. [10] and Wu et al. [70]. The 3D-ShapeNets from Wu et al. was the first method that proposed converting depth images into 3D voxel grids using a Deep Belief Network (DBN) [25]. Wu et al. proposed not only a joint object recognition and shape completion network but also the widely used ModelNet dataset, which is a large-scale 3D CAD model dataset. Another prominent work is the 3D-EPN network from Dai et al., which operates on partial depth scans obtained using volumetric fusion from Curless et al. [8]. The 3D-EPN network uses 3D convolutional networks and non-parametric based shape synthesis for generating shape completions at a resolution of 128^3 . Our approach also uses 3D convolutional networks similar to Wu et al. and Dai et al. However, instead of using depth information directly, the proposed approach lifts the 2D feature maps of RGB images into 3D using camera intrinsics and extrinsics parameters.

The later work of Wu et al. [70] proposed the ShapeHD network which uses RGB images to predict depth, normal, and silhouette images. The depth image is later passed into the shape completion network for generating a voxel grid of resolution 128^3 . An adversarially pretrained CNN is used for calculating a "naturalness" loss for the shape completion network, which helps avoid blurry outputs. Our approach, however, uses an autoencoder from Denninger et al. [14] for generating TSDF volumes with resolutions as high as 512^3 .

3. Methodology

3.1. Image Formation

3.1.1. Pinhole Camera Model

The pinhole camera model [43] is a simple camera model that explains the relationship between the coordinates of a point in 3D and its projection onto the image plane. However, the model does not account for distortion and blurring caused by the lenses. Usually, the distortion increases from the center of the image to the edge of the image. However, distortion can be accounted for in the transformation equations from the 3D coordinates to the 2D pixel coordinates. An illustration of a pinhole camera model is shown in Figure 3.1.

Projection

In a pinhole camera model, a 3D point P_w is projected into its corresponding pixel p using the perspective transformation. Without accounting for the image distortion, the perspective transformation of the pinhole camera model is given by Equation 3.1. Here, P_w represents the 3D point in the world coordinate system, p is the 2D pixel point in the image plane where $p = (u, v)$, K is the camera intrinsic matrix, R and t are the rotation matrix and translation vector respectively for transforming the coordinates from the world to the camera coordinate system, and s is the projective scaling which is not part of the camera model.

$$s p = K [R|t] P_w \quad (3.1)$$

The camera intrinsic matrix K projects 3D points in the camera coordinate system to 2D pixel coordinates as shown in Equation 3.2, where P_c is a 3D point in camera coordinate system and p is the 2D pixel point. The camera intrinsic matrix K is composed, as shown in Equation 3.3, of focal lengths f_x and f_y expressed in pixel units, as well as the principal points c_x and c_y , which are usually close to the center of the image. Equation 3.4 is derived by replacing the camera intrinsic matrix K in Equation 3.2 with Equation 3.3. The camera intrinsics matrix K remains constant for a scene unless the focal length of the camera is changed. If the focal length is changed, the camera intrinsic matrix K should be scaled up or scaled down accordingly.

$$p = K P_c \quad (3.2)$$

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \quad (3.4)$$

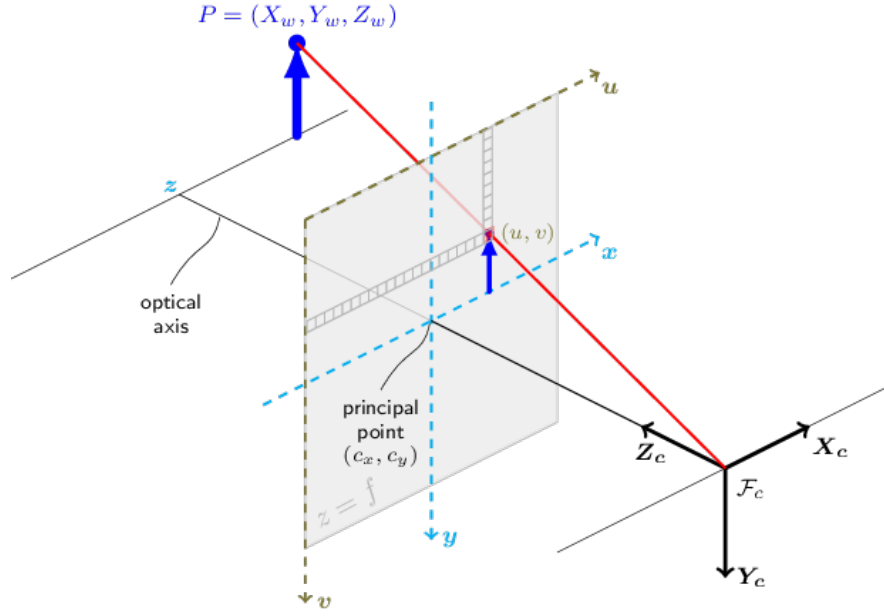


Figure 3.1.: Figure shows the pinhole camera model with P_w in world coordinate system, P_c in camera coordinate and p in the 2D image plane. The image is taken from OpenCV Camera Calibration and 3D Reconstruction documentation [47].

The 3-by-4 perspective transformation is given by Equation 3.5 where $x' = X_c/Z_c$ and $y' = Y_c/Z_c$ in normalized camera coordinates. More details for perspective transformation are explained in 3.1.2. Equation 3.6 transforms the 3D points from the world coordinate system to the camera coordinate system. The homogeneous transformation is composed of a 3-by-3 rotation matrix R and a 3-by-1 translation vector t as shown in Equation 3.7. The 3-by-3 rotation matrix can also be represented as a 3-by-1 rotation vector using Euler angles. However, the 3-by-3 rotation matrix makes the math easier. There are other rotations representations as well, like quaternions, and it is easy to switch from one rotation representation to another. Each rotation representation has its own advantages and disadvantages. Equation 3.8 is derived from Equation 3.6 using the homogeneous transformation as specified in Equation 3.7.

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3.5)$$

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w \quad (3.6)$$

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.8)$$

Equation 3.11 shows the projection of the 3D points in the world coordinate system to 2D pixel coordinates which can be simplified further to get Equation 3.12 and Equation 3.13.

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.9)$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.10)$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.11)$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x X_c / Z_c + c_x \\ f_y Y_c / Z_c + c_y \end{bmatrix} \quad (3.12)$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.13)$$

To summarize, the 3D point in the world coordinate system is transformed into a 3D point in the camera coordinate system, using Equation 3.13. The transformed 3D point in camera coordinate system can now be projected into the 2D image plane using Equation 3.12.

Unprojection/Backprojection

In the projection subsection, equations were presented for converting the 3D point in the world coordinate frame to the 2D pixels in the image plane. The whole projection process can be reversed for going from the 2D pixels in the image plane to the 3D point in the world coordinate frame without depth information. The reverse process of projection is called unprojection or backprojection. Equation 3.14 presents the conversion of a 2D pixel coordinate in the image plane to a 3D point in the camera coordinate frame. Note the depth of point p is unknown, so Z_c is taken to be s as defined in the projection subsection.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} \frac{(u-c_x)*Z_c}{f_x} \\ \frac{(v-c_y)*Z_c}{f_y} \\ s \end{bmatrix} \quad (3.14)$$

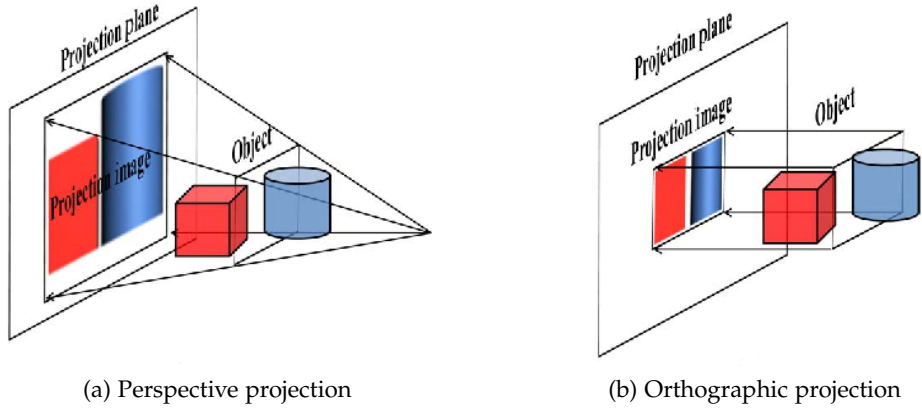


Figure 3.2.: Figure shows how the 3D objects are projected in the 2D image plane using perspective projection in (a) and orthographic projection in (b). The images are taken from Jia et al. [31].

Equation 3.6 changes the 3D point from the world coordinate frame to the camera coordinate frame. The same equation with a slight modification can be used for converting the 3D point in the camera coordinate frame to the world coordinate frame. If the homogeneous transformation is moved to the left side of the equality in Equation 3.15, Equation 3.15 is derived. If the actual point representation of P_c and P_w is plugged in Equation 3.6 and the homogeneous transformation is expanded out, Equation 3.16 is derived where P_c and P_w have been homogenized similar to as in Equation 3.8. Equation 3.8 shows how to convert a 3D point in the camera coordinate frame to the world coordinate frame using the inverse of the homogeneous transformation.

$$P_w = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} P_c \quad (3.15)$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3.16)$$

3.1.2. 3D Projections

Perspective Projection

The perspective projection keeps the Z_c component such that the 2D points $x' = X_c/Z_c$ and $y' = Y_c/Z_c$ are in normalized camera coordinates, as shown in Equation 3.17. Perspective projection is shown in Figure 3.2a.

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3.17)$$

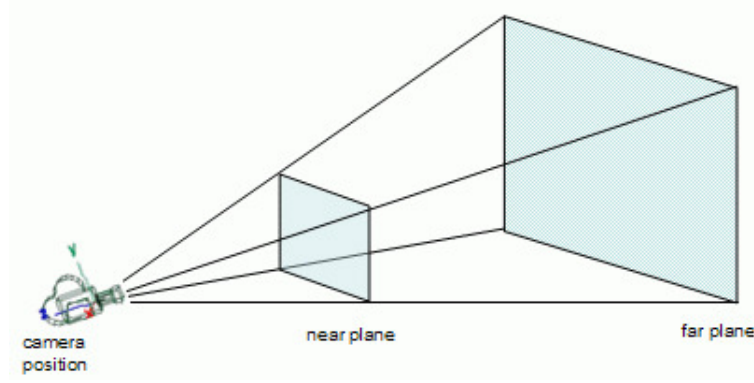


Figure 3.3.: Figure shows the perspective view frustum. The perspective view frustum takes the shape of a truncated pyramid. The image is taken from Lighthouse3d.com [41] website.

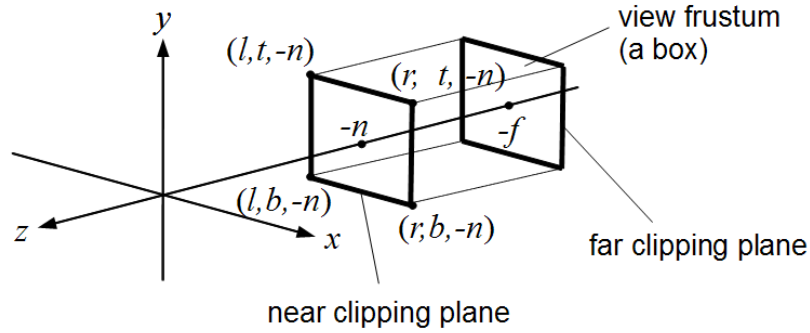


Figure 3.4.: Figure shows the orthographic view frustum. The orthographic view frustum takes the shape of a cuboid. The image is taken from Martin Kraus [36] and it is originally published on Wikipedia.

Orthographic Projection

The orthographic projection eliminates the Z_c component such that the 2D points $x' = X_c$ and $y' = Y_c$ are in normalized camera coordinates, as shown in Equation 3.18. Orthographic projection is shown in Figure 3.2b.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3.18)$$

3.1.3. View Frustum

The view frustum takes a shape of a 3D volume indicating what would be visible on the image plane. A truncation is usually performed at the near plane and far plane, which means everything before the near plane and after the far plane is clipped out. The shape of the view frustum 3D volume depends on the type of projection used. If the perspective projection is used, the view

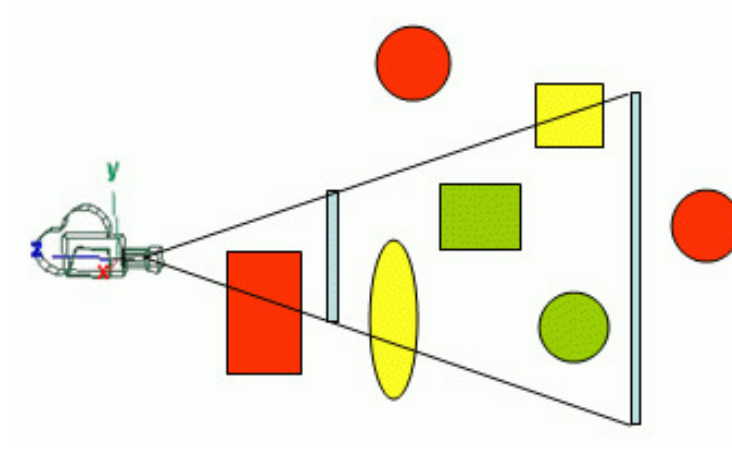


Figure 3.5.: Figure shows view frustum culling being applied to green, red, and yellow objects. Taking the shape of the view frustum into account, all the green objects are rendered, yellow objects are partially rendered and none of the red objects are rendered. The image is taken from Lighthouse3d.com [41] website.

frustum takes the shape of a truncated pyramid as shown in Figure 3.3. If the orthographic projection is used, the view frustum takes the shape of a cuboid as shown in Figure 3.4.

After the shape of the view frustum is known, view frustum culling can be applied to avoid rendering objects outside the viewing frustum, thus saving computational resources. Figure 3.5 shows view frustum culling is being applied for the case of perspective projection. Everything inbetween the near and the far plane is captured on the image screen, which means the green objects are rendered fully while the yellow objects are rendered partially, since part of the object is outside the viewing frustum. None of the red objects are rendered because they are outside the viewing frustum as shown in Figure 3.5 with the current setting.

