Christian Sormann, BSc

Scene Completion for Image-Based Reconstructions

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor

Ass. Prof. Dipl.-Ing. Dr.techn. Friedrich Fraundorfer
Institute of Computer Graphics and Vision

Advisor
Dipl.-Ing. Thomas Holzmann, BSc

Graz, February 2019

Abstract

Image-based 3D reconstruction algorithms are established procedures for creating realistic 3D geometry from real-world environments. However, real-world scene elements with poorly textured surfaces can lead to missing geometry in the 3D reconstruction. Thus, completing geometry in image-based 3D reconstructions in a consistent way has the potential to be applied in many different tasks, such as filling in the missing geometry in reconstructions created for augmented reality or virtual reality applications. Over the course of this thesis, we introduce a convolutional neural network based on a state of the art scene completion method. In addition to an incomplete reconstruction of the scene as input, the proposed architecture uses three dimensional semantic input generated from 2D semantically segmented images. The network utilizes the additionally provided semantic context to improve on the geometric completion. Furthermore, we make use of dilated convolutions, which have been proposed in recent years by researches in the machine learning field, in order to achieve better results without the need for inferring on multiple resolution levels. Experiments were also performed with an alternative encoding for the used input semantic information. We show results of the proposed method and evaluated it quantitatively and qualitatively on synthetic data and on real-world data originating from image-based reconstructions. We show that the proposed method improves the results in terms of geometric completion and semantic prediction on synthetic datasets and in terms of completion in real-world scenarios.

Keywords. scene completion, neural networks, computer vision, image-based 3D reconstruction

Kurzfassung

Algorithmen zur Erstellung von bildbasierten 3D Rekonstruktionen sind eine bereits sehr ausgereifte Methodik, um 3D Rekonstruktionen aus Echtweltumgebungen zu generieren. Allerdings stellen schlecht textuerierte Oberflächen noch immer ein Problem dar, welches zu Löchern in der Geometrie führen kann. Das bedeutet, dass die Vervollständigung von bildbasierten 3D Rekonstruktionen zahlreiche Anwendungsgebiete aufweist, wie zum Beispiel die Vervollständigung fehlender Geometrie in realistischen 3D Modellen für Augmented Reality oder Virtual Reality Anwendungen. In dieser Arbeit präsentieren wir ein neuronales Netz basierend auf einer aktuellen Methode zur Vervollständigung von 3D Szenen. Die präsentierte Architektur nutzt semantische 3D Informationen generiert aus semantisch segmentierten Bildern als Eingabeinformation, zusätzlich zur unvollständigen 3D Rekonstruktion, welche ebenfalls als Eingabeinformation genutzt wird. Die Architektur macht sich auch geweitete Faltungen zu nutzen, um einen größeren globalen Kontext der Szene einzufangen, ohne auf mehreren Auflösungen zu rechnen. Darüber hinaus, wurde auch mit einer alternativen Codierung für die semantische Eingabeinformation experimentiert. In der Arbeit wurde die Methode auf synthetischen Szenen und Echtweltumgebungen, sowohl quantitativ als auch qualitativ evaluiert. Es wird gezeigt, dass die Methode in synthetischen Szenen, sowohl die geometrische Vervollständigung verbessert, als auch die semantische Information der vervollständigten Szene besser schätzt. In Echtweltumgebungen kann gezeigt werden, dass sich die Vervollständigung verbessert.

Schlagwörter Szenen Vervollständigung, neuronale Netze, maschinelles Sehen, bildbasierte 3D Rekonstruktion

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Acknowledgments

I would like to thank my advisor Dipl.-Ing. Thomas Holzmann for his valuable guidance and feedback throughout the writing process of this thesis. Furthermore, I would like to thank Ass. Prof. Dipl.-Ing. Dr.techn. Friedrich Fraundorfer for providing valuable insights and giving me the opportunity to write this thesis at the Institute of Computer Graphics and Vision. I would also like to thank my parents for their support throughout my time studying at the university.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Motivation

The reconstruction of three dimensional geometry from real-world scenes has received a lot of focus from researches in recent years, due to its many use cases like virtual- and augmented reality applications. There are many techniques involving dedicated sensors capturing depth information, however it is also possible to use overlapping images captured on a camera to reconstruct the 3D geometry of the scene captured on the imagery. This process is known as Structure from Motion (*SFM*) and it estimates the camera poses along with a sparse representation of the geometry. In order to refine this result, it is possible to create a dense point cloud or subsequently a mesh by integrating dense depth maps acquired through a Multi-View Stereo (*MVS*) system. Over the years, the methods utilized in *SFM* such as pose estimation and feature detection have evolved significantly, however poorly textured surfaces and occlusions are still challenging problems.

On the other hand, one of the main advantages of image-based 3D reconstructions, when compared to highly accurate laser scans, is that the input data is easily acquired without the need of expensive equipment such as light detection and ranging (*LIDAR*) scanners. Using images for reconstructing scenery also means that one is able use a large repository of already existing data, such as images captured by a large magnitude of other users on the Internet [24]. However, using images for densely reconstructing entire scenes is prone to create models which are incomplete, in the sense that some of the geometry has only been partially reconstructed. This can be the result of certain parts of

the scene obstructing the cameras view. Furthermore dense multi-view stereo algorithms do not produce perfect depth-maps, instead they are incomplete and can contain noise. This problem is more severe when trying to reconstruct poorly textured surfaces. Using active depth sensors for generating depth-maps can also lead to noisy results, as reflective surfaces and infrared interference caused by sunlight in outdoor environments disturb the measurement process.

Sometimes complete and visually consistent reconstructions are demanded, while accuracy might not be the highest priority. Examples for this use case would be virtual reality and augmented reality applications, where the experience of the user can be disturbed when geometry is missing from the displayed models. The possibility to create complete models from real-world image-based reconstructions would make it easier to create user generated content for virtual and augmented reality applications, as well as computer games incorporating 3D graphics. Another application would be generating complete and appealing models from indoor environments of houses or apartments, in order to give potential buyers a more interactive way of looking at the room architecture. Furthermore, users would have the possibility to create image-based 3D reconstructions of their homes to be used in virtual reality chat applications, such that a realistic representation of their environment is shown.

In order to create more complete 3D reconstructions it is necessary to either introduce additional information to the algorithm during the reconstruction process such as geometric priors, or to improve the geometry generated during the reconstruction via additional post processing steps. In recent years, researchers have published several methods with the goal of inferring missing geometry from given incomplete 3D models. For completing objects from sparse measurements traditional approaches have been using a database lookup mechanism, which replaced incomplete models entirely with models from a predefined database [32]. However, this cannot be used for completing general scenes as the performance is limited by the amount of objects contained in the database. With the increasing popularity of neural network based end-to-end learning there have been several methods which leverage the inference capabilities of neural networks [13]. These approaches are promising, as detecting patterns in existing data and inferring hypotheses for missing data is a problem where neural networks have shown to achieve good results, for instance in the field of 2D image in-painting [3]. In this regard, the problem of inferring missing geometry in 3D can be seen as adding another dimension to extend the 2D image in-painting problem.

Recent publications using neural networks have shown great promise for achieving results which are more complete and visually pleasing than geometry created by traditional approaches [47]. Furthermore providing the networks with additional information such as semantic data to provide more context, has also shown to improve performance during inference [18]. To this end, we propose a method based on a recently published neural network architecture for completing large reconstructed scenes called ScanComplete [13], where we provide additional semantic information and evaluate further network changes, in order to improve performance during inference.

1.2 Contributions

In this thesis, we want to improve the completion of missing geometry in image-based 3D reconstructions of indoor environments. The input data from these reconstructions contains missing geometry as a result of poorly textured scene elements or occlusions. In order to achieve a better completion rate on image-based reconstructions of indoor environments, we extend the ScanComplete [13] *CNN* architecture for scene completion to use an additional input channel containing semantic information. This modification should lead to an improvement with regards to the completion performance of the network, as this data provides additional context for the network. When inferring missing geometry the network can now make use of semantic data to decide on how to best complete a given scene component. For instance, if the semantic class surrounding missing geometry is wall, the network can use this context to fill in the missing geometry with a planar surface. Furthermore we modify the network architecture in several other ways and quantify performance changes in our experiments. These changes are:

- dilated convolutions within a single network-hierarchy
- different encoding for the semantic information
- flipped truncated signed distance function

Moreover, we have also created a framework for creating training data, which generates 3D semantic input volumes. This is done by utilizing semantic labels from synthetic ground truth images, in order to create 3D semantic volumes. For evaluating performance we test on both synthetic and real-world scenes generated from image-based reconstructions. Creating semantic labels for real-world scenes is done using the DeepLabv3 network architecture [7] (with parameters/weights from an already trained network using ade20k [62] [63]), which infers semantic labels from RGB images. To conduct our real-world data evaluation, we make use of the ETH 3D multi-view high-resolution dense reconstruction benchmark [45], which provides ground truth data to measure quantitative changes in the reconstructed result and subsequently completion performance. For evaluating performance on synthetic scenes we use the SUNCG dataset [47].

1.3 Outline

In this work, we will first elaborate on the core concepts utilized in this thesis over the course of the theoretical background chapter. Initially we will introduce the fundamentals of multi-view geometry followed by elaborations on the dense 3D reconstruction process. Furthermore we will describe the concepts used in standard neural networks as well as convolutional neural networks. In the next chapter, we present related works in the field of 3D scene and shape completion. We will provide an overview for both traditional and machine-learning based approaches and describe prominent methods in more detail. In the methodology chapter we discuss the used convolutional neural network architecture and training procedure. Finally, we will present our training data generation strategy and our obtained results in the experiments chapter and infer conclusions using the obtained results.

2 Theoretical Background

Contents

In this chapter, we introduce various techniques and methodologies, which serve as the basis for the components of the system proposed in this thesis. We will start with an overview of fundamentals of multi-view geometry in the first section. Next we will elaborate on the process of dense 3D reconstruction. The last two sections in this chapter focus on deep neural networks and convolutional neural networks.

2.1 Fundamentals of Multi-View Geometry

In this section we will discuss the fundamental geometric principles [22] used in order to compute 3D reconstructions from images. This includes the pinhole camera model, the parameterization of this model and computer vision principles such as epipolar geometry and triangulation. The section is concluded with a description of the Structure from Motion (*SFM*) technique.

2.1.1 Pinhole Camera Model

The pinhole camera model is used to describe the basic principles behind projecting a given scene into the viewpoint of a camera. It is modeled after the concept of the camera obscura. The camera obscura describes the process of directing incoming light rays with a small opening, sometimes referred to as a pinhole, onto a plane of light-sensitive material

behind it. This process will create a mirrored projection of the object in front of the pinhole on the light-sensitive material. For the purpose of simplifying the mathematical model, the plane of light-sensitive material, also called the image plane, is positioned in front of the pinhole. However, this mathematical model is equivalent to the physical model in terms of its geometric properties.

The mathematical model describes the process of intersecting a light-ray generated by point X in world coordinates, with the principal point behind the image plane of the camera and subsequently intersecting at a point x on the image plane. The focal length f describes the z-component of normal-distance of the principal point from the image plane. Typically the principal point is positioned in the origin of the camera coordinate system, thus the x- and y-components are zero. The parameter D describes the z-component of the normal-distance of the world point from the principal point. In this setting, one can use the concept of similar triangles to formulate a relationship between the y-coordinate components of X, D and x, which allows for the calculation of x_y given X_y:

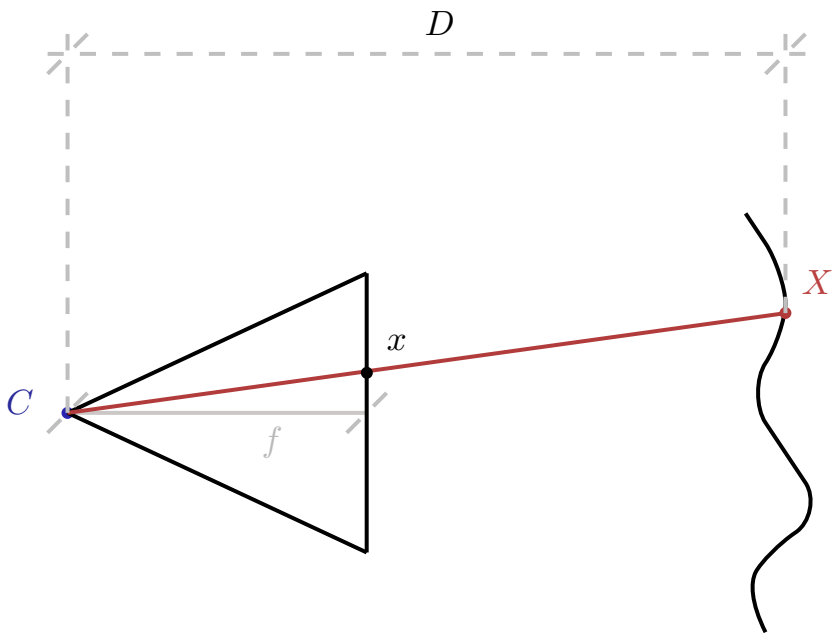


Figure 2.1: Graphical representation of the pinhole camera model: The world point X is projected onto the point x on the image plane through the principal point C. The normal distance from the principal point to the image plane is denoted as the focal length f, while the normal-distance of the principal point and the world point is represented by D.

$$\begin{aligned} \frac{x_y}{f} &= \frac{X_y}{D} \\ x_y &= f \cdot \frac{X_y}{D} \end{aligned} \tag{2.1}$$

This can also be done in an analogous way for the x-coordinate component:

$$\begin{aligned} \frac{x_x}{f} &= \frac{X_x}{D} \\ x_x &= f \cdot \frac{X_x}{D} \end{aligned} \tag{2.2}$$

2.1.2 Camera Calibration Matrix

One can also convert the coordinates on the image plane derived above, which are typically measured in millimeters to coordinates measured in pixels using scaling factors s_x and s_y and multiply these factors with the focal length:

$$\begin{aligned} x_y\,[px] &= \underbrace{s_y \cdot f}_{f_y} \cdot \frac{X_y}{D} \\ x_x\,[px] &= \underbrace{s_x \cdot f}_{f_x} \cdot \frac{X_x}{D} \end{aligned} \tag{2.3}$$

The scaling factors can be computed using the resolution in x-and y-direction of the camera and dividing it by the respective dimensions of the camera sensor in millimeters. However, the reference point for image coordinates using this model would be the point of intersection between the optical axis and the image plane. In order to move this reference point to the top-left of the image for the calculated coordinates, one can add the intersection point of the optical axis with the image plane $c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$ measured in pixel coordinates to the coordinates. Following the previous explanation, in order to retrieve the pixel coordinates of projected world points the following relation is used:

$$\begin{aligned} x_y\,[px] &= f_y \cdot \frac{X_y}{D} + c_y \\ x_x\,[px] &= f_x \cdot \frac{X_x}{D} + c_x \end{aligned} \tag{2.4}$$

The parameters $f = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$ and $c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$ are referred to as the intrinsic camera parameters. This can also be formulated as a matrix multiplication, in this case the 3×3 camera calibration matrix K will be constructed from the aforementioned intrinsic parameters and multiplied with the world-coordinate vector:

$$x = K \cdot X = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_x \\ X_y \\ D \end{bmatrix} \tag{2.5}$$

Note that this multiplication yields the following result:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_x \\ X_y \\ D \end{bmatrix} = \begin{bmatrix} f_x \cdot X_x + c_x \cdot D \\ f_y \cdot X_y + c_y \cdot D \\ D \end{bmatrix} \tag{2.6}$$

Because it is assumed that all of the coordinates on the image plane are located on the z-coordinate component $X_z = 1$, one can convert the resulting coordinates to coordinates on the image plane by dividing all coordinates with the z-component. The result of this division is again the original relation, which has been derived above:

$$\begin{aligned} (f_y \cdot X_y + c_y \cdot D) \cdot \frac{1}{D} &= f_y \cdot \frac{X_y}{D} + c_y \\ (f_x \cdot X_x + c_x \cdot D) \cdot \frac{1}{D} &= f_x \cdot \frac{X_x}{D} + c_x \end{aligned} \tag{2.7}$$

This is possible because when working in projective space, equivalence up to scale is attained when multiplying all coordinate components with the same scalar factor.

2.1.3 Extrinsic Camera Parameters

When calculating pixel coordinates on the image plane, the relationship derived in the previous section assumes that the camera and subsequently the image plane were not rotated or translated with respect to the world coordinate system. However, it would be advantageous to construct a relation that allows for position and orientation changes of the camera in 3D space as well. These changes are formulated as 3×1 translation vectors T and 3×3 rotation matrices R. In order to model these transformations of the camera, one has to first apply the translation and rotation to the 3D world point, so that its coordinates are transformed from the world coordinate system to the local camera coordinate system and then project the point into the view of the camera, as explained in the previous section. This relation is also shown in Figure 2.2. Applying these transformations to the 3D point X can be formulated as a matrix multiplication as well:

$$X_t = R \cdot X + t = \begin{bmatrix} R & t \end{bmatrix} X \tag{2.8}$$

In the equation above $\begin{bmatrix} R & t \end{bmatrix}$ is the column-wise concatenation of the rotation matrix R and the translation vector t. This transformation can be inserted into the original projection equation to account for changes in camera position and orientation:

$$x = \underbrace{K \begin{bmatrix} R & t \end{bmatrix}}_{P} X = PX \tag{2.9}$$

The combination of the camera calibration matrix and the matrix describing the extrinsic parameters is referred to as the projection matrix P. The extrinsic parameters can also be denoted as the camera pose.