3.2. 3D Data Representations

There exists several ways to represent 3D data. The underlying methods change depending on the type of 3D data representation used. The most common 3D data representations are described below, these are shown in Figure 3.8.

3.2.1. Point Cloud

A point cloud is an unstructured and a memory-efficient representation that expresses the shape geometry as 3D continuous points. However, point clouds do not have the concept of free space, and do not capture geometry well due to the points spacing. However, point clouds have received a lot of attention recently, and there are a lot of methods in the literature, like PointNet [53] and PointNet++ [54], which work directly on point clouds as an input.

3.2.2. Binary Occupancy Grid/Voxel Grid

A binary occupancy grid or voxel grid is a 3D representation that encodes the geometry as a 3D grid. Each cell in the 3D grid encodes whether it is occupied or empty. A value of zero is used for

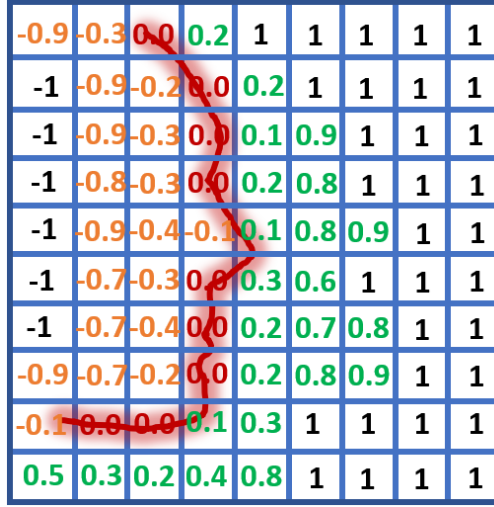


Figure 3.6.: Figure shows a 2D TSDF with a red curve depicting the surface with all the values inside the surface being negative, on the surface being zero, and outside the surface being positive. As can be seen in the figure, a truncation between -1 and 1 is applied. The same concept can also be extended to 3D. The image is taken from Arm Community website [44].

empty cells and a value of one is used for filled cells. The cells maintain spatial information, which means that traditional deep learning architectures can also be applied to this data type. For smaller resolutions, a binary occupancy grid does not capture fine geometric details. As the resolution size increases, the representation suffers from computation and memory-related issues because of the encoding of the occupied and the free space.

3.2.3. Truncated Signed Distance Function (TSDF)

This 3D data representation encodes geometry as a gradient field in terms of the signed distance to the closest surface where all the values inside the surface are negative, the values on the surface are zero and the values outside the surface are positive. Since these signed distance values can be unbounded, the values are usually truncated in a range, for example, -1 to 1 as shown in Figure 3.6; hence, the name Truncated Signed Distance Function (TSDF). TSDF is usually a dense representation, thus it provides a better gradient flow through Deep Neural Network as compared to a binary occupancy grid representation.

Similar to a binary occupancy grid, traditional deep learning operations like convolution can be applied directly to this 3D data representation. Similarly, it also suffers from the same computation and memory-related issues. Thus, it is desirable to have a binary occupancy grid and a TSDF grid of higher resolution for capturing more finer details. Figure 3.7 shows the memory comparison between binary occupancy/voxel grid and TSDF volume for different resolutions. For higher resolutions, TSDF volumes take up more memory than binary occupancy grids. To obtain a mesh from a TSDF volume, a meshification algorithm, like Marching Cubes [42] can be used.

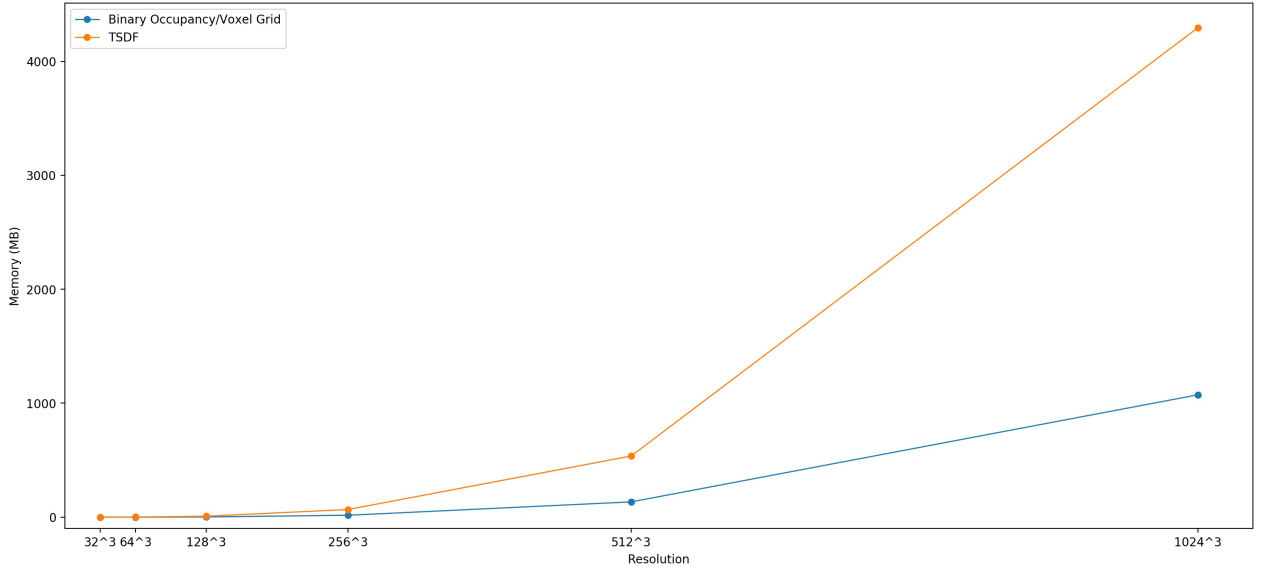


Figure 3.7.: Figure shows the memory comparison between binary occupancy/voxel grids and TSDF volumes of different resolutions. As resolution increases, TSDF volumes take up more memory than binary occupancy grids. For example, a volume of size 512^3 corresponds to more than 134 million voxels. For storing this volume, a binary occupancy grid representation would take approximately 134.22 MB while a TSDF volume representation would take approximately 536.87 MB.

3.2.4. Mesh

A mesh representation explicitly encodes the surface representation using polygons. It is challenging to design deep learning networks that handle mesh data directly because applying convolutional and pooling operations is not directly possible, and most previous works generated mesh from an intermediate TSDF representation. However, there are already works in literature, like MeshNet [17] and MeshCNN [22] that take mesh data directly as an input for a Deep Neural Network. There are also methods like Deep Marching Cubes [40] that provide meshes directly as output from well-sampled point clouds.

3.3. Truncated Signed Distance Function (TSDF) Generation

3.3.1. Problem Statement

The following equations come from Bærentzen et al. [1] and Denninger et al. [14]. In Equation 3.19, \mathcal{T} represents the triangle mesh which is a union of all the triangles T_i where $i \in [1, N]$, N being the total number of triangles.

$$\mathcal{T} = \bigcup_{i=1}^N T_i \quad (3.19)$$

In a 3D voxel grid, each of the voxels contains a scalar that stores the value of the shortest distance to a triangle mesh. The distance from the center of each voxel \mathbf{v} to the triangle mesh is

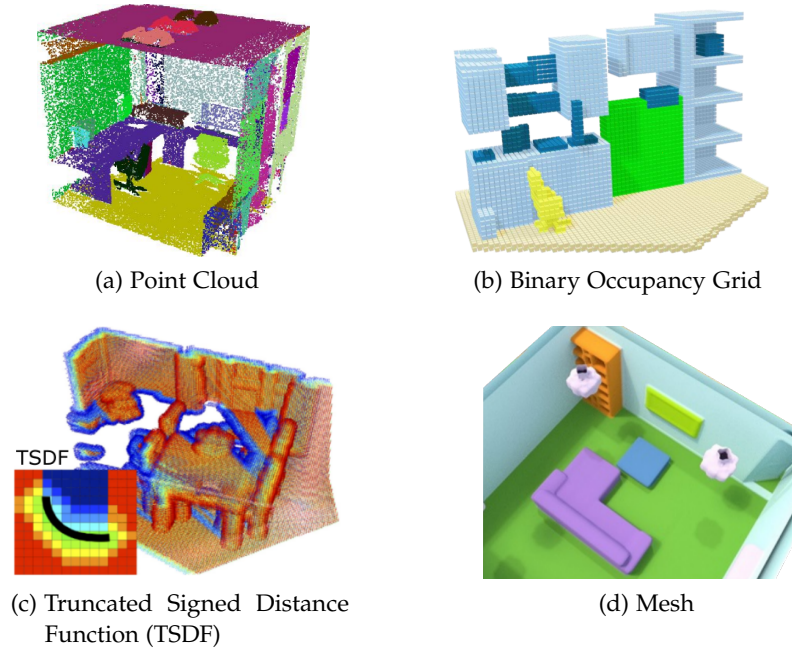


Figure 3.8.: Figure shows the most commonly used 3D data representations. Different methods use different 3D data representations. The images are taken from Wang et al. [68], Wang et al. [69], Song et al. [59], and Dai et al. [11].

defined in Equation 3.20, where \mathbf{p} is center of voxel $\mathbf{v} = (v_x, v_y, v_z)$. If Equation 3.20 is expanded out further, it can be written out as in Equation 3.21 which shows the shortest distance is computed against all the triangles T_i in a triangle mesh \mathcal{T} .

$$d(\mathbf{p}, \mathcal{T}) = \min_{t \in \mathcal{T}} \|\mathbf{p} - t\| \quad (3.20)$$

$$d(\mathbf{p}, \mathcal{T}) = \min_{i \in [1, N]} \left(\min_{t \in T_i} \|\mathbf{p} - t\| \right) \quad (3.21)$$

A truncation is applied on the signed distance field to get the truncated signed distance field or TSDF. If a truncation of $-\sigma_{tsdf}$ to σ_{tsdf} is applied on the signed distance field represented in Equation 3.21, Equation 3.22 specifies the TSDF for all the voxels \mathbf{v} in the 3D voxel grid V . The next section discusses methods for computing the shortest distance $d(\mathbf{p}, \mathcal{T})$ for each voxel \mathbf{v} .

$$V[\mathbf{v}] = d_p = \max(-\sigma_{tsdf}, \min(\sigma_{tsdf}, d(\mathbf{p}, \mathcal{T}))), \forall \mathbf{v} \in \Omega_v \quad (3.22)$$

3.3.2. Methods

In literature, there exists a number of ways for computing TSDFs from triangle meshes. The fundamental problem that all these methods solve is essentially computing the distance for all the points in a voxel grid to all the triangles in a mesh. To determine if a point is inside or outside the surface, a surface normal vector can be used. However, it can be a challenge to compute normals on vertices of a mesh and edges, since it is discontinuous.

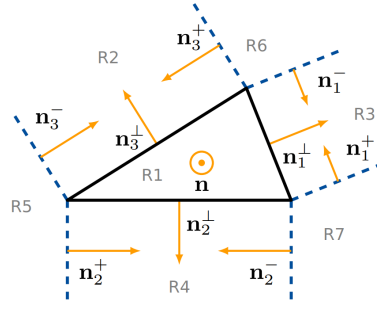


Figure 3.9.: Figure shows all the normals in orange that are used for calculating the distance $d(p, \mathcal{T})$. The orthogonal planes are shown in blue and dashed. The original figure is taken from Denninger et al. [14] and adapted afterward.

Most methods are not efficient enough for computing high-resolution TSDF voxel grids. For example, it can be challenging to compute a TSDF volume of resolution 512^3 which contains more than 134 million voxels. Besides the heavy computation involved in computing the high resolution TSDF volumes, storing such a volume with 134 million voxels for just a single 3D mesh requires significant memory. For example, a 512^3 TSDF volume required 537 MB of memory for storage in 32-bit floating point format. Denninger et al. [14] introduced an efficient way for computing the distance as defined in Equation 3.21 for all the 134 million voxels corresponding to a TSDF volume of resolution 512^3 in the order of seconds using flood filling and octrees. The method precomputes the 10 vectors for each triangle T_i . These vectors, as shown in Figure 3.9 correspond to the normal vector \mathbf{n} of the triangle plane \mathfrak{P} , the vector \mathbf{n}^\perp that are orthogonal to the edges of T_i and lie inside the plane \mathfrak{P} , and the vectors \mathbf{n}^+ and \mathbf{n}^- that are parallel to the edges of T_i . The first thing to do is to check if p is closer to the surface, the edge or the vertex. This is done in the paper by computing the distance between \mathfrak{P} and p , and checking if the projection onto \mathfrak{P} is inside the triangle T_i , using the normal vector \mathbf{n}^\perp . If this is the case, $d(p, \mathcal{T})$ equals $d(p, \mathfrak{P})$, otherwise p is closer to an edge or a vertex. For final checks, vectors \mathbf{n}^+ and \mathbf{n}^- are used for distance $d(p, \mathcal{T})$ calculation.

3.4. Deep Learning

Deep learning [38] is a class of machine learning algorithms that use artificial neural networks for learning representations of data. Neural networks saw its resurgence thanks to the availability of large datasets, better algorithms, and powerful computational resources. Since then, deep learning has achieved success in problems like machine translation, visual object recognition, and 3D reconstruction. For a more detailed discussion about deep learning, the reader is requested to refer to the Deep Learning book by Goodfellow et al. [21].

3.4.1. Residual Network (ResNet)

ResNet, originally introduced in the paper from He et al. [23], was the state-of-the-art method for the task of image classification on the ImageNet [58] dataset when the architecture was introduced back in 2015. Because of this, ResNet found widespread use in the field of computer vision and deep learning. ResNet is used not only for image classification tasks but also for other tasks in

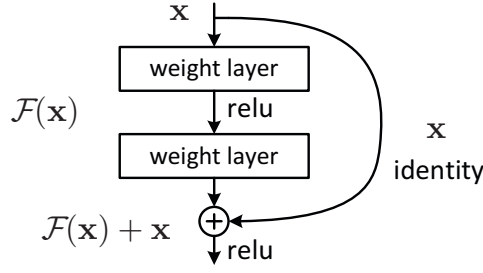


Figure 3.10.: Figure shows a simple residual block that makes use of a skip connection. The skip connection adds the output from the previous layer. The image is taken from the original ResNet [23] paper.

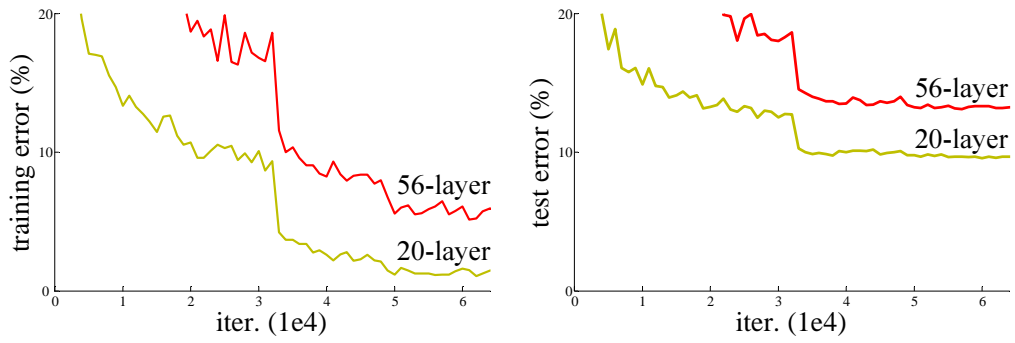


Figure 3.11.: Figures show that both the training and test performance of the neural network suffers when the number of layers are increased from 20 to 56. ResNets alleviated this problem through the use of skip connections. The image is taken from the original ResNet [23] paper.

computer vision, such as object detection and face recognition due to its strengths in representation learning.

The important contribution that the authors of ResNet made was being able to train exceptionally deep neural networks without suffering from the vanishing gradient problem as explained in Glorot et al. [20]. In theory, neural networks with more layers can learn more complex representations. However, this was not the case practically. As the number of layers was increased in the neural network, the training and test performance decreased in comparison to a network with less layers. Figure 3.11 shows that a 20-layer network performs better on both the training and the test data than a 56-layer network. An important thing to note here is that the 56-layer network is not overfitting the data, as it performs worse on both training and test data.

To overcome the challenge in training deep neural networks, the core idea of ResNet was to introduce "identity shortcut connection" or skip connections as shown in Figure 3.10 which allowed the authors to train deeper neural networks with hundreds or thousands of layers, like in the work of He et al. [24]. Instead of allowing the network to just learn $\mathcal{F}(x)$ output, the residual blocks force the network to learn $\mathcal{F}(x) + x$ output. The additional path, firstly, provides an alternate shortcut for the gradient to pass through which helps to alleviate the vanishing gradient problem. Secondly, it ensures that the higher layers perform as well as the lower layers since some of the higher layers can learn just the identity mapping, which is possible through the use of skip connections.

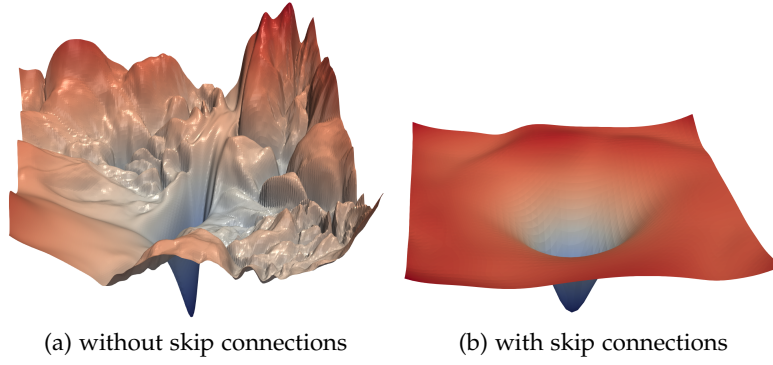


Figure 3.12.: Figure shows the loss surfaces of ResNet-56 with and without skip connections. By adding skip connections, the loss surface becomes smoother leading to better convergence. The image is taken from Li et al. [39].

Later on, the work of Li et al. [39] showed how by adding the skip connections, the loss landscape of a neural network becomes smooth leading to better convergence as can be seen in Figure 3.12. Without skip connections, the optimizer of a neural network can get stuck in local minima but with skip connections, global minima can be found more easily.

ResNet exists in different versions, depending on the number of layers the network has, as indicated in the architectural name. Each ResNet network consists of a collection of ResNet blocks. Different ResNet versions use different residual blocks, such as a standard residual block or a bottleneck residual block as shown in Figure 3.13. The bottleneck residual block initially reduces and then increases the number of channels for the 3×3 convolution using 1×1 convolution. Using this approach, the number of channels can be increased while keeping the same number of parameters in a ResNet.

3.4.2. Autoencoder

Autoencoders, first introduced in Rumelhart et al. [57], are a type of an unsupervised neural networks that are used to encode an input into a compressed and useful representation, also called a latent representation, then decode it back, such that the reconstructed input is as similar as possible to the original input. The encoding part of the autoencoder is known as an encoder while the decoding part of the autoencoder is known as a decoder as shown in Figure 3.14. The other main components in an autoencoder architecture are an input layer, a hidden layer, and an output layer. A similarity loss function is applied to the output of the autoencoder and the original input during the training process.

For image data, it is more common to use convolutional layers as the data is not only processed spatially but also results in the reduction of parameters in both the encoder and decoder parts of the autoencoder. There are different types of autoencoders like Sparse Autoencoder (SAE), Denoising Autoencoder (DAE) and Variational Autoencoder (VAE). The VAE is an autoencoding approach where the latent representation learns probability distribution parameters for modeling the input data as proposed by Kingma et al. [35] in 2014. Some of the prominent application areas in which autoencoders have been used extensively include anomaly detection, dimensionality reduction, image processing, and machine translation.

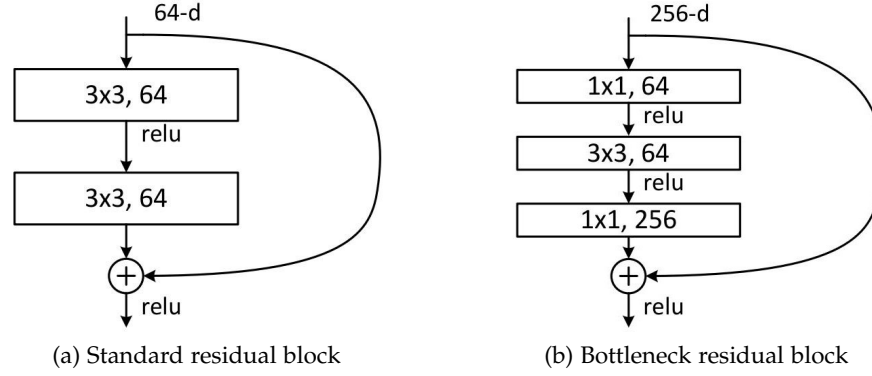


Figure 3.13.: Figures show the two different types of residuals blocks that are used in ResNet. The standard residual block, which is used in ResNet-18 and ResNet-34 architectures, has two convolutional layers of filter size 3×3 while the bottleneck residual block, which is used in ResNet-50, ResNet-101, and ResNet-152 architectures, has three convolutional layers of sizes 1×1 , 3×3 and 1×1 where the 1×1 filters firstly increase and then decrease the channels while the 3×3 filter acts as a bottleneck. The rectangles represent convolutional layers. The filter size and number of channels are mentioned inside the rectangle. The image is taken from the original ResNet [23] paper.

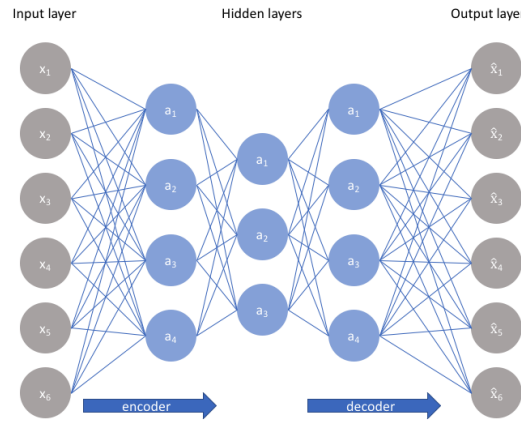


Figure 3.14.: Figure show an autoencoder with an input layer, a hidden layer and an output layer. An autoencoder is used extensively for compressing and decompressing data in an unsupervised way. The image is taken from Jeremy Jordan's blog [32].

4. Our Approach

4.1. Problem Statement and Notation

Given n number of RGB images $I_c : \Omega_c \rightarrow [0, 255]^3$ of dimension $u \times u \times 3, u \in \mathbb{N}$ where $\Omega_c \subset \mathbb{R}^2$ and n number of camera poses as input, the task of multiview 3D shape reconstruction is to generate a mapping from 2D image coordinates $\mathbf{x}_c = (x_c, y_c)$ to 3D object coordinates $\mathbf{x}_s = (x_s, y_s, z_s)$. The output is a high-resolution 3D TSDF $V : \Omega_v \rightarrow [-\sigma_{tsdf}, \dots, \sigma_{tsdf}]$ of dimension $w \times w \times w$ where $\Omega_v = \{0, \dots, 511\}^3$.

4.2. Input and Output

The overall network takes n RGB images, n camera extrinsics, and a fixed set of camera intrinsics for any n number of views as an input which makes the proposed approach independent of the number of viewpoints.

For output, a TSDF representation offers the most benefits when compared with the other 3D representation techniques which are discussed in Section 3.2. The ground truth target is encoded in TSDF representation which is explained in Chapter 5. Therefore, the output of the network should be a high-resolution intermediate TSDF representation which can then be converted into the final mesh representation using the Marching Cubes [42] algorithm as explained in Subsection 3.2.3. However, a TSDF volume with a bigger resolution takes more memory as also explained in Subsection 3.2.3. Furthermore, it is desirable to have TSDF volumes with a bigger resolution, such as a resolution of 512^3 , because such high resolutions can capture finer details.

So to benefit from the high-resolution while at the same time reducing the memory footprint, the proposed approach uses the pretrained autoencoder from Denninger et al. [14], which can be used for both compressing and decompressing the TSDF volumes of resolution 512^3 . Instead of originally generating the higher resolution TSDF volume, the network generates a compressed representation as output which is compared against the compressed representation of the ground truth target during training. The compressed representation for the ground truth target is generated using the encoder part of the autoencoder from Denninger et al. During testing, the compressed representation is decompressed to the original resolution of 512^3 using the decoder part of the autoencoder.

4.3. Architecture

The proposed architecture for training consists of a 2D network, a backprojection layer, and a 3D network. The 2D network is used to calculate the 2D feature maps, the backprojection layer for lifting the 2D feature into 3D using camera intrinsics and extrinsics, and the 3D network for refining the 3D output further. During test time, an additional autoencoder from Denninger et al. [14] decodes the output of the training architecture. Figure 4.1a shows the network architecture

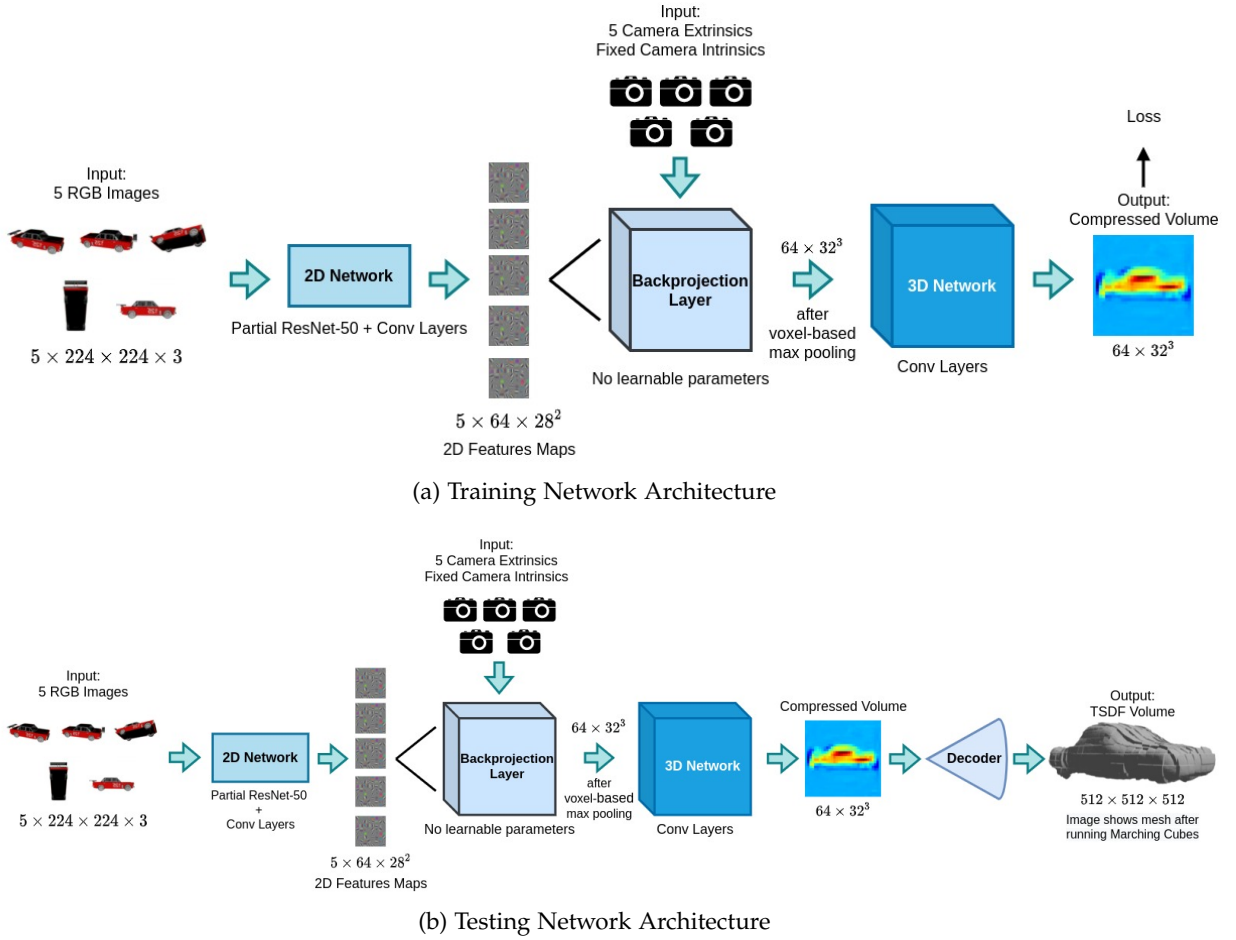


Figure 4.1.: Figure shows the network architectures for training in (a) and testing in (b). During the training phase only the 2D network, the backprojection layer, and the 3D network is used while only during the testing phase, an additional decoder network is used. The network in (a) is trained end-to-end using a single optimizer.

for training, while Figure 4.1b shows the network architecture for testing. The different components that make up the whole architecture are explained further below.