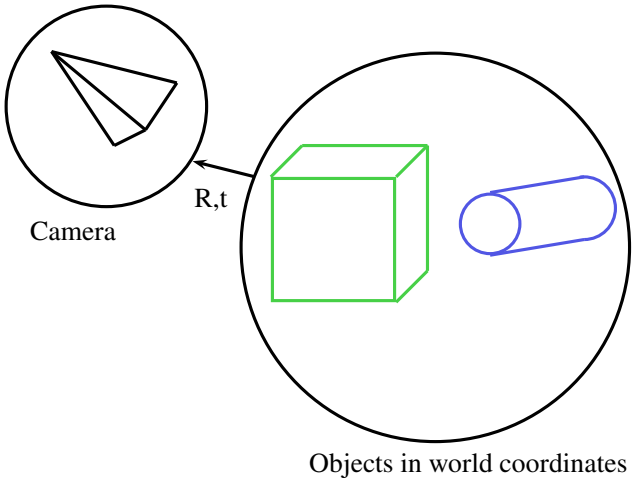


Figure 2.2: Coordinates in the world coordinate system are transformed into coordinates within the local camera coordinate system via rotation R and translation t.

2.1.4 Camera Calibration

The aforementioned intrinsic calibration parameters of a camera can be obtained by solving an optimization problem which uses 2D-3D point correspondences to minimize the re-projection error using a given set of camera parameters [60]. As shown in Figure 2.3, the 2D-3D point correspondences can be acquired by detecting points on a checkerboard pattern, where the dimensions in world coordinates are known. Furthermore it is important that the pattern is printed on a planar surface, as it is assumed that all detected points lie on the same plane. Using a checkerboard pattern, corners can easily be detected and used for generating the 2D-3D correspondences. The 3D correspondences are computed by using a point on the checkerboard pattern as a reference point for the world coordinate system and then generating the coordinates of all other world points in reference to this point. The other points can be generated as it is known that the surface is planar and thus

the points all have the same z-coordinate component. Using these assumptions, matches from the 2D images can be created by first matching the 3D reference point with a 2D correspondence and then subsequently all other points. Various images capturing the pattern from different viewpoints are then acquired, each generating multiple 2D-3D point correspondences. In the following optimization C correspondence pairs $(m_{i,c}, M_c)$ are used, where $m_{i,c}$ represents a 2D coordinate in image i and M_c represents the corresponding 3D point. With these C correspondences found in I images, the following optimization problem is then minimized:

$$\min_{K,R_i,t_i} \sum_{i=0}^{I} \sum_{c=0}^{C} \left|\left| m_{i,c} - K \begin{bmatrix} R_i & t_i \end{bmatrix} M_c \right|\right|^2 \tag{2.10}$$

The formulation above aims to minimize the re-projection error by modifying the intrinsic calibration matrix K, which is identical for all images and the extrinsic parameters R_i, t_i, which are different for varying captured viewpoints. This means that one must optimize a 3×3 matrix K representing the camera calibration, 3×3 rotation matrices $R_i,\ \forall i \in \{1, ..., I\}$ and 3×1 translation vectors $t_i,\ \forall i \in \{1, ..., I\}$ representing the camera poses for I images. Furthermore radial and tangential lens distortion parameters can additionally be estimated in a more complex procedure.

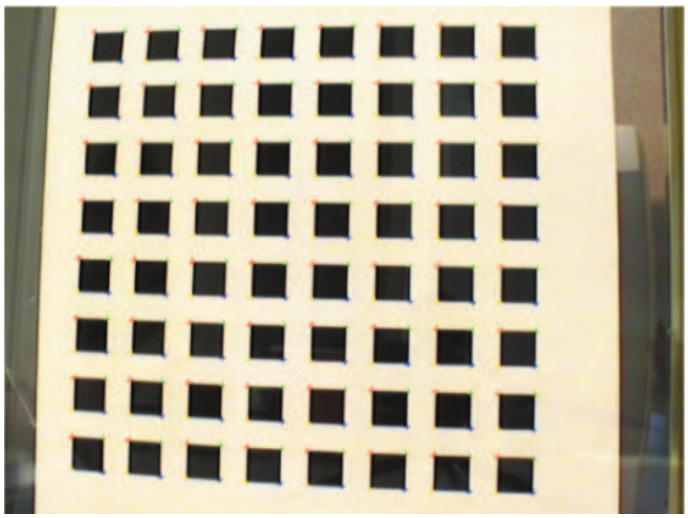


Figure 2.3: Visualization of 2D correspondences detected on a checkerboard calibration pattern. Imagery taken from [60].

2.1.5 Image Features

Corresponding point sets in two different images are used for many geometric algorithms and thus there exist many Computer Vision algorithms performing these correspondence

searches. In order to achieve this task, one needs to first find unique points, also referred to as salient points in each individual image and then find corresponding points by comparing them using a description of the locations. An algorithm, which detects salient points in an image is referred to as a feature detector, while the description of the keypoint is known as a feature descriptor or descriptor vector. There are many established feature detectors, which also define a way on how to compute the respective descriptors such as Scale Invariant Feature Transform (*SIFT*) [34] and Speeded Up Robust Features (*SURF*) [2]. However there are also dedicated feature detectors such as Features from Accelerated Segment Test (*FAST*) [42] and dedicated feature descriptors such as Binary Robust Independent Elementary Features (*BRIEF*) [6].

2.1.6 Epipolar Geometry

The search for correspondences in two neighboring images can be simplified by taking into account the geometric constraints defined by the epipolar geometry. A valid point correspondence between two image coordinates can be generated by a 3D world point seen in both images. This world point and the two principle points of the cameras form a plane Π_e denoted as the epipolar plane. Consequently, this plane is differently parameterized for different 3D points observed by the two cameras. The intersection of this plane with the two image planes forms the respective epipolar lines in both images. The intersection of the line formed by the two principle points with the respective camera planes is denoted as the epipoles e. The relationship between Π_e and e is also visualized in Figure 2.4. As explained above, different 3D world points yield epipolar planes with differing parameters and thus also different epipolar lines. However all of these epipolar lines intersect with the epipole. The epipolar lines allow for a simplified correspondence search, because the matching image location for an image point created from a given 3D world point in an image has to lie on the epipolar line created by this world point in the other image. This relation is encoded by the relative translation and rotation between the camera poses and can be represented via the fundamental matrix F. For image coordinate l in the left image and r in the right image the fundamental matrix fulfills the following constraint:

$$lFr = 0 \tag{2.11}$$

2.1.7 Triangulation

Once point correspondences have been found in two individual images and the calibrated cameras extrinsic parameters have been estimated through fundamental matrix estimation, it is possible to generate 3D geometry from point correspondences through a process called triangulation. The main principle of triangulation is also depicted in Figure 2.5. During triangulation, a linear equation system is formed from two point correspondences

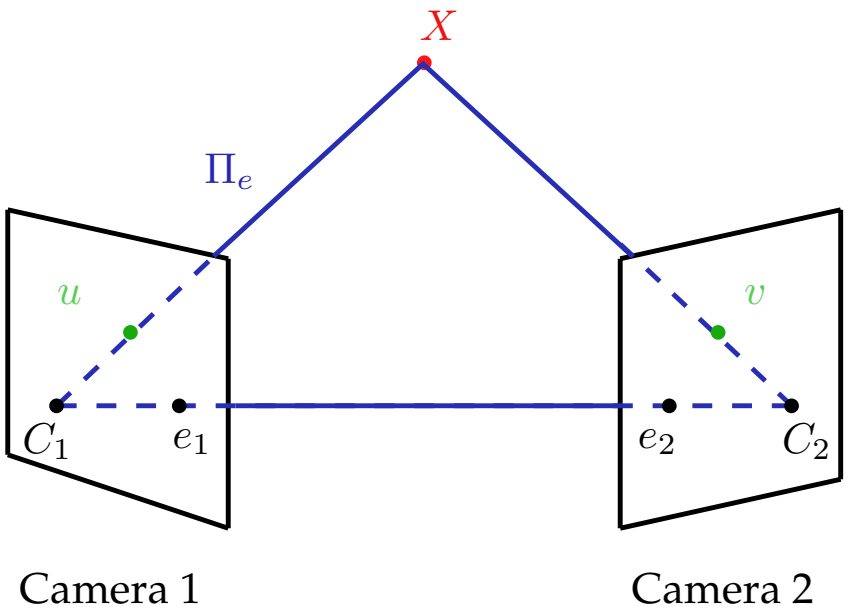


Figure 2.4: Visualization of the epipolar plane Π_e between two images spanned by the world coordinate X and its projections u and v into camera 1 and camera 2 respectively. The intersection of the line spanned by the principal points C_1 and C_2 of the two cameras with the image planes of the cameras is denoted as the epipoles e_1 and e_2.

$u = P_1X$ and $v = P_2X$ with the goal of recovering world point X. In this case u, v are 2D image coordinates projected from the same 3D point X using the respective projection matrices P_1 and P_2 of the two cameras. The linear equation system can be formed by constraining the problem with regards to the viewing rays generated by projecting X into the camera. After solving this problem, the world-point X is recovered from two camera correspondences.

2.1.8 Structure from Motion

Structuring from Motion is a technique which uses the aforementioned point correspondences and triangulation to generate a sparse 3D representation of a scene from images captured on a calibrated camera. After setting the pose of the first camera to be at the world coordinate origin (no rotation and translation), the pose of the second camera to be added to the global structure can be computed via fundamental matrix estimation [36]. Given these two poses and the intrinsic calibration of the camera recovered beforehand, one can triangulate 3D geometry by using point correspondences generated by feature matching executed on the two images. Further camera poses can be estimated using existing 3D points and newly generated 2D feature matches via an algorithm to solve the perspective n-point problem [31]. Using the new pose and feature matches, one can then generate additional 3D points. This process is repeated for every image to be added to

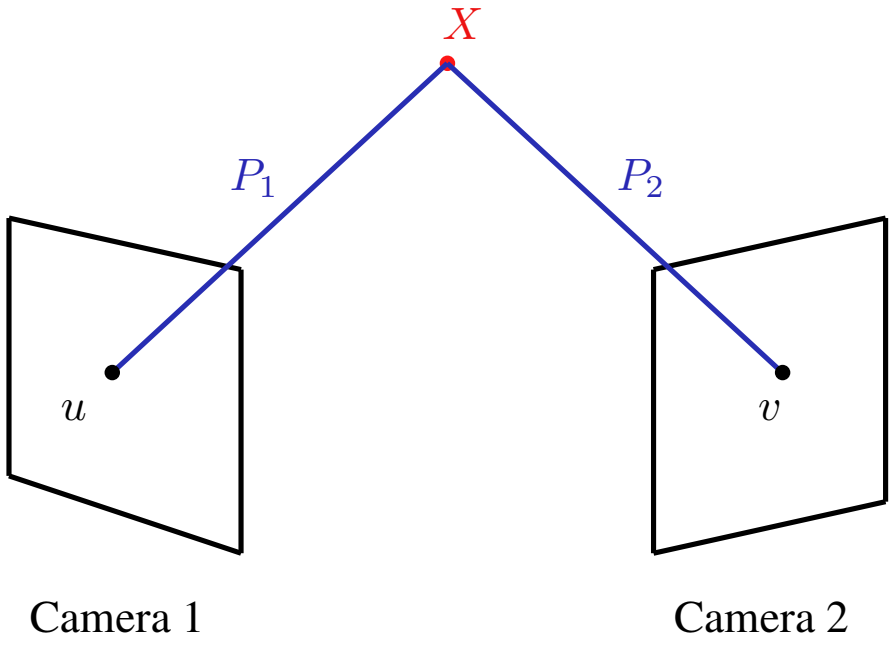


Figure 2.5: The world point X is projected onto the 2D image points u and v via the respective projection matrices P_1 and P_2.

the structure. After adding new camera views an optimization procedure called bundle adjustment [53] estimates corrected camera poses and 3D geometry by minimizing the re-projection error for all images in the global structure.

2.2 Dense 3D Reconstruction

Over the course of this section we will talk about the fundamental concepts involved in dense 3D reconstruction techniques.

2.2.1 Dense Matching

In order to create a dense point cloud as an output, dense reconstruction pipelines compute dense depth maps from input images, which are then combined to create a dense 3D representation of the reconstructed geometry. The process of creating these dense depth maps from input images is called dense stereo matching. There are techniques that compute dense depth maps using two images which is typically denoted by two view stereo. Approaches which generate depth maps using multiple input images including their respective camera poses are called Multi-View Stereo (*MVS*) techniques. During stereo matching, one wants to find image locations within either two images for two view stereo, or within multiple images in the *MVS* case, which are projections of the same 3D point. In order to find correspondences within 2D image points a cost measure is computed which describes

how likely the two points resulted from the same 3D projection. These cost measures are described in more detail in the next section. There are also methods to further refine this solution, such as the belief propagation algorithm [50].

In order to simplify the matching process the images in the two view stereo case are rectified such that their epipolar lines are parallel to the image x-axis, which is denoted as the stereo normal case. Once two matching locations have been found, the depth component Z in the stereo normal case for two-view stereo, with baseline B and focal length f, can be found from the difference in the x coordinate component of the correspondences, denoted as disparity D, as follows:

$$\begin{aligned} \frac{Z}{B} &= \frac{f}{D} \\ Z &= B \cdot \frac{f}{D} \end{aligned} \tag{2.12}$$

It can be seen here that depth and coordinate difference are in an inverse relationship. Thus a higher disparity value means that the resulting depth will be lower and vice versa. For the multi-view case one can either extend the stereo normal case using additional cameras with varying baselines [37] or use an extension of the disparity to depth relation derived by searching along the epipolar lines of multiple views [49]. We will also describe additional methods in the techniques section below.

2.2.2 Cost Measures

A core concept of dense matching techniques are matching costs. Matching costs define a similarity measure between two locations in different images, where a higher cost value means less similarity and a lower cost value implies higher similarity. These costs usually take into account a window around the two given locations to compute the similarity. The way the costs are then computed, depends on the respective cost measure i.e. sum of squared distances (SSD) at location (y, x) between image L and image R using a $W \times W$ window around location (y, x) is computed as follows:

$$SSD(y,x) = \sum_{i,j \,\in\, W \times W} (L(i,j) - R(i,j))^2 \tag{2.13}$$

Another measure is the normalized cross correlation (NCC):

$$NCC(y,x) = \frac{\sum_{i,j \,\in\, W \times W} L(i,j)}{\sqrt{\sum_{i,j \,\in\, W \times W} L(i,j)^2}} \cdot \frac{\sum_{i,j \,\in\, W \times W} R(i,j)}{\sqrt{\sum_{i,j \,\in\, W \times W} R(i,j)^2}} \tag{2.14}$$

2.2.3 Techniques

The basic principle behind both two-view and multi view dense stereo matching algorithms is to constrain the amount of locations for which the matching costs have to be computed. In the case of two-view stereo one utilizes the fact that in the stereo normal case all of the possible matches for location (x, y) in one image have the same y component in the other image, because they lie on the epipolar line in the corresponding image, which is parallel to the y axis for the stereo normal case. However, it is necessary to rectify the images beforehand. It is then possible to compute the costs for each location in an image by searching along the same y coordinate in the other image in the direction of the disparity.

For *MVS* it is also possible to restrict the amount of locations for which to compute matches. If one wants to compute a dense depth map for a reference camera incorporating additional camera poses viewing the same scene, it is possible to use multiple three dimensional planes [17] created from the reference camera, to restrict possible matches in other cameras. To achieve this, points on the plane created in the view of the reference camera are projected onto the image planes of the other cameras by using their respective pose and calibration matrices. It is then possible to use the plane yielding the best costs between a pixel in the reference view and all other views in order to compute the depth of this pixel in the reference view. Another approach utilizes properties of local image patches for *MVS* [4]. Although these algorithms work better than two-view approaches due to an increase of viewpoint information, they also do not generate complete results in every scenario as seen in Figure 2.6.

2.2.4 Fusion of Depth Data

In order to be able to combine the measurements of all depth maps, a frequently used approach is to incorporate all of the measurements into a regular grid of non overlapping cubes called voxel grid. Each voxel within the grid then holds a value of a truncated signed distance function ($TSDF$). This means that in each voxel the distance to the nearest surface is stored. The sign depends on the viewpoint of the camera that produced the input depth map as visualized in Figure 2.7.

Voxels which are in view of the camera are assigned a positive sign, while information behind a surface is represented via a negative sign. The function values can also be truncated in a certain range i.e. between -1 and 1, which means that distance larger than 1 will be set to 1 and distances smaller than -1 will be set to -1. The process of integrating the depth maps generated from different input camera views is typically done via a weighted updating scheme [11]. Starting from an initialization of the $TSDF$ values $TSDF(X) = 0$ and weights $S(X) = 0$ at all possible locations X inside the voxel grid the algorithm applies the following update rule for adding a new depth map m with depth measurements $d_m(X)$ and weights $w_m(X)$ into the $TSDF$:

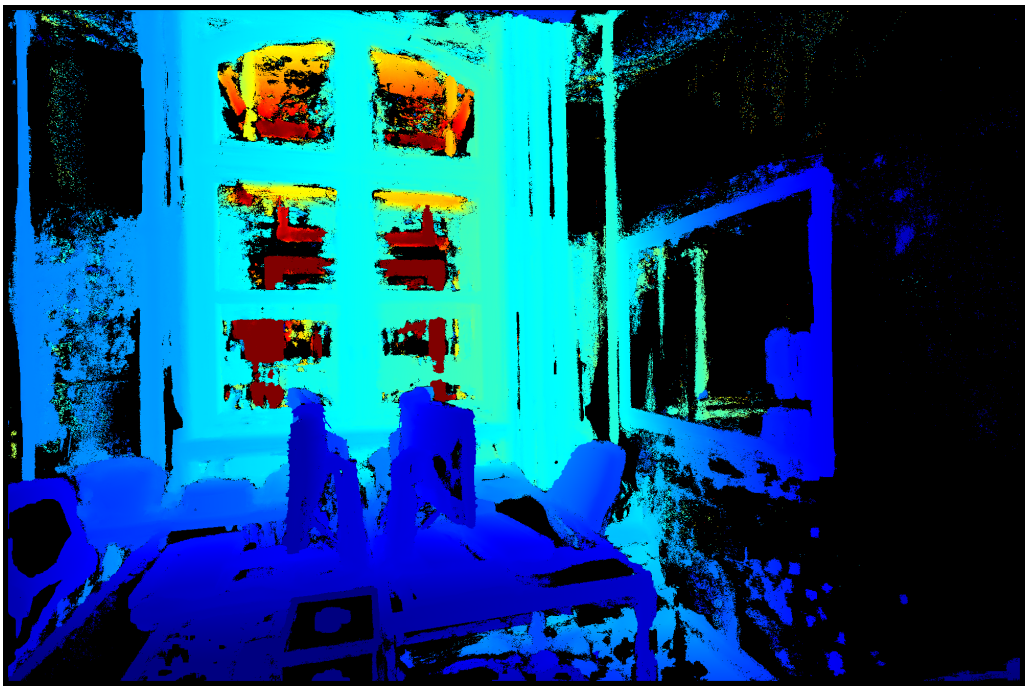


Figure 2.6: Depth map result from multi-view stereo using a patch match based approach [44]. It can be seen that for surfaces which are very homogeneous and do not contain a lot of texture it is hard to find matches and thus produce depth results. Furthermore reflections inside windows also introduce artifacts and are labeled with high depth values in this case.

$$TSDF(X)_{m+1} = \frac{S_m(X) \cdot TSDF(X)_m + w_m(X) \cdot d_m(X)}{S_m(X) + w_m(X)} \qquad (2.15)$$
$$S_{m+1}(X) = S_m(X) + w_m(X)$$

The weights can be chosen to have identical values i.e. 1 for all depth measurements or they can be computed based on additional information [64].

After the (truncated) signed distance function has been generated from the depth maps it is possible to create an output mesh from an iso-surface via the marching cubes algorithm [33]. Such a mesh is visualized in Figure 2.8.

2.3 Deep Neural Networks

In the following sections we will present the core concepts of neural networks in terms of learning and inference [20].

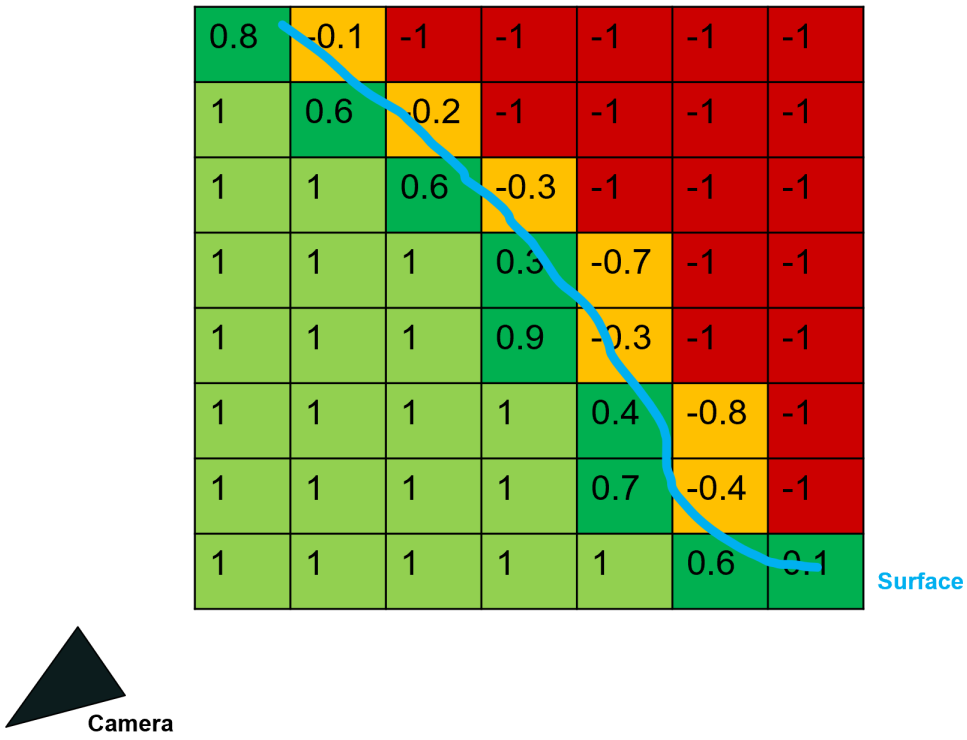


Figure 2.7: Depiction of *TSDF* values around a surface from a specified camera viewpoint in a regular 2D grid. Each of the values encodes the distance to the nearest surface from the viewpoint of the camera.

2.3.1 Base Principles

The essential principle behind neural networks is to create a learning process using a network of many interconnected layers of neurons. The connections between neurons are weighted. Adapting the weights changes the output of the network, when given a specific set of input values. Additionally the output of each neuron depends on its usually non-linear activation function, which introduces the possibility of creating complex output functions depending on the input. The output of the entire network is controlled via the activation function of the final layer, which can be adapted for a specific machine learning task, such as regression or classification. During the learning or training process the network is presented with several input and target pairs, which represent the outputs the network should ideally create, after receiving the given inputs. The network then adapts its weights while training on these samples, to produce the target outputs for the given inputs. The challenge during this process is to provide the network with a diverse selection of possible inputs. Furthermore the network weights should not be fine-tuned specifically to the training samples, as this will drastically reduce inference performance for general datasets, which is also known as over-fitting to a specific dataset. In the following chapters we will provide more detailed description for each of the aforementioned elements present

Figure 2.8: Output mesh from depth data fusion generated via marching cubes, which was created using the Open3D library [65].

in neural networks.

2.3.2 Neurons

A single synthetic neuron in an artificial neural network consists of one or multiple weighted inputs from other neurons, an activation function and one or more outgoing connections to other neurons in the network. A bias parameter is also present for each neuron and can be thought of as an incoming connection from a neuron which always outputs the value one. The output y_j of the neuron is computed from the incoming weights $w_{i,j}$ of I neurons with outputs y_i, the bias b_j and the activation function $g(x)$ as follows:

$$y_j = g\big(\sum_{i=0}^{I} w_{i,j} y_i + b_j\big) \tag{2.16}$$

The individual components of a single neuron are also depicted in Figure 2.9.

2.3.3 Multi-Layer Networks

When expanding the network view to take into account multiple layers of different amounts of neurons, it is advantageous to extend the notation to include individual layers l and the overall number of layers L. To be able to further simplify the model for explaining multi-layer networks, we focus on fully connected networks in further examples. In a fully

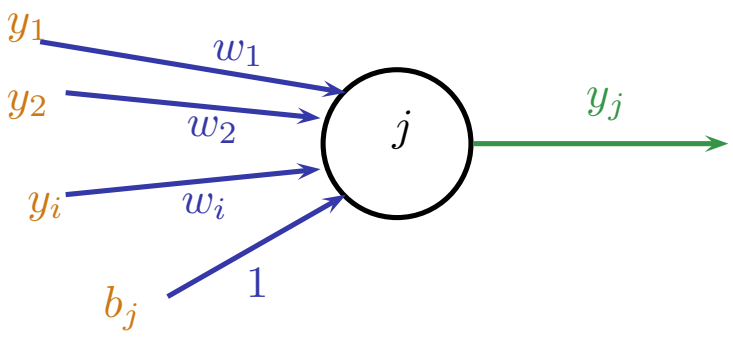


Figure 2.9: Parameters describing a single neuron in a neural network: The neuron receives weighted inputs y_i $\forall i$ with weights w_i $\forall i$ from the other (previous). The output y_j is computed using the weighted inputs and an additional bias b_j.

connected network every neuron from the previous layer is connected to all neurons from the next layer adjacent to it. We are also only considering feed-forward networks so there is only one direction where updates for neuron outputs are carried out and subsequently there are no loops. When looking at the general structure of a multi-layer network, one can find an input layer of one or more neurons which directly output the data of the input vector as the initial layer. Next are one or multiple layers which are denoted as hidden layers which contain varying numbers of neurons. The last layer acts as the output layer which yields the final network output and as such, the activation function and number of neurons in the output layer highly depend on the task the network is aiming to achieve. An example for the network architecture is visualized in Figure 2.10.

Because we are working in a fully connected setting, it is easy to formulate the neuron updates using a weight matrix $W^{(l)} \in \mathbb{R}^{N \times M}$ for all connections between layers l and $l+1$ containing M and N neurons respectively. Rows of this matrix will be indexed with j and columns with i. In each row of this matrix the weights for incoming connections to node j from nodes i are stored. One can the also define a vector $y^{(l)} \in \mathbb{R}^N$ containing all outputs from neurons in layer l. Additionally one can define a bias vector $b^{(l+1)} \in \mathbb{R}^M$ for the neurons in layer $l+1$. This bias can also be integrated into the weight matrix directly, by adding the bias vector as the last column of $W^{(l)}$ and inserting an additional element with the value 1 at the end of the vector $y^{(l)}$. The vector $y^{(l+1)} \in R^N$ can then be computed as follows:

$$y^{(l+1)} = g\big(W^{(l)} y^{(l)} + b^{(l+1)}\big) = g\big(z^{(l)}\big) \tag{2.17}$$

The activation function $g(...)$ in this case is applied element wise to the vector $z^{(l)}$ resulting from $z^{(l)} = W^{(l)} y^{(l)} + b^{(l+1)}$.

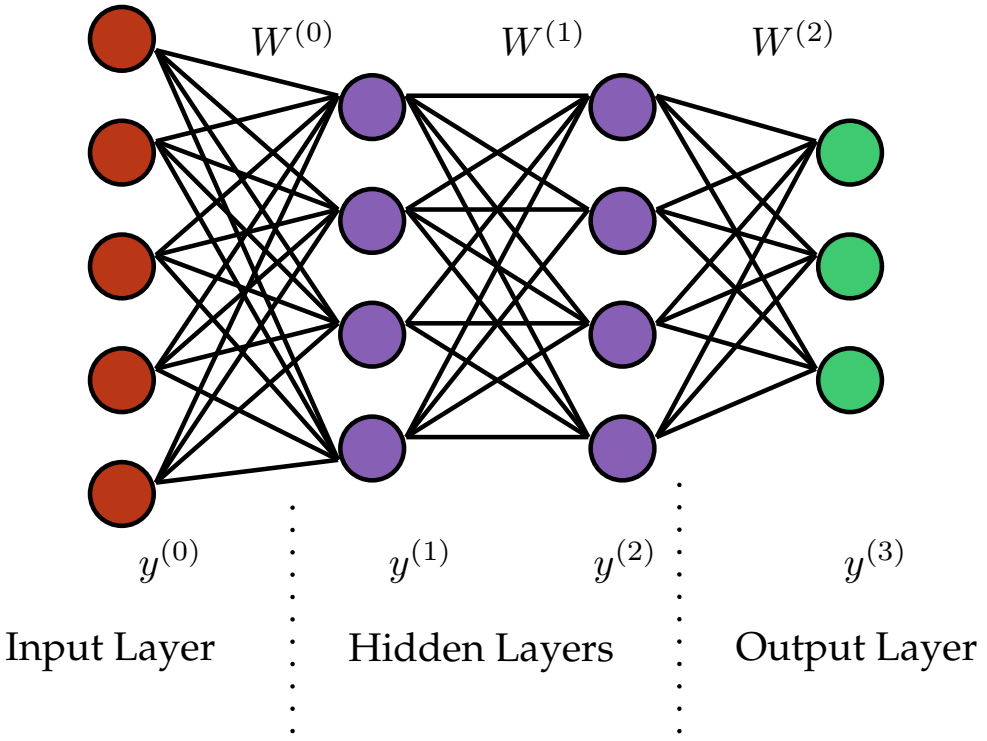


Figure 2.10: Visualization of a neural network with two hidden layers: The output vector $y^{(1)}$ of the first hidden layer is computed as $y^{(1)} = g(W^{(0)}y^{(0)})$ (in this case the bias is included in the weight matrix and an addition value of 1 was added to $y^{(0)}$). Subsequent outputs are also computed using this formulation.

2.3.4 Activation Functions

There are a multitude of non-linear activation functions which can be used in neural networks, however the used function has to be chosen with care, as it has a large impact on the learning process of the network. A popular choice is the rectified linear unit ($RELU$) activation function [20], which is defined as follows:

$$g(x) = \max(0, x) \tag{2.18}$$

Another possibility is the logistic sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

However, the latter function suffers from the vanishing gradient problem: Due to the fact that the gradient of this function is limited in magnitude, the accumulated values for

the overall gradient during the training phase will start to get very small, effectively only changing the network parameters during training by a very small margin.

Activation functions for output neurons are chosen depending on the task of the neural network. For regression tasks, one usually does not want to modify the value the neuron produces, given the weights and outputs of the previous layer, as a floating point value should be estimated directly. The function that is used in this case is defined as $g(x) = x$.

For classifying whether an input vector belongs to $C > 1$ classes, it is preferable that the network yields a probability of the input belonging to either class. This can be achieved with the softmax function which is defined for the vector of all outputs $X = \left[x_1, ..., x_C\right]$ as follows:

$$g(x_c) = \frac{e^{x_c}}{\sum_{i=0}^{C} e^{x^i}} \tag{2.20}$$

2.3.5 Learning and Loss Functions

The back-propagation algorithm is used to update the weights of the neural network during training [20]. It updates the network parameters based on the difference between the current prediction and the desired result. In order to achieve this the network first computes a prediction which is compared to a target and subsequently the network parameters are updated. The loss function is a central component of defining how the gradient is computed for back-propagating the weight updates through the network. For regression tasks, it is common to use the mean squared error function. This means that the error from difference between the network output $y^{(L)}$ of the last layer and a target vector t will be computed as follows:

$$E(y^{(L)}, t) = \frac{1}{2} \cdot (y^{(L)} - t)^2 \tag{2.21}$$

When repeating this procedure for n input/target pairs, all individual errors are averaged:

$$E(y^{(L)}, t) = \frac{1}{2n} \cdot \sum_{i=0}^{n} (y_i^{(L)} - t_i)^2 \tag{2.22}$$

A loss function used for classification is the cross-entropy error function:

$$E(y^{(L)}, t) = \frac{1}{2n} \cdot \sum_{i=0}^{n} \sum_{c=0}^{C} -t_{i_c} \log y_{i_c}^{(L)} \tag{2.23}$$

2.3.6 Computing Parameter Updates

In order to update the parameters of the neural network, one needs to compute the derivative of the error function with respect to the weight matrix $W^{(l)}$ and bias vector $b^{(l)}$ of each layer l. These derivatives can be computed for each input/target pair individually and then summed up and averaged afterwards, because of the structure of the mean squared error function used in this case. When computing the derivatives for the weights one can make use of the chain rule:

$$\frac{\partial E}{\partial W^{(0)}} = \underbrace{\underbrace{\frac{\partial E}{\partial y^{(L)}} \cdot \frac{\partial y^{(L)}}{\partial z^{(L)}}}_{\delta^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial y^{(L-1)}} \cdot \frac{\partial y^{(L-1)}}{\partial z^{(L-1)}} \cdot \ldots \cdot \frac{\partial y^{(1)}}{W^{(0)}}}_{\delta^{(L-1)}} \tag{2.24}$$

The derivative of $W^{(1)}$ is computed in an analogous way except for taking the partial derivative of $y^{(2)}$ with respect to $W^{(1)}$ at the end of the chain. The same holds for the bias parameter as well:

$$\frac{\partial E}{\partial b^{(0)}} = \frac{\partial E}{\partial y^{(L)}} \cdot \frac{\partial y^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial y^{(L-1)}} \cdot \frac{\partial y^{(l-1)}}{\partial z^{(l-1)}} \cdot \ldots \cdot \frac{\partial y^{(1)}}{b^{(0)}} \tag{2.25}$$

When looking at the relations above one can see that $\frac{\partial E}{\partial W^{(0)}} = \delta^{(1)} \cdot \frac{\partial z^{(1)}}{\partial W^{(0)}}$. The second differential can be solved in the following way: $\frac{\partial z^{(1)}}{\partial W^{(0)}} = \frac{\partial W^{(0)} y^{(0)} + b^{(0)}}{\partial W^{(0)}} = y^{0^T}$. From this reformulation the gradient of the weight matrix can be calculated as $\frac{\partial E}{\partial W^{(0)}} = \delta^{(1)} \cdot y^{(0)^T}$. The gradient of the bias term resolves to $\frac{\partial E}{\partial b^{(0)}} = \delta^{(1)}$, because $\frac{\partial z^{(1)}}{\partial b^{(0)}} = 1$. The gradients for general terms $W^{(l)}$ and $b^{(l)}$ are calculated as follows:

$$\begin{aligned} \frac{\partial E}{\partial W^{(l)}} &= \delta^{(l+1)} \cdot y^{(l)^T} \\ \frac{\partial E}{\partial b^{(l)}} &= \delta^{(l+1)} \end{aligned} \tag{2.26}$$

In order to compute the gradient updates it is necessary to compute the network output $y^{(L)}$ beforehand. This relation describes one of the two main components of the back-propagation algorithm, which is used to train neural networks:

- compute the network output $y^{(L)}$ using the input sample and given paramter settings
- compute the gradients $\frac{\partial E}{\partial W^{(l)}}$ and $\frac{\partial E}{\partial b^{(l)}}$ from $y^{(L)}$ and the target t.