4.3.1. 2D Network

The input to the 2D network is n RGB images to calculate feature maps which are used for recovering the 3D shape of the object. The camera parameters are not required here. The 2D network comprises the initial 7 bottleneck residual blocks (explained in Subsection 3.4.1) of ResNet-50 [23] and 4 subsequent convolutional layers of filter size 3×3 including BatchNorm [29] and ReLU [46] layers for feature extraction. The pretrained ResNet-50 layers are trainable which means the gradient backpropagates through the layers until the input layer. Pretrained ResNet-50 for the task of image classification on the ImageNet [58] dataset is used as in Pix2Vox [73], Pix2Vox++ [74] and Spezialetti et al. [60]. Since the image classification task is not closely related to the shape reconstruction, only the initial pretrained layers of ResNet-50 are used. These layers can help in not only detecting edges and other low-level details, but also making the training faster since the

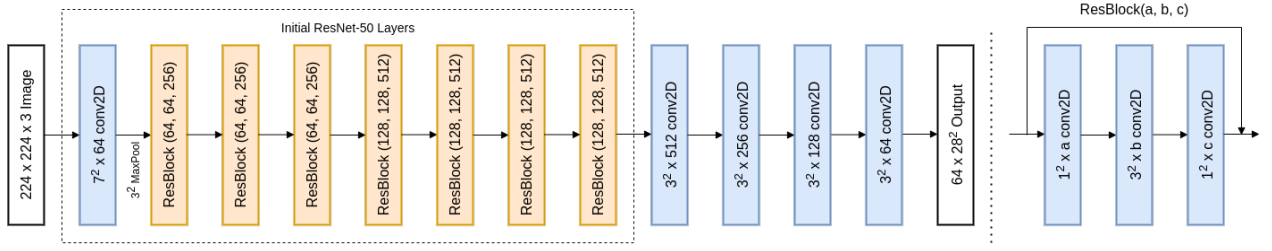


Figure 4.2.: Figure shows the different layers in the 2D network. Initial layers of the 2D network make use of pretrained weights from ResNet-50. 7 bottleneck residual blocks are used from ResNet-50. The bottleneck residual block is depicted in the figure on the right. The 4 subsequent layers consist of regular convolutions that maintain the same spatial dimension. The 2D network and the figure is inspired from Xie et al. [74].

weights are not randomly initialized, as explained in the paper from Oquab et al. [48].

The ResNet layers reduce the spatial dimension of the RGB input from 224×224 with three color channels to 28×28 with 512 feature channels. This output is then passed through the next set of four convolutional layers. The first convolutional layer preserves the 512 feature channels and the spatial dimension of size 28×28 . The remaining three convolutional layers reduce the feature channels by a factor of two until 64 feature channels remain, while preserving the spatial dimension of size 28×28 . The final output from the 2D network then has a dimension of 28×28 with 64 feature channels. The architecture of the 2D network is shown in Figure 4.2.

4.3.2. Backprojection Layer

The backprojection layer, inspired from 3DMV [9], is fully differentiable, and is the main component that connects the 2D network with the 3D network, making the whole method end-to-end trainable. The primary purpose of having this layer is to lift the 2D features into 3D for which it makes use of camera parameters, namely the camera extrinsics and camera intrinsics. The backprojection layer calculates the associations between the 3D voxel grid and the 2D feature maps that are provided by the 2D network.

Since the minimum and the maximum depth is already prespecified, and the image dimensions are already known, Equation 3.14 from Section 3.1.1 can be used for setting up the near and far plane of the frustum bounds. This corresponds to the 8 corner coordinates: 4 corner coordinates for the near plane and 4 coordinates for the far plane. These corner coordinates are in the camera coordinate frame and can be transformed to the world coordinate frame using camera extrinsics which are provided as an input. Finally, the corner points are transformed into the grid coordinate frame. The corner points are then clamped within the voxel grid dimensions of size $w_{3d} \times h_{3d} \times d_{3d}$ as the size of reconstruction output is already specified. This provides the corner points in the grid coordinate frame for the near and the far plane of the frustum bounds.

Afterwards a voxel grid of size $w_{3d} \times h_{3d} \times d_{3d}$ is set up as defined initially in the grid coordinate frame, and a check is performed if the voxel grid is within the size of the frustum bounds. The 3D grid coordinates are transformed into the world coordinate frame, and then the camera coordinate frame using Equation 3.6 from Section 3.1.1. The inverse of the camera extrinsics parameters is then used for transforming from world coordinate frame to camera coordinate frame. Then 3D points in the camera coordinate frame are projected into the 2D image plane using Equation 3.2

from Section 3.1.1. A check is also performed on these 2D pixel points to check if they are between the valid and known image dimensions.

The 3D voxel grid coordinate and its corresponding 2D pixel coordinates (3D-2D associations) are stored as indexes. During the forward pass of the backprojection layer, the network can map a 2D feature map (output of the 2D network) of one viewpoint into the 3D voxel grid coordinates using the computed indexes. Similarly, during the backward pass, the network can map the upstream gradients into the 2D pixel points again using the computed indexes. The indexes are computed on the fly since GPU memory is limited. The process is repeated for the 2D feature map of the other views.

Furthermore, to make the approach invariant to the number of views, voxel max-pooling is used similar to Dai et al. [9]. The reason for using voxel max-pooling is that some voxels will be associated with multiple 2D pixels from all views. Voxel max-pooling, which is calculated across the number of feature channels, combines these projected features from all the input views.

To summarize, n 2D feature maps of size $w_{2d} \times h_{2d}$ with n_{feat} input feature channel are projected into 3D with a final output size of $w_{3d} \times h_{3d} \times d_{3d}$ with n_{feat} output feature channels where n is equal to the number of input views. In our approach, each of the n 2D feature maps has a size of $64 \times 28 \times 28$, where $n_{feat} = 64$, $w_{2d} = 28$ and $h_{2d} = 28$, which is then projected to a 3D volume of size $64 \times 32 \times 32 \times 32$, where $n_{feat} = 64$, $w_{3d} = 32$, $h_{3d} = 32$ and $d_{3d} = 32$.

$$n \times n_{feat} \times w_{2d} \times h_{2d} \rightarrow n_{feat} \times w_{3d} \times h_{3d} \times d_{3d}$$

4.3.3. 3D Network

The output from the backprojection layer is already converted into 3D TSDF grid coordinates. However, to refine the output further, a 3D network is used which applies further convolutional layers. More precisely, the 3D network uses seven convolutional layers with filter size $3 \times 3 \times 3$ along with BatchNorm [29] and ReLU [46] layers. The ReLU activation is not applied to the last layer because the output of the last layer should not be constrained. The input to the 3D network has a size of 32^3 with 64 feature channels (size similar to the output of the backprojection layer). Since the convolutional layers with padding maintain the same spatial dimension as well as the number of feature channels, the output of the 3D network also has a dimension of 32^3 with 64 feature channels, which represents the size of the compressed latent TSDF volume. Figure 4.3 shows the architecture of the 3D network.

4.3.4. Autoencoder

The output from the 3D network has a dimension of 32^3 with 64 feature channels which is then passed into the decoder part of the pretrained autoencoder network from Denninger et al. [14]. The decoder network takes the 3D network's output as input, which is already a compressed representation, to generate the final decompressed TSDF volume with a dimension of 512^3 . Similarly, the ground truth target volume, which is also in a compressed representation, is also decompressed using the same decoder network. This is done to evaluate the difference between the prediction and the ground truth target volume in their original representations. Also notable is that the whole network architecture is trained using the compressed representation as explained in the Data Generation Section 5.2.2.

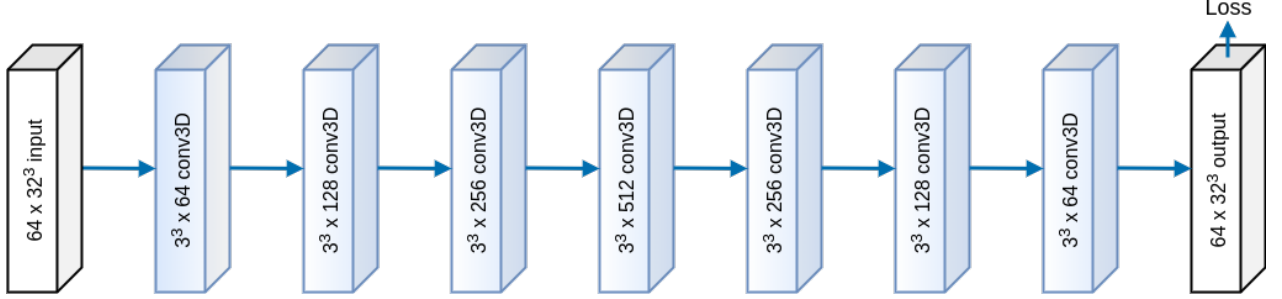


Figure 4.3.: Figure shows the different layers that are a part of the 3D network. All layers consist of a convolutional layer, a BatchNorm layer, and a ReLU layer except for the last layer which does not use a ReLU layer. The layers maintain the same spatial resolution of size 32^3 .

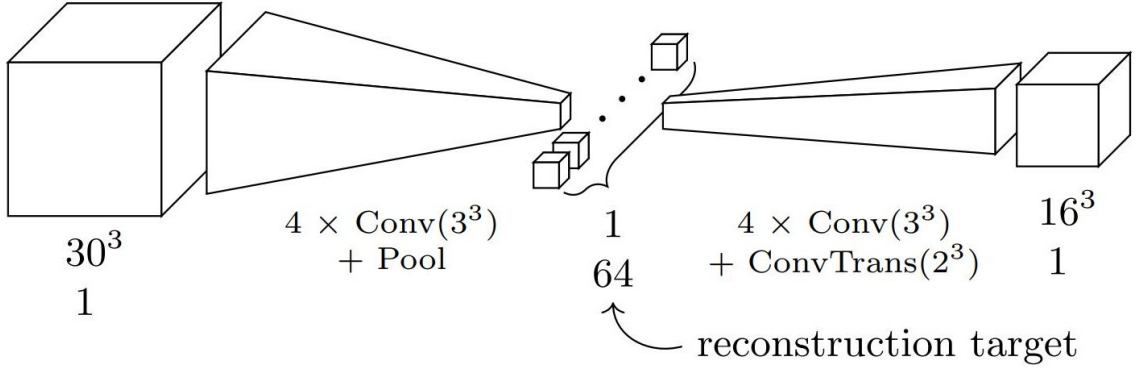


Figure 4.4.: Figure shows the autoencoder used to decompress the TSDFs into a latent representation of size 64×32^3 during the training phase and later to decompress in the original resolution of 512^3 during the testing phase. The image is taken from Denninger et al. [14].

5. Experimental Setup

5.1. Dataset

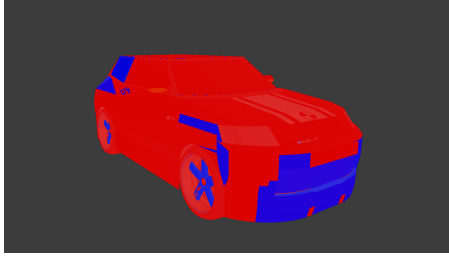
The generated dataset is based on the ShapeNet [4] dataset, which is a large-scale dataset of 3D shapes organized according to the WordNet [16] hierarchy with more than 220,000 unique 3D models classified into 3,135 categories. The ShapeNetCore dataset is a subset of the original ShapeNet dataset. It has two versions, namely the ShapeNetCore v1 dataset which has 55 object categories and the ShapeNetCore v2 dataset which has 57 object categories. Our approach specifically uses a subset of the ShapeNetCore v1 dataset with 13 different categories and 43,784 3D models in total, following the setting used in 3D-R2N2 [6] as shown in Table 5.1. The count column represents the number of 3D objects in a particular ShapeNet category. One 3D object from the rifle category with *source id* 4a32519f44dc84aabafe26e2eb69ebf4 is removed from evaluation as the 3D object is incorrect.

Table 5.1.: Table shows the original number of 3D objects contained in each ShapeNet category. Each ShapeNet category is identified with a unique Synset Id.

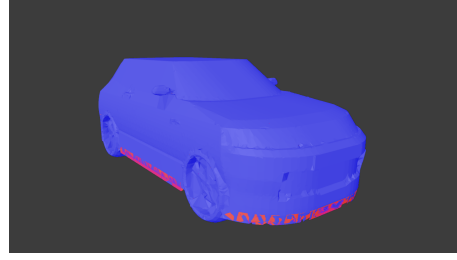
Object Category	Synset Id	Count
airplane	02691156	4045
bench	02828884	1816
cabinet	02933112	1572
car	02958343	7497
chair	03001627	6778
display	03211117	1095
lamp	03636649	2318
speaker	03691459	1618
rifle	04090263	2372
sofa	04256520	3173
table	04379243	8509
telephone	04401088	1052
watercraft	04530566	1939
Overall		43784

5.2. Synthetic Data Generation

The 3D objects are zero-centered and normalized between the inclusive range $[-1, 1]$. Specifically, the normalization is performed with respect to the biggest width, length or height of the 3D object.



(a) The normals pointing out incorrectly in the original object



(b) The normals pointing out correctly in the watertight object

Figure 5.1.: Figures show the normals orientation of the original ShapeNet object in (a) and the watertight version of the same object in (b). In the two figures, the color blue represents that the normal is pointing outwards from the surface while the color red represents that the normal is pointing inwards from the surface. As can be seen in figures, the watertight version has correct normals orientation which is essential for TSDF generation. The used object in both figures is from the car category with *source id* 1a1dcd236a1e6133860800e6696b8284. The normals are visualized using Blender [7].