Once all of the gradients have been computed they are accumulated and depending on the error function, an average of the summed gradients is then used in a gradient descent

step to update the parameters. In this step the accumulated gradients are subtracted from their respective parameters after being multiplied with the learning rate η. The initial parameters are typically set from a normal distribution. Executing the algorithm for I iterations on N training samples s using a learning rate of η is further elaborated on in Algorithm 1:

Algorithm 1: Back-propagation using the mean squared error loss.

$W^{(l)}, b^{(l)} = \mathcal{N}(\mu, \sigma),\ \forall l \in \{0, ..., L-1\}$
while $i < I$ **do**
  $\nabla W^{(l)} = 0,\ \forall l \in \{0, ..., L-1\}$
  $\nabla b^{(l)} = 0,\ \forall l \in \{0, ..., L-1\}$
  while $s < N$ **do**
    compute $y^{(L)}$
    $\nabla W^{(l)} = \nabla W^{(l)} + \frac{\partial E(y^{(L)}, t)}{\partial W^{(l)}},\ \forall \in \{0, ..., L-1\}$
    $\nabla b^{(l)} = \nabla b^{(l)} + \frac{\partial E(y^{(L)}, t)}{\partial b^{(l)}},\ \forall \in \{0, ..., L-1\}$
  $W^{(l)} = W^{(l)} - \eta \cdot \nabla W^{(l)},\ \forall l \in \{0, ..., L-1\}$
  $b^{(l)} = b^{(l)} - \eta \cdot \nabla b^{(l)},\ \forall l \in \{0, ..., L-1\}$

The learning rate parameter η has a large influence on the training process depending on the chosen value. Learning rates which are too small in magnitude have the disadvantage of having to train for many more iterations. On the other hand choosing a learning rate too large in magnitude could result in jumps around the local minimum of the error function and not consistently reaching an end result close to the minimum.

Another method of training is stochastic gradient descent [41]. When using this method one does not iterate over all input/target pairs in every iteration, instead one iterates over B samples, where B is the batch-size parameter. In this case B samples of a batch are random choices from the entire training set. This yields better results for training and also reduces the computational effort in each iteration. In this setting a measure denoted as epochs is used to define how many times the entire training set is given to the algorithm via batches.

2.4 Convolutional Neural Networks

In this chapter we will elaborate on the central ideas behind convolutional neural networks, which can be categorized as standard neural networks with additional caveats and constraints.

2.4.1 Core Concepts

As opposed to standard multi-layer neural networks, convolutional neural networks use the concept of convolutional filters to compute the outputs of neurons. When working with images, all of the neurons in the input layer are organized in a two dimensional grid.

In this case, it is common to either use an input array containing one channel of neurons (e.g. for gray-scale images) or three channels (RGB-images).

The hidden layers in convolutional neural networks consist of differing amounts of channels, with each channel holding neurons in a two dimensional grid. Each of these neurons has connections to a local patch with size $W \times W$ in the two dimensional grid in the previous layer. If the previous layer is the input layer and has three channels the size of the window is $W \times W \times 3$. To limit the amount of parameters to estimate while training the network, each channel of a hidden layer uses the same set of weights for connecting its neurons with local patches in the previous layer. This means that each channel in a hidden layer computes its output depending on $W \cdot W$ parameters, a concept that is also shown in Figure 2.11. This relation yields one of the core components of convolutional neural networks which are denoted as filters. Each channel of a hidden layer represents such a filter. Two dimensional inputs coming from hidden layers or the two dimensional network input can be convolved with a filter, which is defined using a filter kernel with a given size (either $W \times W$ or possibly $W \times W \times 3$ for the input layer) .

Formally, a convolution in discrete space of a two dimensional MxN image I with a two dimensional kernel K of size $W \times W$ is defined as follows:

$$R(y, x) = \sum_{m=0}^{M} \sum_{n=0}^{N} I(m, n) \cdot K(y - m, x - n) \tag{2.27}$$

When accessing locations outside of the kernel or image, a padding strategy i.e. using zeros, has to be defined. When convolving the input with the filter, the filter moves over the input along each of the two dimensions and computes a weighted sum using the values inside the filter window as weights, which again yields a two dimensional output.

Analogous to a standard neural network, the number of neurons in the last layer, which are connected to every neuron from the second-to-last layer, depends on the number of outputs that should be generated by the network.

2.4.2 Additional Hyper Parameters of Convolutional Architectures

Each of the neurons in the hidden layer represents an output which has been computed by convolving with a kernel on a certain spatial location of the input. The number of neurons within each of the filters in the hidden channel depends on a stride parameter. The stride parameter controls by how many neurons the filter is shifted in each dimension when moving the kernel window. A larger stride means less neurons are needed for representing the output. Furthermore pooling layers can be used to increase the receptive field of the network further, however applying them results in a reduction of output neurons compared to the input of the layer. Pooling layers apply an average or maximum operation to compute the result from values inside a filter window moved across the input dimensions with a specified stride. This is used to widen the receptive field of the network, which

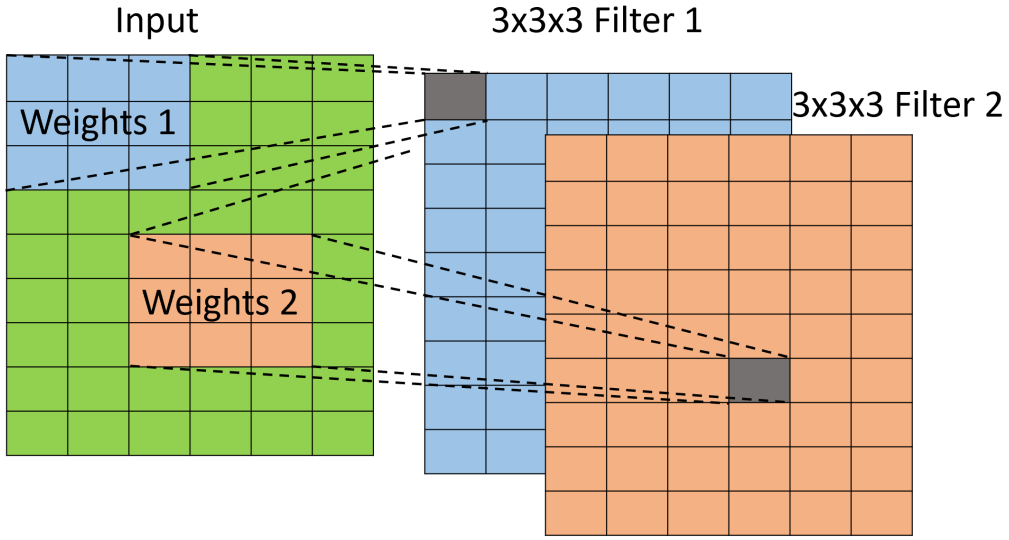


Figure 2.11: Figure depicting the basic principles of using multiple convolutional filters in a *CNN*. It can be seen that each filter uses a different set of input weights for all convolutions with the input. In this case the filter dimensions are 3×3, which means that each of these filters have 9 parameters which will be estimated during the training procedure.

means global contexts of the input can be taken into account. Another method to widen the receptive field are dilated convolutions [56] where a dilation parameter defines spacing between the weights inside the filter window.

The concept of two dimensional convolutional networks can also be extended to be used for three dimensional input data by not only convolving with a kernel volume over the height and width dimension but also along the depth dimension.

2.5 Summary

In this chapter, we first introduced the fundamental principles of multi-view geometry by elaborating on the pinhole camera model and epipolar geometry. Furthermore, we introduced the image-based 3D reconstruction process via *SFM* and dense reconstruction techniques. We have also introduced basic concepts of neural networks, as well as convolutional neural networks.

3 Related Work

Contents

There exist a variety of approaches in the field of 3D scene completion, which focus on different use case scenarios in terms of scope and scalability, but also in terms of the format used for the input data. However, when comparing the techniques on a very high level, one can distinguish between machine learning based approaches and traditional algorithmic approaches. In the following chapters we will discuss various works which can be classified using either of the aforementioned categories.

3.1 Traditional Approaches

In recent years, many methods for scene completion have been proposed, which do not rely on learning based techniques to reconstruct missing geometry. There are approaches that rely on solving optimization problems, which describe the process of completing the geometry in a rigorous mathematical way, such as Poisson surface reconstruction [26]. This method reconstructs a mesh from an oriented point-cloud by solving a Poisson optimization problem. Poisson based reconstruction can also be used as a post-processing step for algorithmically filling holes in meshes which is utilized by Zhao et. al.[61]. Their system first creates a rough result using the advancing front mesh algorithm [19] and further processes the result by solving a Poisson optimization problem. Least squares meshes [48] reconstruct a surface from control points which are used as an input to the algorithm. This is done by setting up an equation system, which is linear in nature and obtaining the solution using a least squares method. However, this method cannot directly infer new triangles in the mesh but rather computes the spatial location of already existing triangles

within the mesh. Additionally there are also methods, which use a database of complete models in order to replace objects with missing geometry. Such a method, which uses a feature based approach to match models from a database with incomplete models, has been developed by Li et. al. [32]. Similarly Kim et. al. [28] learn common shapes from given 3D scenes and use these to complete missing geometry. Furthermore, approaches have been proposed that use specific properties of objects, such as symmetry, to add more information to partially reconstructed geometry by continuing the geometry according to its symmetric properties [52]. Related are methods which try to find such symmetries in 3D space by detecting feasible symmetries via correspondences [35] or by generating points of interest for detecting symmetries via a diffusion equation [46]. Furthermore, an algorithm that checks for patterns by repeatedly transforming parts of the geometry and then utilizes the detected patterns for geometric completion was developed by Pauly et. al. [38]. There are also approaches that use additional geometric scene assumptions, i.e. Holzmann et. al. [25] propose a method where planes extracted from the scene are used to improve the reconstruction result for planar surfaces. Related to this method is an online system developed by Dzitsiuk et. al. [15], which is designed to complete holes on planar surfaces, decrease noise in the reconstructed geometry and provide a segmentation of objects present in the scene. This is done by detecting planes for each object in the scene via a clustering approach. The planes can then be used during the reconstruction process to reduce noise, fill holes in planar regions and segment objects by utilizing the results of the clustering. In the following chapters we will elaborate on a selection of the methods described above in more detail.

3.1.1 Poisson Surface Reconstruction

One of the more prominent traditional approaches for 3D scene completion is Poisson Surface Reconstruction [26], which generates a mesh given an oriented point cloud. In general, the goal of this technique is not to infer larger chunks of missing geometry, but rather construct a surface mesh that spans the missing geometry between points in the input point cloud. The name Poisson surface reconstruction stems from the fact that this technique formulates the reconstruction problem as a Poisson problem. This is done by representing the output of the reconstruction as a three dimensional indicator function $\mathcal{I}$. For a given 3D point the function returns the value 1 if the point is located inside a mesh and 0 if it is located outside a mesh. It follows that the gradient $\nabla\mathcal{I}$ of this function is non-zero only at points that lie on the surface of the mesh itself. The points from the given oriented input point cloud P also lie on the surface of the mesh that needs to be reconstructed. Thus, one wants to find a 3D indicator function that minimizes the difference between the given oriented input points and the gradient of the indicator function. This can be expressed as follows:

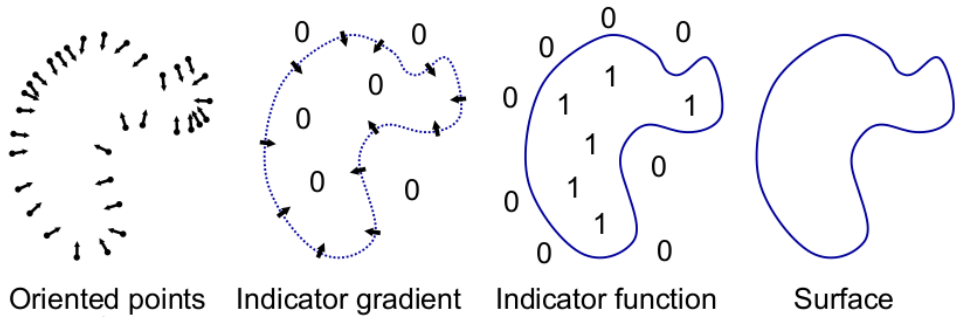


Figure 3.1: Relation of input point cloud, indicator function and surface to be reconstructed. It can be seen that the gradient of the indicator function is zero except on the surface of the mesh itself. The indicator function yields values of 1 inside the surface and values of 0 outside. Figure taken from [26].

$$\min_{\mathcal{I}} F(\mathcal{I}) = \min_{\mathcal{I}} ||\nabla\mathcal{I} - P|| \tag{3.1}$$

A Poisson problem is of the following form: $\nabla^2\phi = f$. One wants to find the function ϕ, whose Laplacian denoted by ∇^2 is equal to the function f. The problem described in Equation 3.1 can be reformulated into a Poisson problem using the divergence operator ∇. Applying this operator yields $\min_{\mathcal{I}} \left|\left|\nabla^2\mathcal{I} - \nabla P\right|\right|$, so effectively one wants to find $\mathcal{I}$ such that $\nabla^2\mathcal{I} = \nabla P$. It can be seen that is equivalent to a Poisson problem with $\phi = \mathcal{I}$ and $f = \nabla P$. The aforementioned relations of the indicator function, input point cloud and the surface to be reconstructed are also visualized in Figure 3.1.

The solution to this problem, after rewriting it into a linear optimization problem, can be obtained using a linear solver. One of the advantages of this technique is, that it is robust to noise in the point cloud measurements. As this method only uses the oriented point cloud as an input, reconstructing the surface via solving the optimization problem can lead to artifacts in the reconstructions. There is also no way for the algorithm to infer geometry from unknown or empty space, as there will be no point cloud data there.

3.1.2 Completion using Database

One can also use a database of predefined shapes or even entire objects to complete missing scene geometry. Li et. al. [32] developed a system using a similarity measure, which utilizes descriptors for reconstructed and database geometry to replace missing geometry. The main steps of the algorithm are depicted in Figure 3.2. The technique was designed to be used during real time 3D reconstruction. Furthermore, this approach can be seen as an extension to traditional image inpainting [3] techniques where similar patches from the input image are used to fill the missing intensity values in the image. To achieve this, keypoints are computed for complete shapes stored in the database, as well

as for incomplete reconstructed shapes. These keypoints are computed by applying a 3D corner detection algorithm to the point cloud representation of the respective geometry. Additionally, descriptors are computed using a distance function. The descriptors encode local distance function information from their respective keypoints in addition to global information based on planes from the entire shape. This global information is used to align the database shape with the reconstructed shape during the descriptor matching phase. This approach works well, if the original scene geometry can be accurately represented by the models retrieved from the database. As the entire model is swapped out, this approach avoids producing artifacts while completing the geometry, if the alignment within the original model is correct. However, the main limitation of this approach is the similarity of the database models to the real reconstructed geometry. Another potential problem is that the amount of keypoints and subsequently models detected within the scene might not match the actual amount of models contained in the space.

3.1.3 Shape from Symmetry

The basic idea of Shape from Symmetry [52], is to exploit the symmetric properties of many objects in 3D in order to complete missing geometry. After a specific type of symmetry that the shape of the object adheres to is inferred from an incomplete object, this knowledge can be used to complete the object geometry. To achieve this symmetries are classified via a two measures. The first is an anchor for the symmetry which binds it to a specific location e.g. either a plane or point. In addition the type of symmetry, for instance reflective or spherical defines how the symmetry is used to create geometry. Examples for different anchors for reflective symmetries are visualized in Figure 3.3.

These symmetric properties can also be combined as well to create more complex symmetries. One can also define a hierarchy, where symmetries that are more general versions of their respective sub-symmetries are positioned at a higher level. The basic idea of how to detect the type of symmetry in the incomplete geometry, is to define a score for the various symmetry types and pick the symmetry which yields the highest score for the given geometry. The score is computed from a probabilistic model, which also accounts for noise. The main metric of the model is determined by computing the complete model under the assumption that the type of symmetry to be scored is the correct one and then check if the points added to the model lie within occluded space or not. If a lot of them are positioned in non-occluded space they should have been visible in the first place and it is less probable that the type of symmetry used to generate these points is the correct one. This approach has the advantage that it creates very good results for objects, where the symmetry assumptions hold and the correct symmetries have been detected. However, for objects which are made up of a lot of different types of symmetry, these become harder to detect and hence the completion result will be negatively impacted. Furthermore more complex non-symmetric geometry will not be completed in a geometrically sound way by this approach.

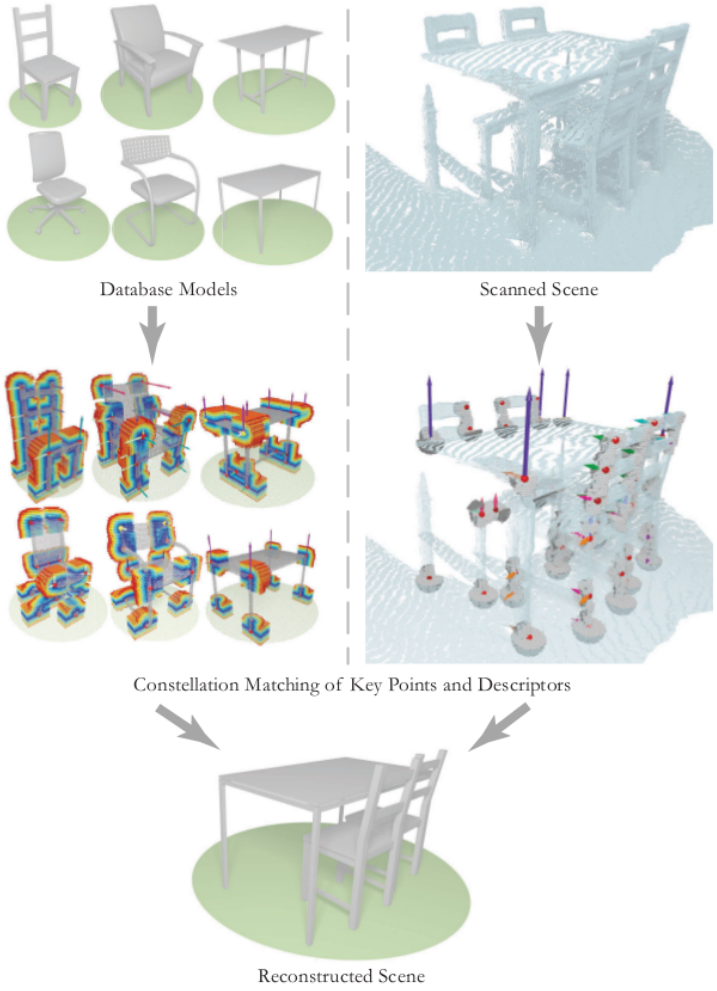


Figure 3.2: Flow chart showing the main steps for database assisted scene completion. First descriptors and computed for the database models and the reconstruction, these are then matched and subsequently the reconstructed results are replaced with matched models from the database. Figure taken from [32].

3.1.4 Semantically Aware Urban 3D Reconstruction with Plane-Based Regularization

Holzmann et. al. [25] propose a method where semantic information is used during the reconstruction of 3D geometry, to create a more complete output mesh as seen in Figure 3.4. In this approach planes are detected and used in a reconstruction pipeline which utilizes tetrahedra. Semantic information is then used to regularize the reconstruction in a way that yields planar geometry for walls and smooth geometry for objects around buildings. To achieve this, a line-based 3D reconstruction is performed. A dense point cloud is then generated from dense depth maps using a multi-view stereo approach. The incorporated semantic information is computed from input images using a *CNN*. This

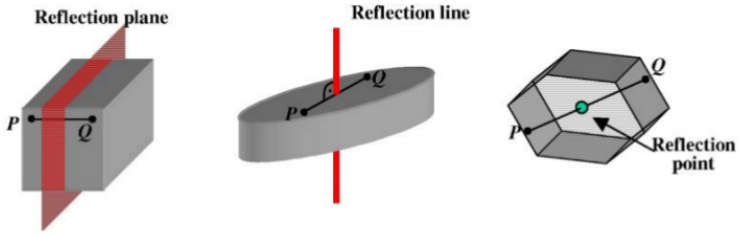


Figure 3.3: Different anchor points for reflective symmetry i.e. planes, lines and points. Image taken from [52].

information is then projected into 3D space using the extrinsic and intrinsic parameters of the respective camera poses. If different semantic information from several images is projected to the same point, a majority vote is used to determine the most probable label. The line-based 3D reconstruction allows for improved performance when detecting planes in the reconstructed geometry. Additionally, the detected planes can also be filtered using the semantic labels of the lines which have been used to reconstruct them. The detected planes and semantic information are then used in a tetrahedral surface reconstruction algorithm. The completion metric of planar surfaces like building walls is improved significantly. This is the case, because custom energy terms depending on the semantic class can be included while solving for labels which determine whether a tetrahedron is inside or outside a 3D surface.

Figure 3.4: Outputs of the reconstruction with plane based regularization. It can be seen that the planar surfaces on the buildings have been densely reconstructed. Figure taken from [25].

3.2 Learning-Based Approaches

Due to the increase in popularity of neural networks in many areas of computer vision research, a wide variety of learning based methods for scene completion has emerged in recent years. Recently generative adversarial neural networks [21] have received an increased amount of focus from researchers. These networks consist of a generator network that generates data and a discriminator network, which infers the viability of the generated

data. This generative adversarial model is utilized in a system proposed by Wu et. al. [55], that was designed to complete the geometry of an object contained in a depth image. In case a depth image is not given it is estimated via an additional neural network. Then the network estimates the complete geometry from the depth map using a neural network. The discriminator network of a generative adversarial network is subsequently used to check if the completed result is viable, which improves the result of the previous network by introducing an additional loss term. This method is limited by the fact that it is only able to complete a single object contained in a depth image. There exist methods relying on random forests to fill in missing geometry caused by occlusions [16] in an entire depth image. However, similar to the work by Dai. et. al. [12] based on *CNN*, this approach is also more focused on completing single objects contained within the depth images and not entire scenes. Furthermore, the method is not able to predict semantic information for objects contained within the scene. In order to achieve this task, approaches have been developed that specifically focus on completing the geometry and semantics contained within a single depth image using a *CNN* [47] [18] or voxel based Conditional Random Field (*CRF*) [27]. A disadvantage of the aforementioned methods is that they focus specifically on the information contained within a single depth map, which can lead to problems in overlapping regions where different predictions have been made. A neural network based technique called ScanComplete [13], which relies on a hierarchical approach, aims to solve this problem and is able to reconstruct larger scenes and estimate the semantic labels of the scene. This method works with a *TSDF* encoded in a voxel grid, similarly to a method developed by Cherabier et. al. [8], which makes use of an encoder and decoder architecture combined with a regularized optimization procedure to complete an input *TSDF* and additionally outputs semantic labels as well. Zeng et. al. [57] developed another neural network based system, which is able to complete and refine larger scenes and takes as an input an initial point cloud reconstruction. It then infers a procedurally generated complete model that encodes the same geometry as the input point cloud. One of the main limitations of using *CNN* based architectures in 3D are the high memory requirements. Hence, Riegler et. al [40] proposed a more efficient data structure than regular voxel grids for 3D deep learning. Moreover, Zhang et. al. introduce spatial group convolutions [58], which allow for more performant inference times. As done in the previous section with traditional approaches, we will elaborate on a selection of the learning based methods mentioned above in greater detail.

3.2.1 ScanComplete

In Dai et. al. [13], a method named ScanComplete was presented, which consists of a fully convolutional, hierarchical and auto-regressive neural network architecture can be utilized to complete missing scene geometry and predict the semantic information linked to this geometry, as exemplified by a result in Figure 3.5. The network is trained on data acquired using the synthetic SUNCG dataset [47], with the following properties: The given

incomplete input scene is represented as a regular voxel grid. Geometric information is encoded within this voxel grid using a truncated signed distance function. The complete ground truth counterpart to a given input scene is represented by a non-signed distance function. In this case, the sign is not used, as all of the space within the ground truth geometry is known. For the input scene the negative sign represents voxels which lie behind a given surface, hence parts of the geometry which might not have been reconstructed properly. Additionally semantic ground truth information which assigns a semantic label to each voxel is provided, in order to train the network to predict voxel semantics from the given input signed distance volume. During inference the network processes the input data using 3D convolutions with filter kernels, which have previously been learned during the training phase. Furthermore due to the autoregressive architecture of the network, the data is split up into 8 equally sized voxel groups. The output of all previously processed voxel groups is used as an additional input for processing further groups. The network features three distinct hierarchies that process voxel grids of different resolutions (coarse to fine). This allows the network to take into account a wider global context, while taking advantage of higher resolution input needed for reconstructing fine details. One of the main advantages of this approach is that it is able to infer new geometry i.e. the base and arm rests of a chair when only given incomplete parts of the supports at the bottom. This can however lead to objects having different geometry than in the actual scene. Furthermore the level of detail which can be reconstructed is limited by the number and encoded voxel sizes of the hierarchy layers. The representation of the encoded truncated signed distance function is also a regular voxel grid, which can lead to memory issues.

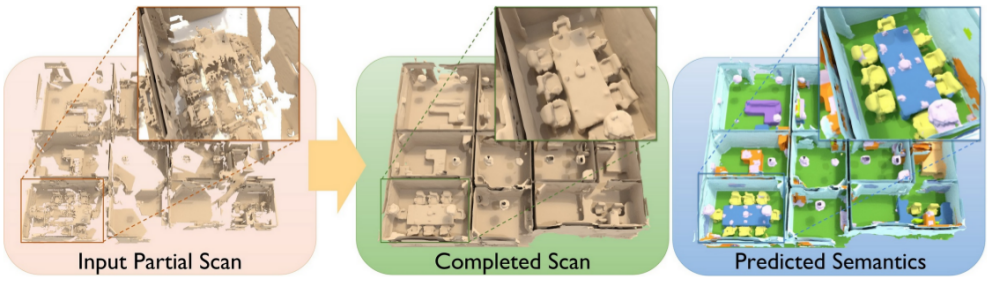


Figure 3.5: Sample result and semantic prediction from ScanComplete [13]. Imagery taken from [13].

3.2.2 Semantic Scene Completion from a Single Depth Image

In this section, we will describe the approach by Song et. al. [47] in detail. The method infers occupancy information and semantic labels from a single depth image using a *CNN* (SSCNet). A result is visualized in Figure 3.6. The input geometry for the network is encoded as a truncated signed distance function stored in a voxel grid. Furthermore the distance function is reversed, such that the gradient of the function points towards

the direction of the surface, which has been shown to improve results. The network then predicts a semantic label for each voxel if it is occupied, otherwise the label will be "empty". In terms of architecture, the network makes use of dilated convolutions to widen its receptive field. Furthermore it is a deep residual network utilizing skip connections. The training data was generated from the synthetic SUNCG dataset [47] and encodes the surfaces as a reversed truncated signed distance function. One of the disadvantages of this approach is, that it only completes geometry for single depth images. If one wanted to complete an entire scene, the completed geometry from single-views would have to be concatenated which can lead to problems on reconstruction boundaries. Furthermore, the context of the surrounding geometry cannot be taken into account. During their evaluation Song et. al. found that implicitly performing occupancy information completion while predicting semantic labels improves the scene completion performance.

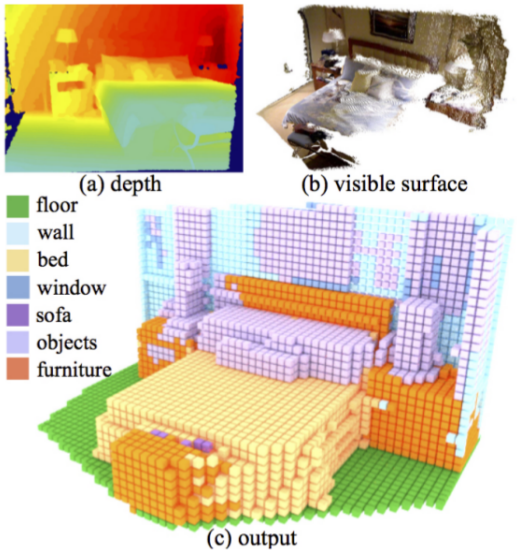


Figure 3.6: Input and predicted result for SSCNet [47]. Figure taken from [47].

3.2.3 Two Stream Semantic Scene Completion

Garbade et. al. [18] use semantic information obtained through RGB images as an additional input for their single depth image occupancy and semantic labeling framework. The used network architecture is built upon from Song et. al. [47] and takes both a reversed truncated signed distance function in voxel grid form and semantic labels for each voxel encoded in three dimensions as an input. The semantic information is inferred from the RGB input images using a *CNN* and then projected into 3D space using the corresponding depth map. Encoding the semantic class labels in three dimensions avoids the memory limitations of a one hot-encoding approach. Furthermore encoding the labels using just their values in one dimension limits the expressiveness of the input. The proposed ap-

proach yields better results in terms of scene completion than SSCNet [47], however, as it still only performs scene completion on a single depth image, the same limitations as mentioned above apply when completing larger scenes.

3.2.4 Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis

This method from Dai et. al. [12] focuses on completing single objects using a *CNN* and then refines the result using a database driven shape synthesis algorithm as visualized in Figure 3.7. This is done by matching the coarsely completed result produced by the neural network with a higher resolution model obtained from a database and then using this model to refine the coarse result. The neural network inferring the coarse result uses an encoder-predictor architecture. One of the novelties of this approach, is that this neural network also takes semantic predictions regarding the shape of the object as an additional input. Overall, the method surpasses the state of the art in terms of performance, however this approach is focused on completing a single object and not an entire scene. Moreover, it is limited by the amount of data available in the database.

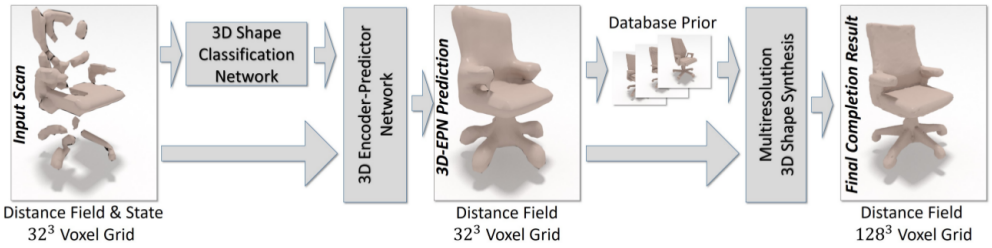


Figure 3.7: Flow charts depicting the main steps of the algorithm proposed by Dai et. al. [12]. The neural network prediction is refined using a model obtained from a database. Imagery taken from [12].

3.3 Summary

Using the works presented above as a point of reference, one can see that completing 3D geometry is a highly complicated task, which either requires (assumed) prior knowledge of the geometry, or powerful learning based techniques to achieve acceptable results. With regards to completing image based reconstructions, learning based approaches are preferable, as using the discussed traditional approaches would introduce severe limitations to the general completion capability of the system. For instance, Poisson surface reconstruction [26] is not able to infer geometry when no measurements are available, which is a key component when trying to fill in missing geometry in image-based reconstructions. Furthermore methods supported by a database [32], would not be able to complete shapes not already known to the system. This is another advantage of learning based methods

as they are able to infer entirely new geometry based on the existing scene. Symmetry based algorithms [52] might work well for some elements of the scene, however their performance is limited by the complexity of the given objects. When utilizing learning based approaches, methods taking the entire scene into account such as ScanComplete are likely to achieve good results. This is not the case when completing single depth maps [47], as one would have to merge the depth maps on the boundaries and introduce overlapping depth measurements in these locations. Moreover, the system would not be able to utilize information outside of the viewpoint of a single depth map, which oftentimes provides a significant amount of additional context. For the aforementioned reasons, we chose the ScanComplete architecture to use as a basis for our work.

4 Methodology

Contents

In this chapter, we first give a high level overview of the proposed method, as well as the format of the input and output data. Then we will elaborate in more detail on the proposed network architecture, which takes semantic information and geometric information encoded as a *TSDF* as an input and produces a completed model and estimated semantic labels as the output.

4.1 Overview

The main purpose of the proposed method is to fill in missing geometry of incomplete 3D reconstructions in a consistent way with respect to the existing geometry. There are many potential use cases for this work and we want evaluate how our approach performs for completing 3D reconstruction generated from synthetic scenes and also investigate the methods performance on image based reconstructions of real world scenes. This method could also be integrated as a post processing step in a dense 3D reconstruction pipeline, in order to improve the quality of the output mesh. In order to achieve this task the method uses geometric data and additional semantic data, which labels the corresponding geometric input using a defined amount of semantic classes. The semantic input provides the network with more context in regards to the physical structure of the geometry that should be completed. For instance, if geometry associated to the semantic classes wall, ceiling or floor surrounds missing structure, the network can infer that the structure of the missing geometry should be planar. The completion itself is performed using a *CNN*

which takes as an input an incomplete geometric reconstruction encoded as a *TSDF* in a regular voxel grid. The voxel grid is obtained by integrating depth information from multiple input depth maps. Additionally semantic labels for each voxel are provided as an input. A high level overview of the work-flow is provided in Figure 4.1. A quantitative analysis of how this modification improves inference performance, when compared to only using truncated signed distance function input, is provided in chapter 6.

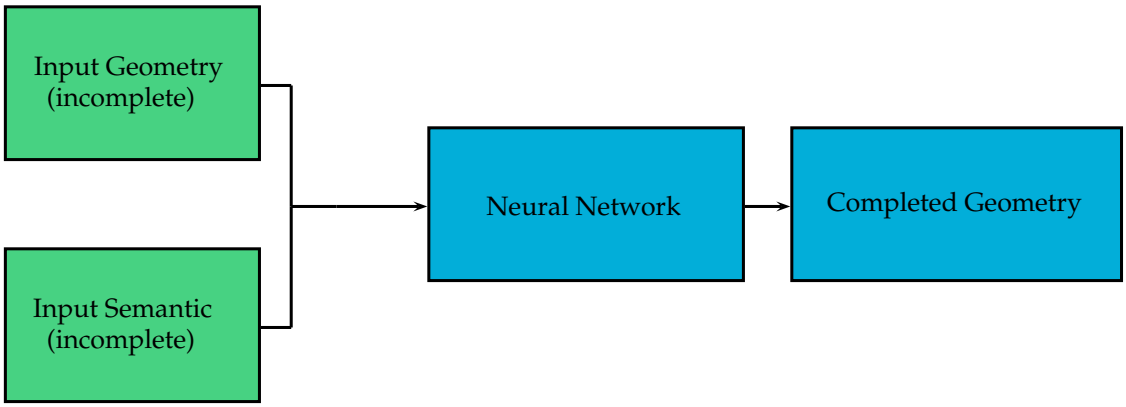


Figure 4.1: The main processing steps of the proposed pipeline. Geometric and semantic data is used as an input for the neural network, which outputs the completed geometry.