The meshes are also made ‘watertight’ using the ManifoldPlus algorithm from Huang et al. [28]. Given a mesh with incorrect normal orientation and thin structures, the ManifoldPlus algorithm extracts watertight manifolds from meshes using the exterior faces between the occupied voxels and the empty voxels, and a projection-based optimization method. One disadvantage of using the ManifoldPlus algorithm is that watertight 3D objects lose the original texture information since the structure of the watertight mesh changes including the count of the vertices and faces. However, generally the count of the vertices and faces increases after making the meshes watertight using the ManifoldPlus algorithm. The TSDF pipeline from Denninger et al. [14], however, does not require the meshes to have texture information since it uses the face orientation information of a mesh for efficiently computing the TSDF grid. So for the RGBA images, the zero-centered and normalized 3D objects with the textures are used to make them as realistic as possible while the TSDF generation pipeline makes use of the watertight 3D meshes without the texture information.

5.2.1. RGB Images, Camera Intrinsics, and Camera Extrinsics

The RGB images are rendered using BlenderProc [12], which is a photorealistic data generation tool based on top of 3D modeling software Blender [7], which means that BlenderProc can make use of features like raytracing and hardware-accelerated rendering. In BlenderProc, the 3D shape is initially loaded after which light sources are placed in the specified position. Then, camera poses are generated relative to the 3D shape. Finally, the RGBA image is rendered. All these steps are explained further below:

3D Shape Loading

Since the 3D objects have already been centered and normalized, they can be loaded without any further processing. Most preprocessing has already been applied such as centering and making them watertight, which also fixes the normals. The 3D objects are stored in a .obj file with the

vertices, faces, and normals. An additional .mtl file contains material information. A majority of the 3D objects also contain texture images.

Lighting

Lighting is notably important for creating realistic images. A point light source with a simulated power of 1000 Watts at a radius of 5 is placed at six different locations around the 3D object, such that the maximum coverage of the light can take effect. These also correspond to the top and bottom of the 3D object, as well the four points in the plane, with all the points separated from each other at 90 degrees relative to the center of the 3D object.

Camera Sampling

The pose of the camera relative to the 3D object is also generated through random camera sampling. Using BlenderProc, the camera poses or camera extrinsics are sampled on the surface of a sphere with a radius of 2.5 around the 3D object. The camera intrinsics parameters that make up the camera matrix K , as explained in Equation 3.3, are also generated, which depends on the resolution of the RGBA images, among other factors.

Rendering

In BlenderProc, the Cycles render engine from Blender is used which is a raytracing-based renderer. Raytracing-based render engines produce photorealistic images that are not possible in classical render engines like OpenGL [45]. This is because raytracing-based render engines simulate the physical behavior of light, like reflections and shadows. Another benefit of raytracing-based render engine is that it can reduce the sim-to-real gap, which means that the neural networks trained on synthetic data generated using raytracing-based render engine have better generalization capabilities, as shown in the work of Hodaň et al. [26] and Denninger et al. [13].

Through the use of BlenderProc, RGBA images of a resolution of $256 \times 256 \times 3$ are rendered. Each of the 43784 3D ShapeNet objects has 24 renderings. In total, there are 1,050,816 RGBA images with a total size of approximately 68.7 GB. Figure 5.2 shows some of the RGB images rendered using BlenderProc. The BlenderProc config is specified in Appendix A.

5.2.2. Truncated Signed Distance Function (TSDF) Volumes

For the TSDF generation, the pipeline of Denninger et al. [14] introduces an efficient way of calculating TSDF volumes from meshes. A modified version of the TSDF pipeline is used in our approach. The modification made are related to the view frustum. Instead of the perspective view frustum, the modified version of the pipeline uses an orthographic view frustum such that the different input views correspond to the same TSDF volume. The pipeline generates TSDF volumes that originally have a resolution of 512^3 . By having a higher output resolution, finer details of the 3D objects can be captured, which would better associate the 2D image features with the 3D volume in our network.

Figure 5.3 shows different slices through the TSDF volume as the actual 3D TSDF volume is difficult to visualize while Figure 5.4 shows the meshes that are generated from TSDF volumes of different resolution.



Figure 5.2.: Figure shows the RGB renderings of the different ShapeNet categories, namely the airplane category with *source id* 4561def0c651631122309ea5a3ab0f04, car category with *source id* e4886a4d0c6ea960fe21694bd5f519d1, chair category with *source id* 1fccc2ac4bfd3da535836c728d324152, sofa category with *source id* 62e90a6ed511a1b2d291861d5bc3e7c8 and table category with *source id* 1011e1c9812b84d2a9ed7bb5b55809f8. All the renderings are generated using Blender-Proc [12].

Since the resolution of 512^3 is sufficiently high with 134,217,728 parameters, the TSDFs are compressed to a smaller resolution of 64×32^3 using just the encoder of the autoencoder from Denninger et al. [14] as shown in Figure 4.4. This is the ground truth target that the network in this thesis tries to predict. Both the prediction and ground truth target can be decompressed to the original representation of size 512^3 using the decoder of the same autoencoder. Table 5.2 shows that the difference between the original TSDF volume and the reconstructed TSDF volume is minute as per the Mean Squared Error (MSE) (the lower, the better), which indicates that the autoencoder from Denninger et al. can be used for both compressing and decompressing, since it generalizes well to the generated TSDFs.

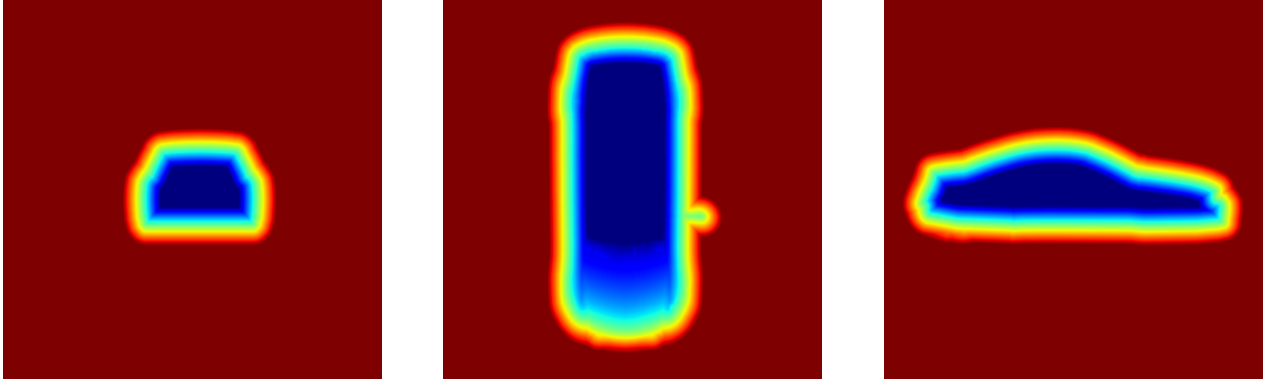


Figure 5.3.: Figure shows the different slices of the complete TSDF volume of a resolution 512^3 are visualized. Values range from -1 (dark blue) to 1 (dark red).

Table 5.2.: Table shows the Mean Squared Error (MSE) between the original 512^3 volume and the decompressed 512^3 volume using the autoencoder from Denninger et al. [14].

Object Category	MSE
airplane	2.892×10^{-6}
car	1.057×10^{-5}
chair	7.599×10^{-6}
table	5.846×10^{-6}
Overall	6.727×10^{-6}

5.3. Neural Network Training

5.3.1. Loss

Since the predictions of our approach are real valued TSDF volumes, regression-based loss functions are used. The most commonly used losses for regression-based reconstruction tasks are Mean Absolute Error (MAE) and Mean Squared Error (MSE) which are also used in our approach. They are explained below:

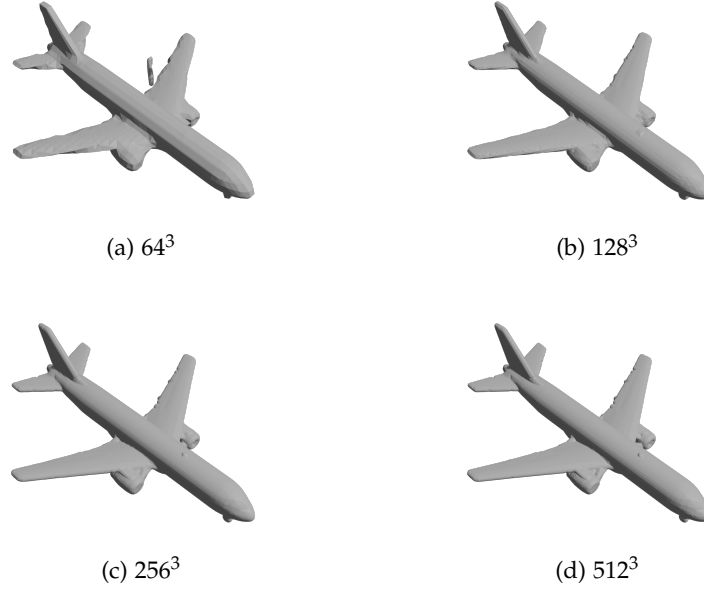


Figure 5.4.: TSDF meshes with different resolutions are visualized for the airplane category with *source id* 4561def0c651631122309ea5a3ab0f04. As the resolution increases, more details are captured. Note that the TSDFs have been converted into meshes using Marching Cubes [42] algorithm.

Mean Absolute Error (MAE) Loss

MAE is the average of the absolute difference between the ground truth 3D volume \mathbf{Y} and the predicted 3D volume $\hat{\mathbf{Y}}$ as shown in Equation 5.1. Here n refers to the total number of samples. In our approach, the loss is computed between the compressed ground truth 3D volume and the compressed predicted 3D volume as hinted in Chapter 4.

$$\mathcal{L}_{mae}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1}^n \|\mathbf{Y}_i - \hat{\mathbf{Y}}_i\|_1 \quad (5.1)$$

Mean Squared Error (MSE) Loss

MSE is the average of the squared difference between the ground truth 3D volume \mathbf{Y} and the predicted 3D volume $\hat{\mathbf{Y}}$ as shown in Equation 5.2. Here n refers to the total number of samples. In our approach, the loss is computed between the compressed ground truth 3D volume and the compressed predicted 3D volume as hinted in Chapter 4.

$$\mathcal{L}_{mse}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1}^n \|\mathbf{Y}_i - \hat{\mathbf{Y}}_i\|_2 \quad (5.2)$$

5.3.2. Evaluation

For evaluating the performance of the proposed network, two metrics, namely Intersection over Union (IoU) and F-Score@1% are used. The two evaluation metrics are explained below:

Intersection over Union (IoU)

The IoU between the ground truth 3D volume \mathbf{Y} and the predicted 3D volume $\hat{\mathbf{Y}}$ is calculated as shown in Equation 5.3. The numerator of Equation 5.3 calculates the intersection or the *logical and* between the thresholded ground truth 3D volume \mathbf{Y} and the thresholded predicted 3D volume $\hat{\mathbf{Y}}$ at zero. The denominator of the same equation calculates the union or the *logical or* between the thresholded ground truth 3D volume \mathbf{Y} and the thresholded predicted 3D volume $\hat{\mathbf{Y}}$ at zero. The IoU is computed between the original ground truth TSDF volume and the decompressed predicted TSDF volume as shown in Figure 4.1b.

$$IoU(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{\sum_{i,j,k} \mathbb{1}_{(\mathbf{Y}_{(i,j,k)} < 0)} \cdot \mathbb{1}_{(\hat{\mathbf{Y}}_{(i,j,k)} < 0)}}{\sum_{i,j,k} \mathbb{1}_{(\mathbf{Y}_{(i,j,k)} < 0)} + \mathbb{1}_{(\hat{\mathbf{Y}}_{(i,j,k)} < 0)}} \quad (5.3)$$

F-score@1%

The F-score introduced in Tatarchenko et al. [65] provides an additional measure for evaluating 3D reconstruction outputs. The F-score between the ground truth 3D volume \mathbf{Y} and the predicted 3D volume $\hat{\mathbf{Y}}$ is calculated as shown in Equation 5.4. $P(d)$ is the precision and $R(d)$ is the recall for a distance threshold d as defined in Equation 5.5 and Equation 5.6. For F-score@1%, $d = 0.01$. Here \mathcal{G} and \mathcal{R} represents the reconstructed point clouds from \mathbf{Y} and $\hat{\mathbf{Y}}$ respectively. $n_{\mathcal{G}}$ and $n_{\mathcal{R}}$ refer to the number of points in \mathcal{G} and \mathcal{R} respectively. The ground truth 3D TSDF volume \mathbf{Y} and the predicted 3D TSDF volume $\hat{\mathbf{Y}}$ are converted into a mesh representation using Marching Cubes after which 8,192 points are then sampled from both object surfaces for computing F-score. Higher F-score corresponds to better reconstruction outputs. Open3D [80] is used for implementing F-score. The F-score@1% is computed between the original ground truth TSDF volume and the decompressed predicted TSDF volume as shown in Figure 4.1b.

$$F\text{-score}(\mathbf{Y}, \hat{\mathbf{Y}}, d) = \frac{P(d) \cdot R(d)}{P(d) + R(d)} \quad (5.4)$$

$$P(d) = \frac{1}{n_{\mathcal{R}}} \sum_{r \in \mathcal{R}} (\min_{g \in \mathcal{G}} \|g - r\| < d) \quad (5.5)$$

$$R(d) = \frac{1}{n_{\mathcal{G}}} \sum_{g \in \mathcal{G}} (\min_{r \in \mathcal{R}} \|g - r\| < d) \quad (5.6)$$

5.4. Training Procedure

5.4.1. Train, Validation and Test Splits

Initially, a single network is trained for handling all views where the number of views were changed randomly in the inclusive range $[1, 24]$ after each epoch. However, with the number of views changing in each epoch, the training becomes unstable, which means the network does not converge to a minimum. Xie et al. trained their multiview 3D shape reconstruction networks, Pix2Vox [73] and Pix2Vox++ [74] using single-view images initially for 250 epochs without training their multi-scale context-aware fusion module. Afterwards, they trained the multiview networks for a further 100 epochs but this time varying the number of views randomly between 1 and 24. In

the interest of training time, four different networks were trained that each handle 1, 6, 12 and 24 number of views.