4.2 Input Data

In the following section we further elaborate on the used input data formats and on the concept of generating 3D semantic information from given semantic segmentation images and camera poses.

4.2.1 Conventions

The proposed pipeline expects geometric and semantic input data. The format of the geometric data is specified as a truncated signed distance function (*TSDF*) encoded in a regular voxel grid. The *TSDF* encodes the distance to the nearest surface for each voxel, while the sign relates to the spatial position of the voxel in relation to the surface. The metric in which this distance is measured are voxels, in our case a value of one encodes a distance of three voxels. The truncation clips values above a defined threshold and for the input data of the proposed method the threshold is set to one. We obtain values for the *TSDF* by combining depth data from multiple input depth maps. Further details with regards to this combination and *TSDF* are provided in Section 2.2.4.

The semantic input information is encoded via class labels and stored in a regular voxel grid with equal dimenions as the grid storing the geometric input. In order to acquire the semantic information in the form of a voxel grid, we project labels from semantically

segmented images into 3D space using the respective camera poses the images were taken with as described in Section 4.2.1.1. The semantic classes used in this additional semantic input were adopted from ScanComplete [13] and have been extended by an additional clutter class. The chosen labels provide separate classes for common items available in indoor environments. Moreover three classes, namely furniture, object and clutter, are used to label a variety of other objects that do not fit the geometry of the rest of the classes. The semantic labels are defined as follows: wall, ceiling, floor, bed, chair, furniture, objects, clutter, sofa, desk, TV, window, unknown space, empty space.

The dimensions of both input voxel grids are not restricted to a specific value, as the network architecture does not contain convolutions using a stride higher than one or pooling layers, hence it is a fully convolutional architecture and as such it can handle arbitrary input sizes.

4.2.1.1 Integrating Semantic Information into Voxel Grid

For creating three dimensional semantic information, we leverage generated depth maps and their respective camera poses. Each of the depth maps is integrated into a dynamically allocated voxel grid using the weighted updating scheme described in Section 2.2.4. In this case, the weights are chosen as follows: $w_m(X) = 1,\ \forall X$, where X are all possible voxel locations within the voxel grid. In order to also be able to create a voxel grid containing semantic class information, we introduce a per voxel array containing class votes that adheres to the following update scheme:

$$H(X, c, m+1) = H(X, c, m) + l(X, c),\ \forall c \in C \tag{4.1}$$

In Equation 4.1 the parameter C denotes all of the possible classes, while $H(X, c, m)$ represents the per voxel array containing class votes for class c after integrating depth map m. In this case $l(X, c)$ represents the class information obtained from the semantic segmentation image projected into 3D space using the depth information obtained from the depth image at a given pixel location:

$$l(X, c) = \begin{cases} 1 & \text{if } \text{proj}(X, m) = c \\ 0 & \text{else} \end{cases} \quad \forall c \in C \tag{4.2}$$

The function returns a value of one for a given c, if the semantic class of this voxel is c according to the projected information from the segmentation image denoted by $\text{proj}(X, m)$, otherwise it returns zero. This means that if the projected class label from the semantically segmented image, which projects to voxel X matches c, the function returns 1.

The set C represents all possible class labels $C = \{0, ..., 13\}$. Class labels with the

highest score obtained from Equation 4.1 are chosen as the result in a final post processing step:

$$L(X) = \underset{c}{\operatorname{argmax}}\, H(X, c) \tag{4.3}$$

As mentioned above, the voxel grid is dynamically allocated as specified in the Open3D implementation [65] [64]. This means that the grid is constructed from smaller sub-grids of a defined size. To generate these sub-grids, a point cloud is first created from the currently processed depth image and camera pose. The sub-grids are then created, if necessary, in order to be able to integrate the information within a distance of three voxels from each point in the point cloud.

4.3 Network Architecture

In this chapter, we discuss the main features and the convolutional architecture used in ScanComplete [13] in more detail, as well as on the extensions that were performed to take advantage of semantic input information.

4.3.1 Network Modalities

The network architecture we use in our work, is designed to process *TSDF* data encoding up to three different levels of granularity, as originally proposed in ScanComplete [13]. These levels are defined by the size of individual voxels in the regular input voxel grid. The network is able to complete missing information in the *TSDF* and at the same time predict the semantic labels of every voxel in the grid. This is done via two network branches using different loss functions for the respective tasks, which are described in more detail in Section 4.3.3.

Subsequently the different levels are trained on data with different voxel sizes, while the network architecture for each of the hierarchy levels is identical. Furthermore, the inference results of the coarser hierarchy levels are used as an additional input channel in the finer hierarchy level directly succeeding it.

Another feature of the network architecture is an autoregressive training procedure. The main idea behind an autoregressive model is to split up the estimation of a probability distribution into multiple conditional distributions which are then multiplied. This means that one has to split the input data of the network into smaller sub volumes and then use the predictions of previous sub volumes when inferring on the current sub volume, as its prediction depends on the output of the previous predictions because of the aforementioned underlying conditional dependence. Such a training procedure has already been investigated for the 2D image case [54] and performant implementations exist [39].

For the autoregressive approach of the ScanComplete [13] architecture, the input voxel

grid is split into equally large groups of voxels. The groups are computed by enumerating all possible sub grids with a stride of two from the input grid. This needs to be done because the voxels contained in a sub volume must be independent of each other in order for the assumptions of the autoregressive model to work, hence a stride of two is introduced. Because the grid is three dimensional and the stride is two there are eight possible sub-grids with a stride of two that can be extracted from the input. In this case each of these sub-grids marks a group of voxels. The weights of the network are then determined for each of the groups separately, however the ground truth truncated distance values for all groups on which the current group depends, are used as additional inputs. Moreover, as already mentioned above the inference result of the coarse hierarchy level is also added as an input channel for training all of the groups. When training the network, one can either use the inference result or the ground truth data for the inputs of the coarser hierarchy levels as an additional input for each of the groups. Thus, the dimensions of the coarser result have to be half the dimensions of the current result in order to be concatenated with the voxel group data, which has been generated with a stride of two from the current input volume.

4.3.2 Convolutional Architecture

The initial geometric and semantic input data contained in each voxel group is first convolved with three dimensional convolutional kernels with size $3 \times 3 \times 3$, which are activated using $RELU$. In the network architecture there is a sequence of convolutions which is repeatedly used, which we denote as the basic element of the network. The resulting filter channels from the intial convolution of the input data are then convolved with this basic element of the network. This element is structured as follows: An initial convolution with a $1 \times 1 \times 1$ kernel activated by $RELU$, followed by a convolution with a $3 \times 3 \times 3$ kernel activated by $RELU$ and a final convolution with a $1 \times 1 \times 1$ kernel. The original input of the basic element is then directly added to the non activated output of the last $1 \times 1 \times 1$ convolution. A $RELU$ activation function is then applied to the result of this addition. This follows the structure of deep residual networks [23], which utilize these connections to be able to train neural networks with a large amount of layers.

The data from the coarse hierarchy level is first convolved with $3 \times 3 \times 3$ filter kernels. Then this data is also convolved using the basic element described above. What follows in the network architecture is the combination of the filter channels which are the results from applying the above described convolutions to all of the input data. Next, two of the basic convolutional elements are applied to this combination of filter channels.

The network then splits into two separate branches, one for predicting the semantic labels of each voxel group and one for predicting the $TSDF$ function. Within each branch, the incoming data is convolved with a basic element first, before a final convolution with a $1 \times 1 \times 1$ kernel is conducted. The number of filters for this final kernel depends on the number of network outputs. It follows, that for the semantic branch the number of

filters is the number of semantic classes used and for the *TSDF* branch there is only one output filter, which directly predicts the function value. During inference, the class label with the maximum probability for each voxel is chosen as the final label assignment in the semantic prediction branch. Furthermore, each of these branches is trained using a different loss function, a concept which is elaborated on in more detail in Section 4.3.3. The architecture of the network is further visualized in Figure 4.2.

There also exists an additional option for encoding the input and output values of the *TSDF* prediction branch. In this alternative encoding, all of the *TSDF* values are binned into a specified number of possible values. This means that also the number of filters of the final convolutional kernel has to be equal to the number of possible values. Hence, the loss function needs to be modified as well, which is described in Section 4.3.3. The predicted *TSDF* values for each voxel during inference are drawn from the probability distribution defined by the probabilities in the prediction result from all possible values.

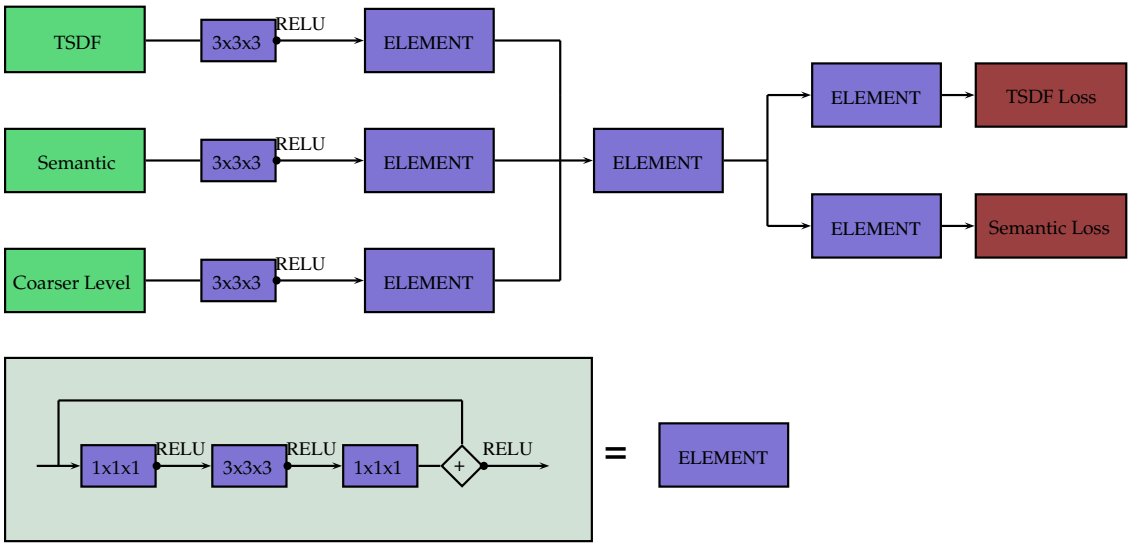


Figure 4.2: Visualization of the network architecture, which is extended from ScanComplete [13]. It can be seen that the network splits into two branches for with separate *TSDF* and semantic prediction loss functions. The basic element of the network is used repeatedly after the initial input data is convolved using a $3 \times 3 \times 3$ kernel. Furthermore semantic information is used as an input to the network as well as *TSDF* input and inputs of the coarser hierarchy level.

4.3.3 Loss Functions

The branching structure of the network architecture, which was proposed in ScanComplete [13], allows for filling in missing values in the *TSDF* function, as well as predicting semantic labels for all voxels present in the voxel grid. In order to achieve both tasks, there are two different branches with separate loss functions as explained in Section 4.3.2. Because of the autoregressive nature of the architecture, the loss functions are applied for

each of the voxel groups extracted from the input voxel grid.

The loss function for the *TSDF* prediction branch is an absolute error function which is defined as follows:

$$E_{\text{tsdf}} = \frac{1}{G} \sum_{g}^{G} \sum_{v}^{V} |\text{TSDF}_{\text{pred}}(v) - \text{TSDF}_{\text{gt}}(v)| \tag{4.4}$$

In the equation above, the number of voxel groups is represented by G and the number of voxels within that group is denoted by V. The network output for a voxel v is obtained through the function $\text{TSDF}_{\text{pred}}(v)$, while the value of the ground truth is obtained in an analogous way via $\text{TSDF}_{\text{gt}}(v)$.

For the semantic prediction branch, a cross entropy error function is used after applying a softmax activation function to the convolutional result of this branch. In order to be able to apply the cross entropy error, a one hot encoding is performed on the ground truth semantic labels for each voxel within a group. This means that for a given ground truth label, a vector of C zeros is created and a single value of 1 is set for the vector on the position that corresponds to the numerical value of the assigned label. The cross entropy error function is defined as follows:

$$E_{\text{semantic}} = \frac{1}{G} \sum_{g}^{G} \sum_{v}^{V} \sum_{c}^{C} -\text{SEM}_{\text{gt}}(v, c) \cdot \log(\text{SEM}_{\text{pred}}(v, c)) \tag{4.5}$$

where $\text{SEM}_{\text{pred}}(v, c)$ provides the semantic prediction for a given voxel v and class c and $\text{SEM}_{\text{gt}}(v, c)$ provides the ground truth label for a given voxel v and class c acquired via one hot encoding. In order to balance the effect of both loss functions on the parts of the network where the two branches have converged, the overall semantic loss is multiplied with a scalar factor i.e. $0.1 \cdot E_{\text{semantic}}$.

Furthermore, in case the alternative encoding for the predicted output *TSDF* values is used, the loss function for the *TSDF* branch is the cross entropy error as well, with the number of classes replaced by the number of possible values.

4.3.4 Training Procedure

Within the training procedure, the ADAM optimizer was used to update the weights, which has been proposed by Kingma and Ba [29]. This optimizer utilizes adaptive gradient descent [14], where all of the parameters, in this case weights and biases, are updated using separate learning rates. Furthermore, it incorporates RMSProp [43] [30], which scales the learning rates using the magnitude of previously computed gradients. The ADAM optimizer also incorporates an input learning rate into the parameter update for a given parameter θ, which is formulated as follows for training step n, according to the

conventions in the Tensorflow [1] implementation:

$$\begin{aligned} g(n) &= \frac{\partial E(\theta(n-1))}{\partial \theta} \\ l(n) &= \eta(n) \cdot \frac{\sqrt{1-\beta_2^n}}{1-\beta_1^n} \\ m(n) &= \beta_1 \cdot m(n-1) + (1-\beta_1) \cdot g(n) \\ v(n) &= \beta_2 \cdot v(n-1) + (1-\beta_2) \cdot g(n) \cdot g(n) \\ \theta(n) &= \theta(n-1) - l(n) \cdot \frac{m(n)}{\sqrt{v(n)+\epsilon}} \end{aligned} \tag{4.6}$$

During training the parameters are set to the following values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \cdot 10^{-9}$. The used input learning rate $\eta(n)$ is updated using an exponential decay which is performed every S steps, as specified in Tensorflow [1]. This update is effective until a specified minimum rate is reached. The learning rate for a given training step n is computed from an initial learning rate η_0 as follows:

$$\eta(n) = \eta_0 \cdot \lambda^{\frac{n}{S}} \tag{4.7}$$

where λ parameterizes the rate of the decay. The parameters are set as follows: $\eta_0 = 0.001$, $\lambda = 0.92$ and $S = 5000$, which is equal to the settings suggested by ScanComplete [13].

Training is performed using batches of eight input/target pairs. In order to provide some randomization with regards to the input batches used they are continuously shuffled in a queue during training.

4.3.5 Extensions

In this section we will discuss further modifications and additions to the network, which will be evaluated quantitatively in Chapter 6.

4.3.5.1 Alternative Encoding of Semantic Input Information

As already described in Section 4.3, we use semantic information originating from 2D semantic labels as input information. The semantic input information is first transformed from class labels $C \in \{0, .., 13\}$ to values $c \in [-1, 1]$ using $c = \frac{C}{|C|} \cdot 2 - 1$. This means that the semantic information is encoded in the range $[-1, 1]$ within a single channel. This is also done for the semantic input of the coarser hierarchy level. After this encoding the semantic information is convolved with $3 \times 3 \times 3$ kernels and subsequently with a basic convolutional element of the network, the structure of which is described in Section 4.3.2.

An alternative encoding for the semantic input has also been explored. For this variant the semantic information is encoded in four channels comparable to [18]. The encoding is the binary representation of the decimal class label for each voxel. Because there are 14 classes a binary representation using 4 bits is sufficient. This alternative encoding should give the network more implicit differentiation between class data. Note that we provide this alternative encoding as additional channels to the channel containing values in range $[-1, 1]$.

4.3.5.2 Flipped TSDF Function Values

Before being processed by the network, the values of the truncated signed distance function for the input data and the distance function for the ground truth data are first transformed as follows based on the original proposal of ScanComplete [13]. The distance function for the ground truth data does not include a sign as elaborated on in Section 5.2.4.

$$\begin{aligned} \mathrm{TSDF}_{\mathrm{input}}(v) &= 2 \cdot |\mathrm{TSDF}_{\mathrm{raw\ input}}(v)| - 1 \\ \mathrm{TSDF}_{\mathrm{gt}}(v) &= 2 \cdot |\mathrm{TSDF}_{\mathrm{raw\ gt}}(v)| - 1 \end{aligned} \tag{4.8}$$

The value ranges after the conversion has been applied are shown in Figure 4.3.

In general, the values for $\mathrm{TSDF}_{\mathrm{raw}}(v)$ are clipped to be in range $[-1, 1]$ and $[0, 1]$ for the signed distance function and ground truth distance function respectively. The truncation is performed in such a way that the range $(0, 1)$ is encoded within a maximum of three voxels. This means that the transformed *TSDF* function will be symmetrical with negative values near the surface which become larger and are eventually truncated to the value one farther away from the surface.

We experimented with another encoding of the input *TSDF* values, where the original distance function is flipped. As a result the inference performance of the network should be improved according to [47] [18]. This means that for voxels close to the actual surfaces of the given geometry, values for the distance function will be close to 1 and -1 respectively, while values farther away from surfaces will be close to 0 and eventually truncated to 0. The flipped *TSDF* is computed from standard $\mathrm{TSDF}_{\mathrm{raw}}(v)$ values as follows:

$$\mathrm{TSDF}_{\mathrm{flipped}}(v) = -(\mathrm{TSDF}_{\mathrm{raw}}(v) - 1 \cdot \mathrm{sgn}(\mathrm{TSDF}_{\mathrm{raw}}(v))) \tag{4.9}$$

In Equation 4.9 the sgn operator returns the sign of a given value. The value ranges after the *TSDF* has been flipped are shown in Figure 4.4.

4.3.5.3 Dilated Convolutions

In order to test the effects of a larger receptive field during network processing dilated convolutions [56] with a dilation of 3 across all dimensions are used in the two basic

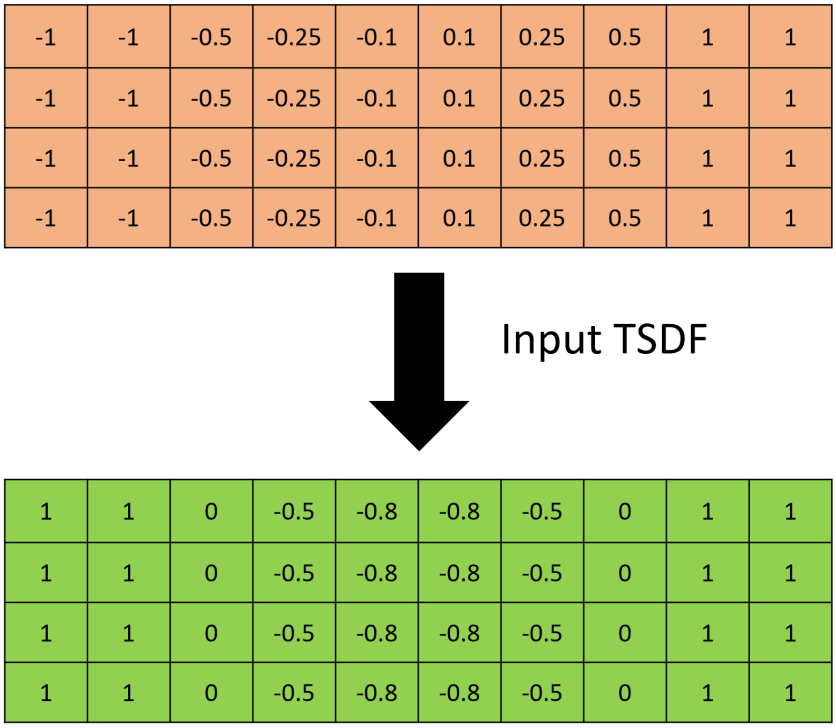


Figure 4.3: Visualization of the $\text{TSDF}_{\text{input}}(v)$ value ranges. The raw *TSDF* values are close to zero at the surface and then either increase to a maximum of 1 or decrease to a maximum of -1. This is converted to an internal representation in ScanComplete [13] where the values at the surface are close to -1 and then increase up to a value of 1 the further away the voxel is from the surface.

convolutional elements, which are applied after convolving the input data. The main idea behind dilated convolutions [56] is to introduce a spacing between the elements of the filter kernel (dilate them). This increases the receptive field and as such, it makes it possible for the network to take more of the global context into account, while not having to perform a sub sampling of the data, as the convolution is changed.

4.3.5.4 Synthetic Noise in Training Set

We use semantic segmentation images for generating additional semantic input information, which can be noisy when generated with DeepLabV3 [7] (with parameters/weights from an already trained network using ade20k [62] [63]). In order to deal with noisy semantic segmentation images, when evaluating on real world data, we inject synthetic noise into our input semantic data during training. This gives the network the ability to deal with misclassified voxels in the input data. The noise is injected by replacing input voxel semantic labels at random locations with random new labels. For each voxel in the grid there is a 5 percent chance to be replaced. The actual voxels are determined by drawing

Figure 4.4: Visualization of the $\text{TSDF}_{\text{flipped}}(v)$ value ranges. The raw *TSDF* values are close to zero at the surface and then either increase to a maximum of 1 or decrease to a maximum of -1. The flipped *TSDF* values are inverted such that the values are close to zero further away from the surface and close to -1 and 1 near the surface.

from a uniform distribution. Once a voxel has been chosen the respective class label is replaced with a random new label, which is also drawn from a uniform distribution.

4.4 Summary

In this chapter, we introduced the neural network architecture of the proposed approach, which is extended from ScanComplete [13]. The architecture makes use of additional 3D semantic input, while having the option to use dilated convolutions on one hierarchy level as well as an alternative semantic encoding. Furthermore, we provided the option to augment the training data with synthetic noise or use a flipped representation of the input *TSDF*. In addition to the training procedure, we have also elaborated on the L1 error function for geometric prediction and the cross entropy error function for semantic prediction.

5 Data Generation

Contents

In this chapter, we first give a general overview of the proposed data generation pipeline. We then focus on the generation of the training data used to estimate the network parameters during the training procedure. Furthermore we elaborate on the data used to quantitatively evaluate the proposed system.

5.1 Data Generation Pipeline

In this section we give a high level overview of the performed processing steps of the proposed pipeline and elaborate on software frameworks used to for generating the training and testing data.

5.1.1 Pipeline Overview

In the following sections we will describe the base procedure of acquiring semantic information and encoding the *TSDF* values in a voxel grid, for both real world and synthetic scenes and elaborate on further processing performed on the input data once it has been converted into voxel grid form.

5.1.1.1 Synthetic Data

We decided to train on synthetic data, because ground truth information which is complete is hard to acquire from real-world environments. An option would be to use a laser

scanner, however occlusions still result in an incomplete model in some scenarios. Moreover acquiring semantic ground truth information from real-world environments represents another challenge.

For synthetic data, images containing semantic segmentations are acquired from multiple camera poses in a virtual scene provided by the SUNCG dataset [47]. This dataset consists mostly of indoor environments with small scale outdoor areas present in some scenarios. The dataset contains semantic information for each of the objects present in a given scene and thus allows for the generation of semantically segmented images. In addition to semantically segmented images, images containing depth information are acquired. After being loaded using OpenCV [5], these two input image sets, combined with the camera poses from which the images have been captured, are then used by a modified version of Open3D [65] to generate a truncated signed distance function of the scene encoded in a voxel grid as well as semantic labels encoded in a voxel grid as described in Section 4.2.1.1. In addition to semantically segmented images and depth images, RGB images of the synthetic scene can also be generated as visualized in Figure 5.2. This data is used as an input for the *CNN* implemented in Tensorflow [1], which has been trained to complete the given *TSDF* values and returns the completed data in voxel grid form as well. One can also enable the option to infer semantic labels for each of the voxels in the grid. This process is also shown in Figure 5.1.

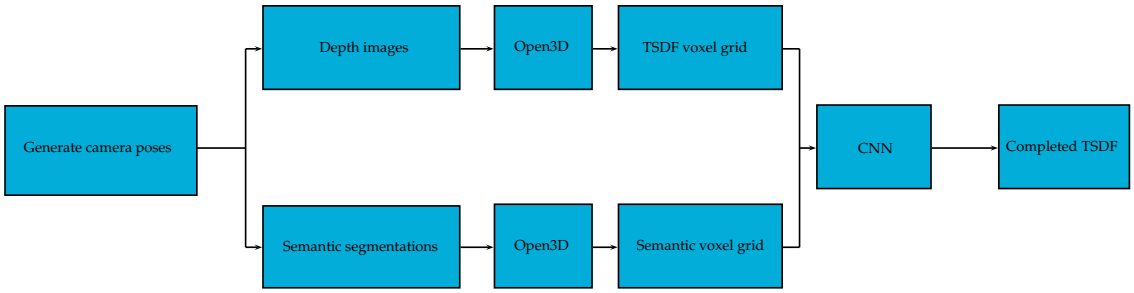
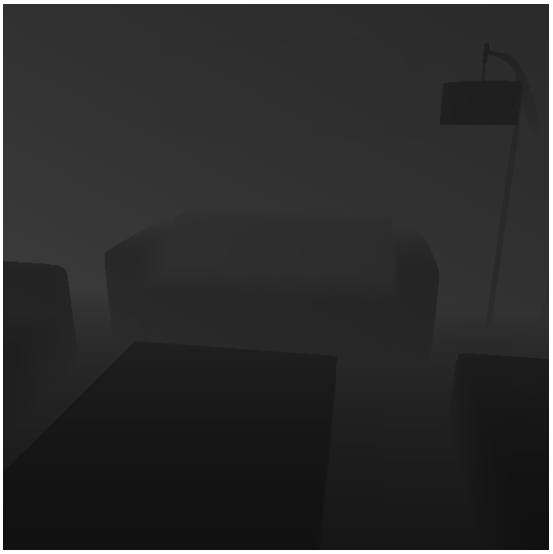


Figure 5.1: Visualization of the core pipeline for synthetic data.

(a) RGB image



(b) Depth image

(c) Semantically segmented image

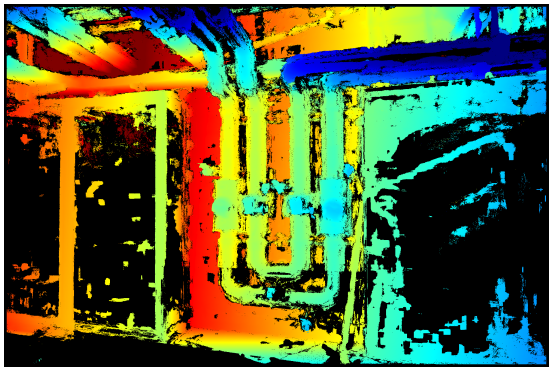
Figure 5.2: Visualization of different image types for synthetic data. The color coding for the semantic classes is also visualized in Section 6.3.

5.1.1.2 Real-World Data

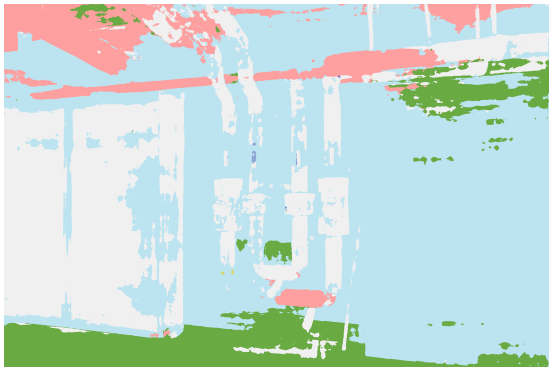
When working on real world data from the ETH3D Benchmark [45], we use the given camera poses included in the training datasets of the benchmark in a dense *MVS* step performed with COLMAP [44]. For evaluation on non-benchmark data the poses would first have to be computed using an *SFM* pipeline, however, in order to achieve parity during our evaluation we use the provided poses. This step yields depth maps for each

individual reference view. The DeepLabv3 [7] *CNN* (with parameters/weights from an already trained network using ade20k [62] [63]) is used to acquire semantic segmentation images from the RGB image input sequence. The different image types are visualized in Figure 5.3. Analogous to synthetic data, the images are loaded for further processing using OpenCV [5]. Open3D [65] is used to generate voxel grid representations of the semantic information, as well as the geometric information. This process is described in more detail in Section 4.2.1.1 The source of input for these tasks are the semantic segmentation images and depth maps acquired earlier. This data is then processed using the *CNN* implementation in Tensorflow [1], which yields a voxel grid with completed *TSDF* values and optionally semantic labels for each of the voxels in the grid. We have also visualized the pipeline for real world data in Figure 5.4.

(a) RGB image



(b) Depth image



(c) Semantically segmented image

Figure 5.3: Visualization of different image types for real-world data. The color coding for the semantic classes is also visualized in Section 6.3.

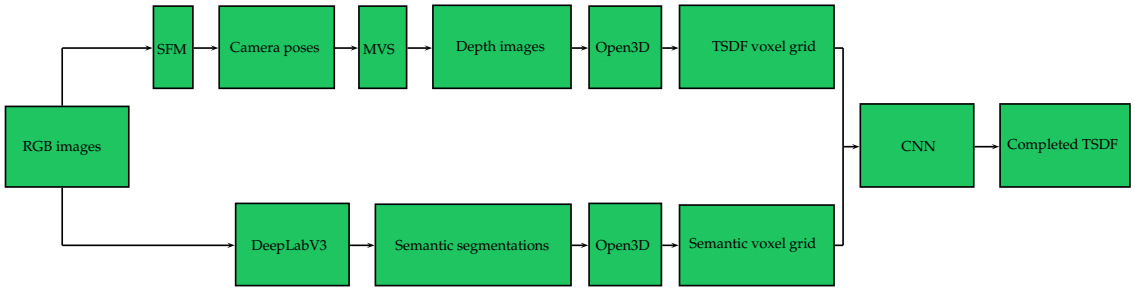


Figure 5.4: Visualization of the core pipeline for real-world data.

5.2 Training Data Generation

This section is dedicated to the procedure of generating training data from the synthetic SUNCG dataset [47]. The process of generating training data using the SUNCG dataset [47] involves three core steps:

- Generating virtual camera poses and extracting images
- Integrating the acquired information into a voxel grid
- Generating ground truth data

In the following sections we will focus on each of these steps individually.

5.2.1 SUNCG Dataset

In general, the SUNCG [47] dataset is organized in a hierarchical manner. The data is organized via a multitude of $N = 45622$ different house scenarios, which largely focus on indoor environments, however there are also outdoor areas and objects present in some scenes. Each of these houses is defined through a *.json*-File. This file splits the house into several basic structural elements:

- levels
- rooms
- objects

The structure is also shown in Figure 5.5. Additionally, it is possible to retrieve the semantic classes for each object through a defined mapping. In this mapping objects are individually assigned to one of the 90 semantic classes, that are translated into one of the 14 labels described in Section 4.1.

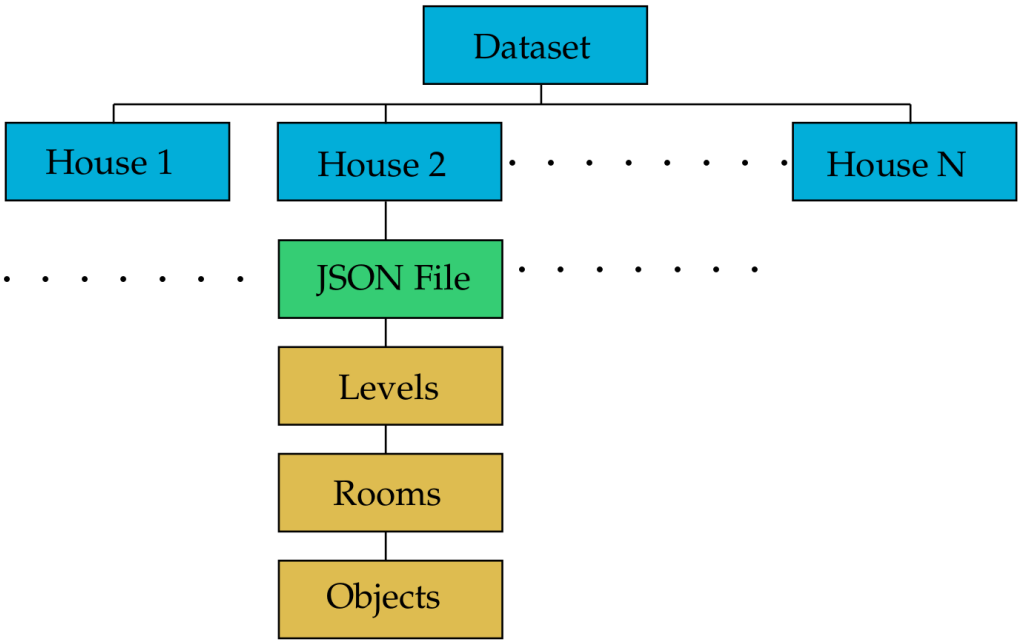


Figure 5.5: Structure of the SUNCG dataset [47].

5.2.2 Generating Virtual Camera Poses and Extracting Images

The virtual camera poses were generated using the SUNCG toolbox [47], which is provided alongside the dataset. Utilizing the functionality implemented in the SUNCG toolbox [47], these poses can be generated using different organizational elements of the dataset as an anchor point. We choose to create poses on a per room and a per object basis. We now elaborate on how the poses are generated using the toolbox [47].