The train, validation and test data splits for the four networks are shown in Table 5.3. For each particular view, the number of RGB images as well the camera extrinsics change. However, the ground truth TSDF target for each of the networks remains the same.

Table 5.3.: Table shows the train, validation and test data splits which follow the same split distribution from 3D-R2N2 [6].

Views		Train	Validation	Test
RGB Images and Camera Extrinsics (Input)	1 view	30642	4371	8770
	6 views	183852	26226	56620
	12 views	367704	52452	105240
	24 views	735408	104904	210480
TSDF Volumes (Target)		30642	4371	8770

5.4.2. Processing

The RGBA images generated from BlenderProc have a resolution of $u \times u \times 3$ where $u = 256$. The generated images have focal lengths $f_x = 355.556$ and $f_y = 355.556$, and principal points $c_x = 127.5$ and $c_y = 127.5$. However, the pretrained ResNet layers used in the 2D network require the input RGBA images to have a resolution of $v \times v \times 3$ where $v = 224$, the RGBA images have to be scaled down. Alongside the RGBA images, the camera intrinsics parameters, which are used in the backprojection layer, also have to be rescaled in the same proportion. The camera matrix K is introduced in Equation 3.3 in Section 3.1.1. For scaling the camera matrix K to camera matrix $K_{resized}$, Equation 5.7 is used in the implementation where the scale factor $\frac{v}{u}$ adjusts the focal lengths f_x and f_y , and the scale factor $\frac{v-1}{u-1}$ adjusts the principal points c_x and c_y . After scaling, the focal lengths are $f_x = 311.112$ and $f_y = 311.112$, and principal points are $c_x = 111.5$ and $c_y = 111.5$. To recall, the camera intrinsic are used in setting up the backprojection layer as explained in Subsection 4.3.2.

$$K_{resized} = \begin{bmatrix} f_x * \frac{v}{u} & 0 & c_x * \frac{v-1}{u-1} \\ 0 & f_y * \frac{v}{u} & c_y * \frac{v-1}{u-1} \\ 0 & 0 & 1 \end{bmatrix} \quad (5.7)$$

5.4.3. Implementation Details

PyTorch [51] is used for implementing 2D and 3D networks. The backprojection layer is implemented using the PyTorch autograd following the implementation of Dai et al. [9]. For training, Adam [34] optimizer with a β_1 of 0.9 and a β_2 of 0.999. The learning rate of all four networks is initially set to 0.01 which is halved every 15 epoch. The loss function for training all the networks in MSE. The batch sizes of the 1, 6, 12 and 24 view networks are 24, 16, 11 and 7 respectively. No data augmentation is used in all of the networks because some of data augmentations like cropping changes the camera intrinsic parameters, and this is not accounted for during training. The network is trained using a fixed set of camera intrinsics. The camera intrinsics are not scaled initially.

Furthermore, each network is trained on a separate GPU with 24 GB VRAM. A SLURM [79] based cluster is used for scheduling and monitoring the jobs for these four networks.

6. Results

The network is trained and evaluated on the data split distribution from 3D-R2N2 [6], as already mentioned in Chapter 5. This is done so that the performance of our proposed approach can be compared with other methods in literature even though the input and output data used in our approach is different from those used in literature.

6.1. Quantitative

The quantitative results of the proposed approach are shown in the two tables below. Table 6.1 shows the quantitative result on the test set using IoU as an evaluation metric while Table 6.2 shows the result on the test set using the F-score@1% as an evaluation metric.

Table 6.1.: Table shows the result of our approach on the test set of ShapeNet following the test split distribution from 3D-R2N2. The metric used here is IoU.

Object Category	1 view	6 views	12 views	24 views
airplane	0.074	0.272	0.322	0.320
bench	0.015	0.049	0.068	0.067
cabinet	0.235	0.429	0.450	0.460
car	0.555	0.619	0.642	0.634
chair	0.067	0.185	0.190	0.213
display	0.056	0.155	0.175	0.143
lamp	0.017	0.073	0.086	0.092
speaker	0.236	0.417	0.423	0.382
rifle	0.002	0.065	0.081	0.066
sofa	0.252	0.453	0.479	0.491
table	0.022	0.093	0.132	0.147
telephone	0.205	0.408	0.414	0.410
watercraft	0.072	0.235	0.309	0.258
Overall	0.163	0.275	0.301	0.302

One thing to take away from Table 6.1 and Table 6.2 is that as the number of views increases, the performance of the proposed network also increases. However, when approaching a higher number of views, the performance change is not substantial. Another trend that is visible in the tables is that some ShapeNet categories performed better than the rest. For example, the car category performed the best in terms of the IoU and F-score@1% metrics in all the views. The reason for this could be that the car category is easier for our network to learn since car objects in ShapeNet are similar to each other with less variation. This also means that the networks are better able to associate the similar 2D features of the car with the 3D output, which helps in generating better

Table 6.2.: Table shows the result of our approach on the test set of ShapeNet following the test split distribution from 3D-R2N2. The metric used here is F-score@1%.

Object Category	1 view	6 views	12 views	24 views
airplane	0.083	0.250	0.297	0.299
bench	0.016	0.068	0.109	0.112
cabinet	0.085	0.172	0.187	0.190
car	0.341	0.434	0.485	0.475
chair	0.028	0.108	0.109	0.131
display	0.042	0.117	0.139	0.118
lamp	0.012	0.041	0.052	0.058
speaker	0.068	0.134	0.140	0.135
rifle	0.004	0.136	0.165	0.133
sofa	0.114	0.217	0.232	0.244
table	0.018	0.111	0.165	0.187
telephone	0.156	0.291	0.316	0.332
watercraft	0.073	0.231	0.267	0.073
Overall	0.097	0.195	0.227	0.224

reconstructions. Also, another reason could be that the car object contains fewer fine details as compared to other 3D objects like a lamp. The main component of the car object is the body which our network learned to reconstruct well, as shown in the Qualitative Section.

6.2. Qualitative

Table 6.3 and Table 6.4 present the qualitative results of our approach. The results show how the 1 view, 6 views, 12 views and 24 views networks perform for two different instances from the test set. The evaluation metrics, namely IoU and F-score@1% are mentioned under the reconstructed output.

Table 6.3 shows that our approach can reconstruct the car object sufficiently well. However, the reconstruction is missing some fine details like tires, side-view mirrors, and the spoiler of the car. Table 6.4 shows an example where our approach fails to reconstruct the cabinet object fully. However, the partial reconstruction relates well with the original target, especially in the 24 input views model. The reconstructions could be made better by using loss functions better suited for TSDF volumes like loss shaping from Denninger et al. [14].

6.3. Comparison to other approaches

There are no literature methods that directly compare to our approach since this thesis uses a newly generated dataset. However, most of the methods from literature are trained on the 3D-R2N2 dataset, which uses binary voxel grids of only a resolution of 32^3 . Our approach makes use of TSDF volume with a resolution of 512^3 . But, it is helpful to get insight about the performance of the other works in comparison to our approach. Table 6.5 lists down the metrics of the other

6. Results

Table 6.3.: Table shows the qualitative results for one particular test instance from the car category with *source id* 998f600899c76e4583653a771e25099b where our approach works well. The rows show the results of the 1 view, 6 views, 12 views and 24 views model.


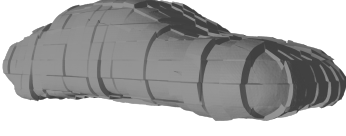
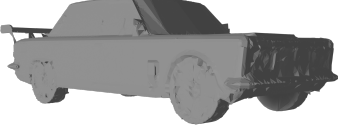

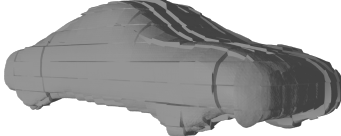
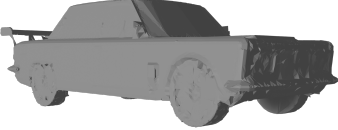

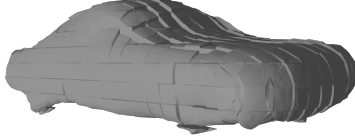
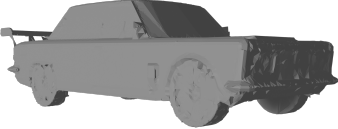

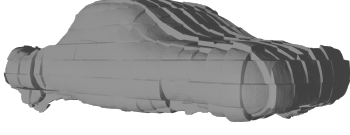
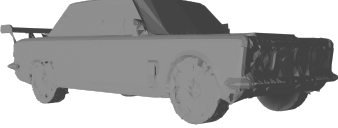
Images	Output	Target
		
IoU=0.791, F-score@1%=0.540		
		
IoU=0.811, F-score@1%=0.577		
		
IoU=0.836, F-score@1%=0.623		
		
IoU=0.799, F-score@1%=0.566		

Table 6.4.: Table shows the qualitative results for one particular test instance from the cabinet category with *source id* 91ac5c074c7d137762646c8cd54d58b4 where our approach does not perform well. The rows show the results of the 1 view, 6 views, 12 views and 24 views model.

Images	Output	Target
	 IoU=0.138, F-score@1%=0.039	
	 IoU=0.067, F-score@1%=0.043	
	 IoU=0.136, F-score@1%=0.058	
	 IoU=0.219, F-score@1%=0.118	

approaches and our approach. Our approach’s metrics are low compared to other approaches but this is expected since the output size of our approach is a volume of size 512^3 which is 4096 times more data to predict.

Table 6.5.: Table shows the evaluation metrics for different input views of the other works from literature. However, all the other works use the dataset from 3D-R2N2 so the results are not directly comparable.

Method		1 view	6 views	12 views	24 views
3D-R2N2 [6]	IoU	0.560	-	0.636	-
	F-score@1%	0.351	-	0.382	-
AttSets [77]	IoU	0.642	-	0.688	-
	F-score@1%	0.395	-	0.445	-
Pixel2Vox/A [73]	IoU	0.661	0.700	0.704	0.706
	F-score@1%	0.405	0.456	0.460	0.462
Pixel2Vox++/A [74]	IoU	0.670	-	0.717	0.720
	F-score@1%	0.436	-	0.460	0.473
Our method	IoU	0.163	0.275	0.301	0.302
	F-score@1%	0.097	0.195	0.227	0.224

6.4. Space Complexity

To test the space complexity, our method is compared against other prominent multiview 3D shape reconstruction methods. Table 6.6 shows the number of parameters in millions and memory usage in MB of the different multiview 3D shape reconstruction methods, which predict a reconstruction of resolution 32^3 . Our approach instead predicts a compressed representation of resolution 64×32^3 , which is decompressed using an autoencoder to generate reconstructions of resolution 512^3 . The proposed method has lower space complexity as compared to other methods except for the Pix2Vox++/A method. The memory requirement of the proposed multiview 3D shape reconstruction method is evaluated using the torchinfo tool from Yep et al. [78].

Since there is an additional decompressing step for generating reconstructions of resolution 512^3 using the autoencoder from Denninger et al., the proposed method has a higher time complexity as compared to the other multiview 3D shape reconstruction methods.

6.5. Compressed Output Visualization

Our network is trained using a compressed representation of size 64×32^3 . The compressed representation is generated from the original TSDF volume of size 512^3 . It is an interesting insight to see what the network has learned in the compressed representation. Figure 6.1 shows the three slices of the compressed output of different ShapeNet categories, namely aeroplane, bench, cabinet, car, and sofa. It can be seen in Figure 6.1 that compressed output contains the essential features that

Table 6.6.: Table compares the numbers of parameters and the memory requirement of different multiview 3D shape reconstruction methods. The data of the other methods comes from Xie et al. [74].

Method	Number of Parameters (million)	Memory (MB)
3D-R2N2 [6]	35.97	1407
AttSets [77]	17.71	3911
Pixel2Vox++/F [74]	4.83	647
Pixel2Vox++/A [74]	96.31	2411
Our method	14.75	4095

correlate with the final reconstruction output. Thus, the visualization confirms that the compressed representation that the network has learned is meaningful for multiview 3d shape reconstruction.

6.6. Changing Input Views

In our approach, four different networks are trained, which correspond to the 1, 6, 12 and 24 input views. The number of views remains fixed during training, unlike other approaches like Pix2Vox++ that change the number of views after each epoch in their multiview 3D shape reconstruction network. It is interesting to see how the trained networks perform if the number of views are changed for our approach. Table 6.7 shows the result of two such experiments.

In the first experiment, the input view is changed to 24 for the 1 view network while in the second experiment, the input view is changed to 1 for the 24 views network. Table 6.7 also contains the result from Table 6.1 and Table 6.2 of the original 1 view and 24 views network for comparison purposes. Both the modified networks perform worse when the input views are changed. The results show that the networks have a strong dependence on the original input views. One way to alleviate this problem could be to change the input views during training, similar to how it is done in Pix2Vox and Pix2Vox++.

Table 6.7.: Table shows the effect of testing the networks with different input views. The 1 view network is tested using 24 input views while the 24 views network is tested using 1 input view.

Original input views		1 view	24 views
New input views			
1 view	IoU	0.163	0.009
	F-score@1%	0.097	0.011
24 views	IoU	0.013	0.302
	F-score@1%	0.006	0.224

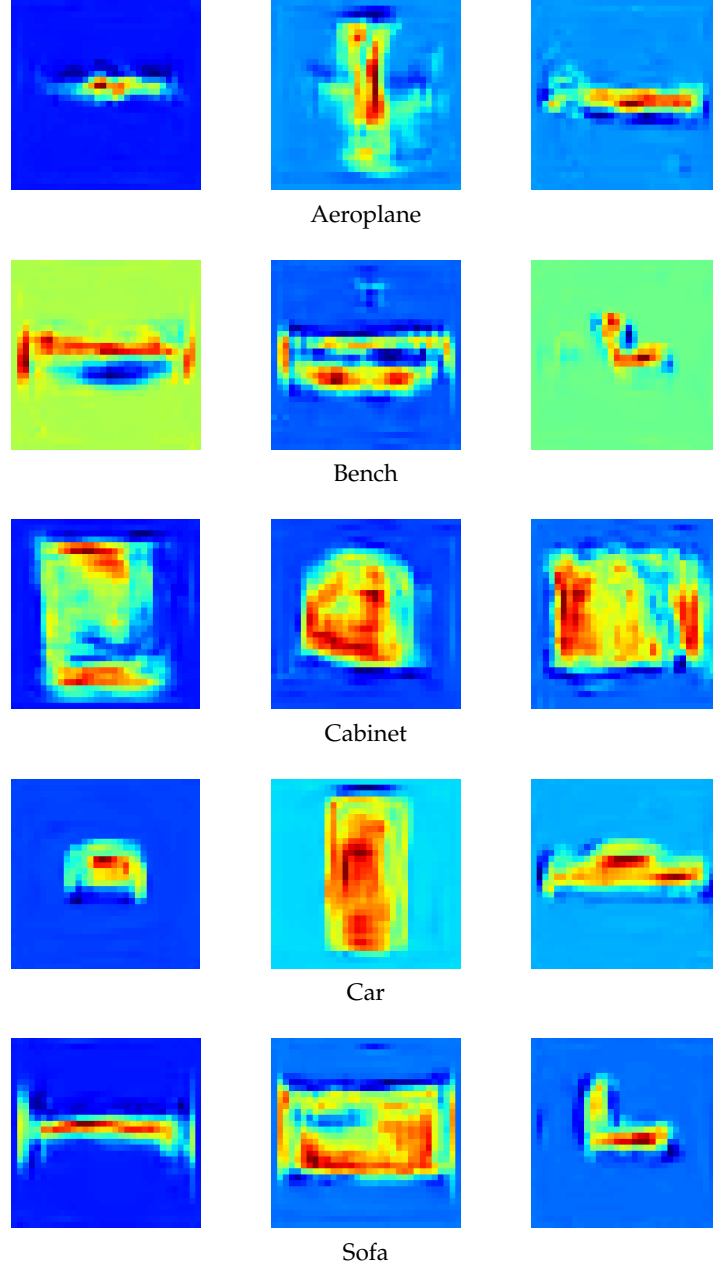


Figure 6.1.: Figures show the visualization of the three slices of the compressed output from different test data examples. The categories include aeroplane, bench, cabinet, car, and sofa. The compressed output has a size of 64×32^3 . The first channel is visualized only using the 24 views model.

7. Future Work

There are some further steps that can be explored as a possible future work building on top of our methods. Some exciting future work ideas are summarized below:

7.1. Problem Benchmark and Dataset

As of this writing, a good benchmark and dataset does not exist for evaluating multiview 3D shape reconstruction approaches. The most well known benchmark and dataset for the task of multiview 3D shape reconstruction comes from 3D-R2N2 [6]. However, the dataset only consists of a limited set of ShapeNet categories. Also, the RGB images and the binarized voxel grids are limited in resolution. Xie et al. [74] introduced a larger scale dataset with 1.68 million images of 280,000 objects named Things3D. However, since the Things3D dataset was based upon the SUNCG [59] dataset, it is not available anymore.

Our dataset is generated and follows the same data distribution splits of the 3D-R2N2 in order to have comparison on equal grounds. Nonetheless, a new large scale multiview 3D shape reconstruction dataset could be created with more ShapeNet categories using the tools developed in this thesis and BlenderProc [12], which would ultimately benefit the wider research community.

7.2. Uncertainty Estimation

Estimating uncertainty as an additional output of the network can be useful for some safety critical applications like autonomous driving since the neural networks tend to be overconfident in the predictions that they make as shown in [37]. Even though the problem of multiview 3D shape reconstruction has been studied extensively, there are not any known methods in literature that estimate uncertainty alongside the actual task of multiview 3D shape reconstruction. So as a potential future research direction, network architectural changes can be made so that uncertainty can be estimated as an additional output. Some methods that could be used as a part of the future work for estimating uncertainty include Monte Carlo Dropout [19] and Postels et al. [52].

7.3. Real World Transfer

Another future direction would be to explore how well the proposed approach transfers to the real world examples, since the data is synthetically generated. However, considerable effort is spent in ensuring the synthetic dataset as realistic as possible. There are datasets available in the literature that provide real world RGB images along with 3D original mesh object like Pix3D [63]. As a future work, it would also be insightful to see if the domain gap between Sim2Real transfer [27, 30] is low, which is difficult to achieve as previously mentioned in literature.

7.4. 3D Scene Datasets

Another possible future direction would be to try the proposed method on realistic looking 3D scene datasets like the SUNCG [59] and Replica [61] datasets. Since the proposed approach should also work on 3D scene datasets with some changes in ground truth 3D target, it would also be interesting to see how well the proposed method performs on 3D scene data. However, there might be difficulties in handling the 3D scene data. Possible problems that could be encountered maybe, for example, too many 3D objects in one scene or occlusion. Hence, it may be more challenging to get accurate reconstructions of 3D scenes.

7.5. Camera Intrinsics and Extrinsics

One of the limitations of the proposed approach is the reliance on the camera intrinsics and extrinsics parameters. For robotic applications, it might be easy to obtain such parameters since the system is already well calibrated, for example, DLR's Rollin' Justin [2]. However, for some other applications, it might be difficult to obtain these parameters. For future work, a new architecture can be proposed that does not require camera parameters as input. There are already works that do camera pose estimation alongside the task of 3D shape reconstruction like the work of Spezialetti et al. [60]. It is also important to note that methods that use camera parameters usually outperform methods that do not as explained in Xie et al. [74].

7.6. Adversarial Training

Our proposed network is trained end-to-end from input to output. There are, however, adversarial training strategies that can improve the network performance slightly, as reported in Roldao et al. [56]. Some works from literature claimed to get better performance when trained adversarially, including the work of Wang et al. [69], Chen et al. [5] and Wu et al. [71]. Following this trend, it would be worth trying adversarial training because it provides a better training supervision mechanism.

Figure 7.1 shows how neural networks can be trained adversarially. If adversarial training is to be used, the end-to-end proposed network can be used as a generator network while the discriminator network classifies the ground truth reconstructions from the output of the generator as being either real or fake. The goal for generator network is then to produce accurate reconstructions so that it can fool the discriminator, while the goal of the discriminator is to correctly classify whether the reconstruction comes from the generator or the ground truth reconstruction. Through this technique, both the generator as well as the discriminator networks improve in their tasks over time. Later on, the generator network alone can be used for generating reconstructions.

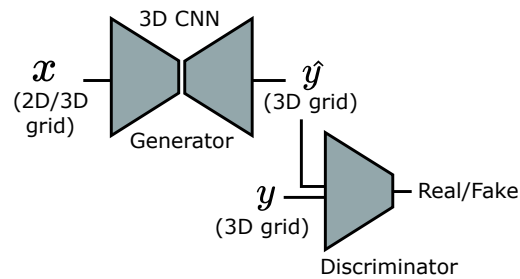


Figure 7.1.: Figure shows how neural networks can be trained adversarially which consists of a generator network whose task is to generate realistic looking samples and a discriminator network whose task is to determine if the samples comes from the generator network or ground truth data distribution. Similar to a min-max approach, both generator and discriminator networks improve over time. The image is taken from Roldao et al. [56].

8. Conclusion

In this thesis, a novel learning-based approach for multiview 3D shape reconstruction is explained. The proposed approach takes as input RGB images taken from multiple viewpoints and camera parameters, and it makes use of a 2D network, a backprojection layer, and a 3D network. Unlike existing methods in the literature, the proposed method directly associates the 2D feature maps with the 3D output. The method can reconstruct complex 3D objects with an output resolution of 512^3 , making it one of the few methods in the literature for 3D shape reconstruction in such a high resolution.

Alongside the proposed approach, the thesis also contributed with an improved dataset, based upon the ShapeNet [4] dataset, with 1,050,816 image renderings of 3D objects and 43,784 TSDF volumes, which corresponds to 24 different viewpoints following the split distribution from 3D-R2N2 [6]. The thesis also presents detailed quantitative and qualitative results on the newly introduced multiview 3D reconstruction dataset. The results conclude that the performance of our multiview reconstruction method increases up to a certain input views after which the performance stagnates.

A. BlenderProc Config

```
1  # Args: <path_to_shape-net-core> <output_dir>
2  version: 3
3  setup:
4      blender_install_path: /home_local/<env:USER>/blender/
5      pip:
6          - h5py
7  modules:
8      - module: main.Initializer
9        config:
10            global:
11                output_dir: <args:3>
12      - module: loader.ShapeNetLoader
13        config:
14            data_path: <args:0>
15            used_synset_id: <args:1>
16            used_source_id: <args:2>
17            move_object_origin: false
18      - module: lighting.LightLoader
19        config:
20            lights:
21                - type: POINT
22                  location: [0, 0, 5]
23                  energy: 1000
24                - type: POINT
25                  location: [0, 0, -5]
26                  energy: 1000
27                - type: POINT
28                  location: [5, 5, 0]
29                  energy: 1000
30                - type: POINT
31                  location: [-5, -5, 0]
32                  energy: 1000
33                - type: POINT
34                  location: [-5, 5, 0]
35                  energy: 1000
36                - type: POINT
37                  location: [5, -5, 0]
38                  energy: 1000
```

```
39 - module: camera.CameraSampler
40   config:
41     intrinsics:
42       resolution_x: 256
43       resolution_y: 256
44     cam_poses:
45       - number_of_samples: 24
46         location:
47           provider: sampler.Sphere
48           center: [0, 0, 0]
49           radius: 2.5
50           mode: SURFACE
51         rotation:
52           format: look_at
53           value:
54             provider: getter.POI
55 - module: renderer.RgbRenderer
56   config:
57     transparent_background: true
58     output_key: colors
59     use_alpha: false
60     samples: 350
61 - module: writer.ShapeNetWriter
62 - module: writer.CameraStateWriter
63   config:
64     attributes_to_write: [location, rotation_euler, fov_x, fov_y,
65                           shift_x, shift_y, cam_K, cam2world_matrix]
66 - module: writer.Hdf5Writer
67   config:
68     write_alpha_channel: true
```

List of Figures

2.1.	Figure shows the binary occupancy grid output of resolution 32^3 of the different multiview 3D shape reconstruction methods on the dataset introduced in 3D-R2N2 from Choy et al. [6]. The image is taken from Xie et al. [74].	5
3.1.	Figure shows the pinhole camera model with P_w in world coordinate system, P_c in camera coordinate and p in the 2D image plane. The image is taken from OpenCV Camera Calibration and 3D Reconstruction documentation [47].	8
3.2.	Figure shows how the 3D objects are projected in the 2D image plane using perspective projection in (a) and orthographic projection in (b). The images are taken from Jia et al. [31].	10
3.3.	Figure shows the perspective view frustum. The perspective view frustum takes the shape of a truncated pyramid. The image is taken from Lighthouse3d.com [41] website.	11
3.4.	Figure shows the orthographic view frustum. The orthographic view frustum takes the shape of a cuboid. The image is taken from Martin Kraus [36] and it is originally published on Wikipedia.	11
3.5.	Figure shows view frustum culling being applied to green, red, and yellow objects. Taking the shape of the view frustum into account, all the green objects are rendered, yellow objects are partially rendered and none of the red objects are rendered. The image is taken from Lighthouse3d.com [41] website.	12
3.6.	Figure shows a 2D TSDF with a red curve depicting the surface with all the values inside the surface being negative, on the surface being zero, and outside the surface being positive. As can be seen in the figure, a truncation between -1 and 1 is applied. The same concept can also be extended to 3D. The image is taken from Arm Community website [44].	13
3.7.	Figure shows the memory comparison between binary occupancy/voxel grids and TSDF volumes of different resolutions. As resolution increases, TSDF volumes take up more memory than binary occupancy grids. For example, a volume of size 512^3 corresponds to more than 134 million voxels. For storing this volume, a binary occupancy grid representation would take approximately 134.22 MB while a TSDF volume representation would take approximately 536.87 MB.	14
3.8.	Figure shows the most commonly used 3D data representations. Different methods use different 3D data representations. The images are taken from Wang et al. [68], Wang et al. [69], Song et al. [59], and Dai et al. [11].	15
3.9.	Figure shows all the normals in orange that are used for calculating the distance $d(p, T)$. The orthogonal planes are shown in blue and dashed. The original figure is taken from Denninger et al. [14] and adapted afterward.	16

3.10. Figure shows a simple residual block that makes use of a skip connection. The skip connection adds the output from the previous layer. The image is taken from the original ResNet [23] paper.	17
3.11. Figures show that both the training and test performance of the neural network suffers when the number of layers are increased from 20 to 56. ResNets alleviated this problem through the use of skip connections. The image is taken from the original ResNet [23] paper.	17
3.12. Figure shows the loss surfaces of ResNet-56 with and without skip connections. By adding skip connections, the loss surface becomes smoother leading to better convergence. The image is taken from Li et al. [39].	18
3.13. Figures show the two different types of residuals blocks that are used in ResNet. The standard residual block, which is used in ResNet-18 and ResNet-34 architectures, has two convolutional layers of filter size 3×3 while the bottleneck residual block, which is used in ResNet-50, ResNet-101, and ResNet-152 architectures, has three convolutional layers of sizes 1×1 , 3×3 and 1×1 where the 1×1 filters firstly increase and then decrease the channels while the 3×3 filter acts as a bottleneck. The rectangles represent convolutional layers. The filter size and number of channels are mentioned inside the rectangle. The image is taken from the original ResNet [23] paper.	19
3.14. Figure show an autoencoder with an input layer, a hidden layer and an output layer. An autoencoder is used extensively for compressing and decompressing data in an unsupervised way. The image is taken from Jeremy Jordan's blog [32].	19
4.1. Figure shows the network architectures for training in (a) and testing in (b). During the training phase only the 2D network, the backprojection layer, and the 3D network is used while only during the testing phase, an additional decoder network is used. The network in (a) is trained end-to-end using a single optimizer.	21
4.2. Figure shows the different layers in the 2D network. Initial layers of the 2D network make use of pretrained weights from ResNet-50. 7 bottleneck residual blocks are used from ResNet-50. The bottleneck residual block is depicted in the figure on the right. The 4 subsequent layers consist of regular convolutions that maintain the same spatial dimension. The 2D network and the figure is inspired from Xie et al. [74].	22
4.3. Figure shows the different layers that are a part of the 3D network. All layers consist of a convolutional layer, a BatchNorm layer, and a ReLU layer except for the last layer which does not use a ReLU layer. The layers maintain the same spatial resolution of size 32^3	24
4.4. Figure shows the autoencoder used to decompress the TSDFs into a latent representation of size 64×32^3 during the training phase and later to decompress in the original resolution of 512^3 during the testing phase. The image is taken from Denninger et al. [14].	24

5.1.	Figures show the normals orientation of the original ShapeNet object in (a) and the watertight version of the same object in (b). In the two figures, the color blue represents that the normal is pointing outwards from the surface while the color red represents that the normal is pointing inwards from the surface. As can be seen in figures, the watertight version has correct normals orientation which is essential for TSDF generation. The used object in both figures is from the car category with <i>source id</i> 1a1dcd236a1e6133860800e6696b8284. The normals are visualized using Blender [7].	26
5.2.	Figure shows the RGB renderings of the different ShapeNet categories, namely the airplane category with <i>source id</i> 4561def0c651631122309ea5a3ab0f04, car category with <i>source id</i> e4886a4d0c6ea960fe21694bd5f519d1, chair category with <i>source id</i> 1fccc2ac4bfd3da535836c728d324152, sofa category with <i>source id</i> 62e90a6ed511a1b2d291861d5bc3e7c8 and table category with <i>source id</i> 1011e1c9812b84d2a9ed7bb5b55809f8. All the renderings are generated using BlenderProc [12].	28
5.3.	Figure shows the different slices of the complete TSDF volume of a resolution 512^3 are visualized. Values range from -1 (dark blue) to 1 (dark red).	29
5.4.	TSDF meshes with different resolutions are visualized for the airplane category with <i>source id</i> 4561def0c651631122309ea5a3ab0f04. As the resolution increases, more details are captured. Note that the TSDFs have been converted into meshes using Marching Cubes [42] algorithm.	30
6.1.	Figures show the visualization of the three slices of the compressed output from different test data examples. The categories include aeroplane, bench, cabinet, car, and sofa. The compressed output has a size of 64×32^3 . The first channel is visualized only using the 24 views model.	40
7.1.	Figure shows how neural networks can be trained adversarially which consists of a generator network whose task is to generate realistic looking samples and a discriminator network whose task is to determine if the samples comes from the generator network or ground truth data distribution. Similar to a min-max approach, both generator and discriminator networks improve over time. The image is taken from Roldao et al. [56].	43

List of Tables

5.1.	Table shows the original number of 3D objects contained in each ShapeNet category. Each ShapeNet category is identified with a unique Synset Id.	25
5.2.	Table shows the Mean Squared Error (MSE) between the original 512^3 volume and the decompressed 512^3 volume using the autoencoder from Denninger et al. [14]. .	29
5.3.	Table shows the train, validation and test data splits which follow the same split distribution from 3D-R2N2 [6].	32
6.1.	Table shows the result of our approach on the test set of ShapeNet following the test split distribution from 3D-R2N2. The metric used here is IoU.	34
6.2.	Table shows the result of our approach on the test set of ShapeNet following the test split distribution from 3D-R2N2. The metric used here is F-score@1%.	35
6.3.	Table shows the qualitative results for one particular test instance from the car category with <i>source id</i> 998f600899c76e4583653a771e25099b where our approach works well. The rows show the results of the 1 view, 6 views, 12 views and 24 views model.	36
6.4.	Table shows the qualitative results for one particular test instance from the cabinet category with <i>source id</i> 91ac5c074c7d137762646c8cd54d58b4 where our approach does not perform well. The rows show the results of the 1 view, 6 views, 12 views and 24 views model.	37
6.5.	Table shows the evaluation metrics for different input views of the other works from literature. However, all the other works use the dataset from 3D-R2N2 so the results are not directly comparable.	38
6.6.	Table compares the numbers of parameters and the memory requirement of different multiview 3D shape reconstruction methods. The data of the other methods comes from Xie et al. [74].	39
6.7.	Table shows the effect of testing the networks with different input views. The 1 view network is tested using 24 input views while the 24 views network is tested using 1 input view.	39

Bibliography

- [1] A. Bærentzen and H. Aanæs. “Generating Signed Distance Fields From Triangle Meshes.” In: *Informatics and Mathematical Modelling (IMM) Technical Report* (2002).
- [2] O. Birbach, U. Frese, and B. Bäuml. “Rapid calibration of a multi-sensorial humanoid’s upper body: An automatic and self-contained approach.” In: *The International Journal of Robotics Research* (2015).
- [3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. “Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age.” In: *IEEE Transactions on Robotics* (2016).
- [4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.
- [5] Y.-T. Chen, M. Garbade, and J. Gall. “3D Semantic Scene Completion from a Single Depth Image Using Adversarial Training.” In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019.
- [6] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016.
- [7] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: <http://www.blender.org>.
- [8] B. Curless and M. Levoy. “A Volumetric Method for Building Complex Models from Range Images.” In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 1996.
- [9] A. Dai and M. Nießner. “3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.
- [10] A. Dai, C. R. Qi, and M. Nießner. “Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis.” In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.
- [11] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner. “ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans.” In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2018.
- [12] M. Denninger, M. Sundermeyer, D. Winkelbauer, D. Olefir, T. Hodan, Y. Zidan, M. Elbadrawy, M. Knauer, H. Katam, and A. Lodhi. “BlenderProc: Reducing the Reality Gap with Photorealistic Rendering.” In: *Robotics: Science and Systems (RSS)*. 2020.

- [13] M. Denninger, M. Sundermeyer, D. Winkelbauer, D. Olefir, T. Hodan, Y. Zidan, M. Elbadrawy, M. Knauer, H. Katam, and A. Lodhi. "BlenderProc: Reducing the Reality Gap with Photorealistic Rendering." In: *Robotics: Science and Systems (RSS)*. July 2020. URL: <https://elib.dlr.de/139317/>.
- [14] M. Denninger and R. Triebel. "3D Scene Reconstruction from a Single Viewport." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.
- [15] H. Fan, H. Su, and L. Guibas. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." In: *Computer Vision and Pattern Recognition*. 2017.
- [16] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
- [17] Y. Feng, Y. Feng, H. You, X. Zhao, and Y. Gao. "MeshNet: Mesh Neural Network for 3D Shape Representation." In: *AAAI 2019* (2018).
- [18] J. Fuentes-Pacheco, J. R. Ascencio, and J. M. Rendon-Mancha. "Visual simultaneous localization and mapping: a survey." In: *Artificial Intelligence Review* (2012).
- [19] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. 2016.
- [20] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Journal of Machine Learning Research - Proceedings Track* (2010).
- [21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [22] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. "MeshCNN: A Network with an Edge." In: *ACM Transactions on Graphics (TOG)* (2019).
- [23] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: "CVPR". 2015.
- [24] K. He, X. Zhang, S. Ren, and J. Sun. "Identity Mappings in Deep Residual Networks." In: *Computer Vision – ECCV 2016*. 2016.
- [25] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Comput.* (2006).
- [26] T. Hodaň, V. Vineet, R. Gal, E. Shalev, J. Hanzelka, T. Connell, P. Urbina, S. Sinha, and B. Guenter. "Photorealistic Image Synthesis for Object Instance Detection." In: *IEEE International Conference on Image Processing (ICIP)* (2019).
- [27] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, M. Mozifian, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. K. Liu, J. Peters, S. Song, P. Welinder, and M. White. *Perspectives on Sim2Real Transfer for Robotics: A Summary of the R:SS 2020 Workshop*. 2020.
- [28] J. Huang, Y. Zhou, and L. Guibas. "ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups." In: *arXiv preprint arXiv:2005.11621* (2020).
- [29] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

- [30] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. "Sim-To-Real via Sim-To-Sim: Data-Efficient Robotic Grasping via Randomized-To-Canonical Adaptation Networks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [31] J. Jia, J. Liu, G. Jin, and Y. Wang. "Fast and effective occlusion culling for 3D holographic displays by inverse orthographic projection with low angular sampling." In: *Applied optics* (2014).
- [32] J. Jordan. *Introduction to autoencoders*. 2018. URL: <https://www.jeremyjordan.me/autoencoders/> (visited on 10/08/2021).
- [33] A. Kar, C. Häne, and J. Malik. "Learning a Multi-View Stereo Machine." In: *NIPS*. 2017.
- [34] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.
- [35] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes." In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014.
- [36] M. Kraus. *File:Orthographic_view_frustum.png*. 2011. URL: <https://commons.wikimedia.org/w/index.php?curid=15389490> (visited on 10/07/2021).
- [37] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017.
- [38] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning." In: *Nature* (2015).
- [39] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. "Visualizing the loss landscape of neural nets." In: *Advances in Neural Information Processing Systems*. 2018.
- [40] Y. Liao, S. Donné, and A. Geiger. "Deep Marching Cubes: Learning Explicit Surface Representations." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [41] Lighthouse3d.com. *View Frustum's Shape*. URL: <http://www.lighthouse3d.com/tutorials/view-frustum-culling/view-frustums-shape/> (visited on 10/07/2021).
- [42] W. E. Lorensen and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. 1987.
- [43] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003.
- [44] R. L. Mendez. *The Rise of Depth on Mobile*. 2018. URL: <https://community.arm.com/developer/tools-software/graphics/b/blog/posts/the-rise-of-depth-on-mobile> (visited on 10/05/2021).
- [45] A. Munshi, D. Ginsburg, and D. Shreiner. *The OpenGL ES 2.0 programming guide*. Addison-Wesley, 2009.
- [46] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010.

- [47] OpenCV. *OpenCV Camera Calibration and 3D Reconstruction Documentation*. 2021. URL: <https://docs.opencv.org/4.5.3/d9/d0c/group%20%5Ctextunderscore%20calib3d.html> (visited on 10/06/2021).
- [48] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014.
- [49] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer. "A survey of structure from motion." In: *Acta Numerica* (2017).
- [50] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems* 32. 2019.
- [52] J. Postels, F. Ferroni, H. Coskun, N. Navab, and F. Tombari. "Sampling-Free Epistemic Uncertainty Estimation Using Approximated Variance Propagation." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.
- [53] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In: *CVPR* (2016).
- [54] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: (2017).
- [55] S. R. Richter and S. Roth. "Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers." In: *CVPR*. 2018.
- [56] L. Roldao, R. de Charette, and A. Verroust-Blondet. "3D Semantic Scene Completion: a Survey." In: *International Journal of Computer Vision*. 2021.
- [57] D. E. Rumelhart and J. L. McClelland. "Learning Internal Representations by Error Propagation." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987.
- [58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* (2015).
- [59] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic Scene Completion from a Single Depth Image." In: *CVPR* (2017).
- [60] R. Spezialetti, D. J. Tan, A. Tonioni, K. Tateno, and F. Tombari. "A Divide et Impera Approach for 3D Shape Reconstruction from Multiple Views." In: *2020 International Conference on 3D Vision (3DV)*. 2020.

- [61] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briaies, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe. "The Replica Dataset: A Digital Replica of Indoor Spaces." In: *arXiv preprint arXiv:1906.05797* (2019).
- [62] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. "Multi-view convolutional neural networks for 3d shape recognition." In: *Proc. ICCV*. 2015.
- [63] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman. "Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [64] M. Tatarchenko, A. Dosovitskiy, and T. Brox. "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs." In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.
- [65] M. Tatarchenko, S. R. Richter, R. Ranftl, Z. Li, V. Koltun, and T. Brox. "What Do Single-view 3D Reconstruction Networks Learn?" In: 2019.
- [66] D. Wang, X. Cui, X. Chen, Z. Zou, T. Shi, S. Salcudean, Z. J. Wang, and R. Ward. "Multi-view 3D Reconstruction with Transformer." In: 2021.
- [67] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images." In: *ECCV*. 2018.
- [68] W. Wang, R. Yu, Q. Huang, and U. Neumann. "SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation." In: *CVPR*. 2018.
- [69] Y. Wang, D. J. Tan, N. Navab, and F. Tombari. "ForkNet: Multi-branch Volumetric Semantic Completion from a Single Depth Image." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019.
- [70] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum. "Learning Shape Priors for Single-View 3D Completion And Reconstruction." In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*. 2018.
- [71] S.-C. Wu, K. Tateno, N. Navab, and F. Tombari. "SCFusion: Real-time Incremental Scene Reconstruction with Semantic Completion." In: *2020 International Conference on 3D Vision (3DV)*. 2020.
- [72] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3D ShapeNets: A Deep Representation for Volumetric Shapes." In: *Computer Vision and Pattern Recognition*. 2015.
- [73] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang. "Pix2Vox: Context-aware 3D Reconstruction from Single and Multi-view Images." In: *ICCV*. 2019.
- [74] H. Xie, H. Yao, S. Zhang, S. Zhou, and W. Sun. "Pix2Vox++: Multi-scale Context-aware 3D Object Reconstruction from Single and Multiple Images." In: *International Journal of Computer Vision (IJCV)* (2020).
- [75] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann. "DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction." In: *Advances in Neural Information Processing Systems* 32. 2019.

- [76] F. Yagubbayli, A. Tonioni, and F. Tombari. *LegoFormer: Transformers for Block-by-Block Multi-view 3D Reconstruction*. 2021. arXiv: 2106.12102 [cs.CV].
- [77] B. Yang, S. Wang, A. Markham, and N. Trigoni. "Robust Attentional Aggregation of Deep Feature Sets for Multi-view 3D Reconstruction." In: *IJCV*. 2019.
- [78] T. Yep and S. Chandel. *torchinfo*. <https://github.com/TylerYep/torchinfo>. Accessed 29 January 2014. 2018.
- [79] A. B. Yoo, M. A. Jette, and M. Grondona. "SLURM: Simple Linux Utility for Resource Management." In: *Job Scheduling Strategies for Parallel Processing*. 2003.
- [80] Q.-Y. Zhou, J. Park, and V. Koltun. "Open3D: A Modern Library for 3D Data Processing." In: *arXiv:1801.09847* (2018).