Room based poses are created by iterating through a defined number of camera angles inside the room bounding box. For each of the camera angles, multiple positions are sampled. A score is computed for poses created with these positions, which defines how well the current angle and position capture all of the objects present in the room. This score is calculated by counting the number of pixels each object contained within the image occupies, in relation to the total number of pixels displayed on the image. It is defined that more than one percent of the overall pixels in the image have to be assigned to an object before the score is computed for this specific object and the object is seen as part of the current camera view. If this is the case the score for this object is computed as:

$$S(o) = \log\left(\frac{P(o)}{0.01 \cdot W \cdot H}\right) \tag{5.1}$$

where $P(o)$ is the number of pixels assigned to object o and W, H are the image

width and height respectively. The score is then accumulated over all objects visible in the image. If there are less than 4 objects part of the current camera view the score is set to zero. It can thus be concluded that the overall score for image I is computed as follows: $A_{\text{room}}(I) = \sum_o S(o),\ \forall o \in I$. The camera pose creating the image with the best score I^* is then stored for each angle:

$$I^* = \underset{I}{\operatorname{argmax}}\, A_{\text{room}}(I) \tag{5.2}$$

Object based cameras are created from a camera position near the center of the bounding box of the object. Once again multiple camera angles are sampled and a score for each angle is computed. In this case the score is computed by first creating N points on the object surface in 3D space. The score is the fraction of points visible from the camera perspective compared to the overall number of points on the object N. Thus, the score for a camera I created using a specific angle is computed as $A_{\text{object}}(I) = \frac{V}{N}$. Analogous to the case for entire rooms, the pose for image I^* which yields the highest score with respect to all angles is saved:

$$I^* = \underset{I}{\operatorname{argmax}}\, A_{\text{object}}(I) \tag{5.3}$$

Using the previously generated camera poses, depth images are then captured with the toolbox [47]. The depth data is stored as floating point numbers in a binary file format. Images capturing semantic segmentations are also stored, where the semantic labels are computed from the previously established mapping elaborated on in Section 5.2.1.

In order to increase sparsity when integrating the depth and semantic data into a voxel grid in the next processing step, object cameras are randomly dropped with a probability of 50 percent. Furthermore, there is a 50 percent chance that a camera angled towards the ceiling is created per object camera, because evaluations showed that these areas were often not covered in the reconstructions and the network would not be able to infer meaningful results in these regions.

5.2.3 Integrating Information into Voxel Grid for Training

The previously acquired camera poses, depth and semantic segmentation images are now used as an input for an extended version of Open3D [65], which allows for integrating semantic information into a three dimensional voxel grid as elaborated on in Section 4.2.1.1.

5.2.4 Generating Ground Truth Data

For generating ground truth *TSDF* and semantic data encoded in a voxel grid, we first use the SUNCG toolbox [47] to convert the scene described by a *.json*-File into an *.obj*-File.

During this process we additionally save the semantic class information for each of the stored triangles. As a next step, we extract a signed distance function from the given *.obj*-File using the SDFGen Tool by Christopher Batty [9]. This tool creates a signed distance function encoded in a voxel grid with a defined voxel size. To achieve this task, it calculates the normal distance to all voxels within a defined range for each triangle and stores it in the voxel grid. If subsequent triangles yield a smaller distance for a given voxel, the current distance stored in the voxel grid for this voxel is replaced by the smaller distance. Because per triangle semantic information is also saved, a second voxel grid can be created where the semantic class of the triangle with the closest distance is stored for each voxel. In order to align the ground truth data with the voxel data from the reconstruction, the origin of the voxel grid created by SDFGen [9] is chosen such that it coincides with the origin of the voxel grid created by Open3D. The size of the ground truth voxel grid is also defined such that it matches with the grid from the reconstruction.

5.2.5 Extracting Training Sub-Volumes

After having acquired an input and ground truth voxel grid of the same size, we extract smaller sub-grids out of the complete grids for training. The sizes for the sub-grids are chosen depending on the sizes of the individual voxels within the grid. In terms of voxel sizes and sub-grid sizes for the different hierarchy levels, we have used the same conventions defined in ScanComplete [13]. The coarsest level uses a voxel size of 0.18 and a sub-volume size of 16, the next level uses 0.09 and 32, while the final level uses 0.045 and 64. It should be noted that the SUNCG dataset [47] defines distances on a metric scale. Furthermore, we check how many voxels of a given extracted ground truth and input sub-volume contain non-truncated signed distance function values. This percentage needs to be higher than one percent for both the ground truth and input data, in order for it to be included in the training set.

5.3 Evaluation Data Generation

In this section we will focus on the process of generating data needed to quantitatively evaluate the implemented network modifications on real world and synthetic scenes.

5.3.1 Synthetic Scenes

For synthetic scenes the process of generating the evaluation data is identical to the steps elaborated on in Section 5.2 excluding the last step of extracting the sub-volumes. In this step the entire voxel grid is extracted when generating evaluation data. The evaluation dataset consists of 40 house scenarios from SUNCG [47]. They contain indoor enviroments of different sizes, which contain typical objects found inside homes such as tables, chairs and sofas. They also contain different room scenarios such as living rooms, kitchens and bathrooms as visualized in Figure 5.6.

Figure 5.6: Sample RGB images for synthetic scene environments.

5.3.2 Real World Scenes

The scenes used for evaluating on real-world image-based reconstructions from the ETH3D benchmark [45] are four indoor environments. These are denoted as *pipes*, which consists of an underground hallway with pipe-like structures, where the walls do not contain a lot of texture, such that it is difficult to generate complete depth maps using image-based algorithms. The *office* dataset contains a small office room also with white walls which do not contain a lot of texture, while the *relief* dataset offers a bigger hall like structure with columns. The *terrains* dataset consists of a hallway with a reflective floor, which is also difficult to reconstruct. We have visualized a few sample RGB input images in Figure 5.7.

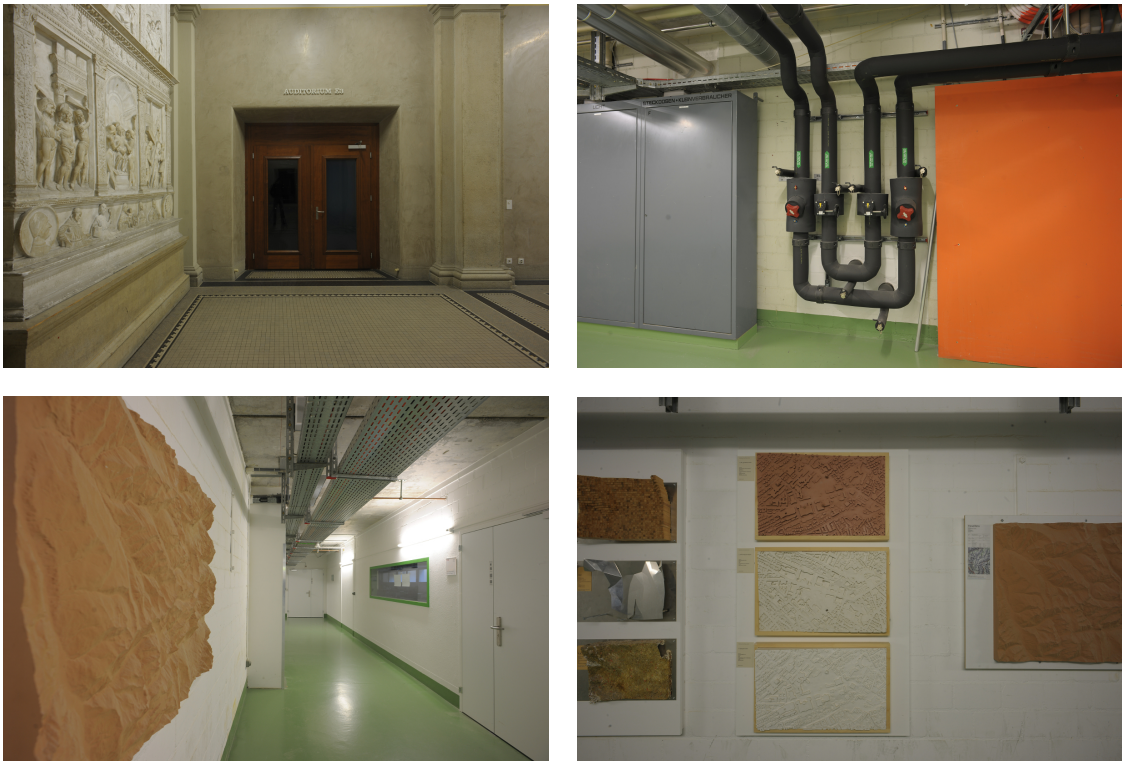


Figure 5.7: Sample RGB images for real-world scene environments.

5.4 Summary

In this chapter, we introduced our pipelines for generating training data for synthetic scenes, as well as evaluation data for synthetic and real world scenes. We elaborated on the process of creating incomplete 3D representations of scenes from the SUNCG [47] dataset and how we generated 3D semantic information from 2D semantically segmented images. Furthermore, we discussed the used evaluation sets for both real world and synthetic data, which we used to quantitatively and qualitatively evaluate the proposed approach in Chapter 6.

6 Experiments

Contents

In this chapter, we first elaborate on our evaluation parameters and metrics and subsequently present the results of the experiments performed on the data generated with the conventions described in Chapter 5. Furthermore, we discuss the implications of the presented results in addition to possible improvements for future work.

6.1 Evaluation Parameters and Metrics

In the next sections we give an overview of the used evaluation metrics and parameters. We first present the used metrics and then elaborate on the evaluation dataset as well as the parameters used when training the network. Furthermore we provide additional explanations for the included visualizations.

6.1.1 L1 Error

The evaluation metric used for evaluating on synthetic data is an error metric measuring the absolute difference between the resulting distance function values and the ground truth, which we denote L1 error:

$$\mathrm{L1} = \frac{1}{|X|} \sum_{x}^{X} |\mathrm{TSDF}_{\mathrm{gt}}(x) - \mathrm{TSDF}_{\mathrm{prediction}}(x)| \tag{6.1}$$

In the above equation X represents the set of all voxels in the grid and $|X|$ is the number of voxels all present in the grid. The variable x enumerates all voxel locations in the set X, while $\text{TSDF}_{\text{gt}}(x)$ and $\text{TSDF}_{\text{prediction}}(x)$ yield the ground truth *TSDF* values and predicted output *TSDF* values from the network for voxel x. Note that while the distance function values here will always have a positive sign, we still refer to the values as *TSDF* values. This error is also evaluated on a different set of voxels within the grid which, represents a different underlying geometric context. This set O is the occupied set, which denotes all voxels in the ground truth voxel grid which contain a non truncated distance function value:

$$\text{L1}_{\text{occupied}} = \frac{1}{|O|} \sum_{o}^{O} |\text{TSDF}_{\text{gt}}(o) - \text{TSDF}_{\text{prediction}}(o)|\,,\ o \in O \subseteq X :\ \text{TSDF}_{\text{gt}}(o) < 1 \tag{6.2}$$

The value $|O|$ represents the number of voxels in the set O and o enumerates all voxels in this set. The functions $\text{TSDF}_{\text{gt}}(o)$ and $\text{TSDF}_{\text{prediction}}(o)$ yield the respective ground truth and predicted values for voxel location o.

6.1.2 Semantic Accuracy

We also provide a semantic accuracy measure, which yields the fraction of correctly predicted semantic classes for each voxel present in the completed voxel grid, when compared to the ground truth:

$$\text{Acc} = \frac{|C|}{|X|},\ C \subseteq X :\ \text{class}_{\text{pred}}(c) = \text{class}_{\text{gt}}(c),\ \forall c \in C \tag{6.3}$$

In the equation above the set C represents all voxels where the predicted label is equal to the ground truth label, while $|C|$ represents the number of voxels in C and $|X|$ represents the number of voxels in X, which is the set of all voxels. The functions $\text{class}_{\text{pred}}(c)$ and $\text{class}_{\text{gt}}(c)$ yield the predicted and ground truth classes for voxel location c respectively. Furthermore, similar to the L1 error, we also provide this measure for occupied voxels O, which are non truncated voxels in the ground truth:

$$\text{Acc}_{\text{occupied}} = \frac{|C|}{|O|},\ C \subseteq O : \text{class}_{\text{pred}}(c) = \text{class}_{\text{gt}}(c), \forall c \in C,\ o \in O \subseteq X : \text{TSDF}_{\text{gt}}(o) < 1 \tag{6.4}$$

6.1.3 F1 Score

The error metric, which we use to measure performance on real world data is an F1 score [45] calculated as follows from an accuracy measure A and a completeness measure C:

$$\mathrm{F1} = \frac{2 \cdot A \cdot C}{A + C} \tag{6.5}$$

The score itself is computed using the provided tool included in the ETH 3D benchmark [45]. It provides an overall relation between the accuracy measure, which defines how many points of the point cloud computed from the completed voxel grid, lie within a defined distance from a points of the ground point cloud, when taking the laser scan accuracy into account as proposed by Schöps et.al. [45]. The completeness measure on the other hand, defines how many points of the ground truth point cloud, lie within a defined distance from a given point of the point cloud computed from the completed voxel grid. The distance used for both measures is set to 0.09.

The F1 score metric is evaluated on a ground laser scan point cloud from the ETH 3D benchmark [45] and the completed point cloud, which is generated by creating a point for every voxel location which contains a distance function value less than or equal to 0.15. Because a distance function value of 1 represents three voxels, a distance of 0.15 equates to 0.45 voxels, which is $4.05cm$ in world coordinates. This means that a tolerance value of $9cm$ covers voxels on both sides of a surface.

6.1.4 Evaluation Set and Parameters

The evaluation data itself has been generated as described in Section 5.3. During our evaluation, we enable semantic prediction, which results in improved results when using additional input semantics, as we give the network an initialization for the semantics as an input. Thus, we are also able to benefit from improved input data for the fine grained hierarchy level due to an improved semantic prediction. Furthermore, it is noteworthy, that the scaling factor for semantic prediction has been set to 0.1 for all experiments. We also enable semantic prediction for the ScanComplete [13] approach, however, because we use the small semantic loss scaling of 0.1, the impact on geometric completion when using semantic prediction, which has been described by Dai. et. al [13], is minimized, which we also show in our first experiment on one hierarchy level in Section 6.4.2.1. When experimenting with this parameter for the proposed approach and increasing the factor to 0.5, the results showed no significant impact, hence we did not perform additional experiments with further changes to this parameter.

The computed error metrics are averaged over the 40 evaluation sets for synthetic data, however we also provide graphs showing the detailed results for each of the tested data sets. For all of the sets we remove the extra voxels below the ground level generated by

the volumetric integration before inferencing and add truncated distance function values back after inference, as also performed in [13]. As also done for synthetic data, we remove additionally added voxels below the ground level for the real world evaluation datasets as well.

6.2 Training Parameters

All of the networks were trained using 80000 iterations on an Nvidia RTX 2080 Ti graphics card. The training parameters were set as described in Section 4.3.4. The training set was generated according to Section 5.2 and contains data from 806 ScanComplete scenes yielding 43800 input/target pairs in total.

6.3 Explanation for Visualizations

All visualizations were generated using MeshLab [10] after creating a mesh via the iso-surface library function implemented in MATLAB [51]. The color for each of the faces relates to the L1 error of the underlying voxels using the color scheme shown in Figure 6.1. In the completeness visualizations for the ETH3D benchmark [45], green points represent complete points, while red points represent incomplete points. In the accuracy visualization blue points are unobserved (outside of laser scan range), while red points are inaccurate and green points are accurate. The semantic classes are visualized as point clouds generated for voxels which are occupied. The specific colors assigned to each class are visualized in Figure 6.2.

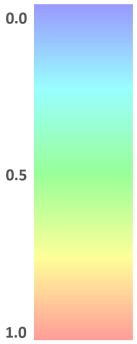


Figure 6.1: Visualization of how the color in the presented geometric outputs relates to the underlying L1 error.

Figure 6.2: Colors assigned to specific classes in visualizations for semantic predictions

6.4 Experimental Results

In the following section, we present our experimental results and provide explanations as well as conclusions for our qualitative findings. We provide a detailed evaluation on synthetic data when using only one hierarchy level and show the benefits of our extensions. Subsequently we provide quantitative results when using two hierarchy levels on synthetic data. Furthermore, we show the effect of our contributions on real world image based reconstructions.

6.4.1 Results on Synthetic Data

The purpose of this section is to provide detailed evaluations on synthetic data from the SUNCG [47] dataset. We will provide detailed results evaluating our contributions to the network architecture.

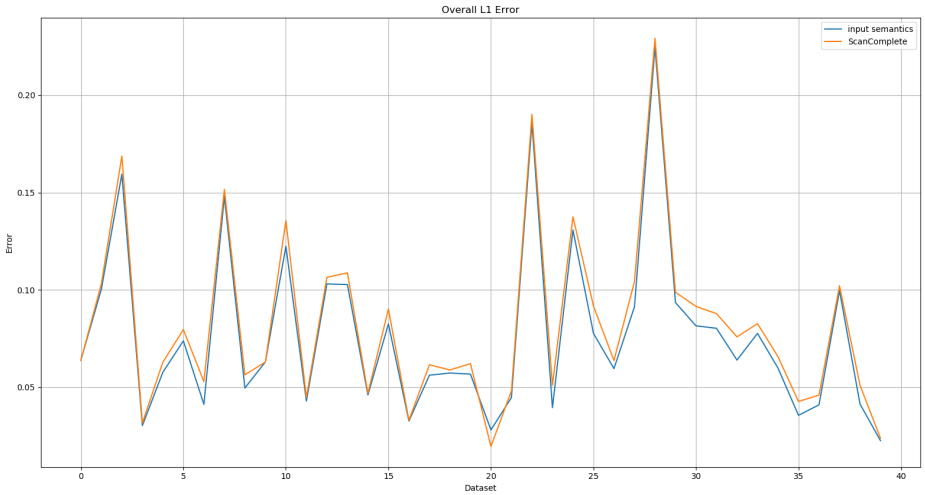
6.4.2 Detailed Evaluation for One Hierarchy Level

This section is dedicated to providing a detailed analysis of the effects the modifications of our proposed network architecture have by comparing the results on the synthetic evaluation set with results from the original architecture proposed in ScanComplete [13] on one hierarchy level. To achieve parity, we have trained the original ScanComplete [13] architecture on the training set we have created according to Section 6.2. In this evaluation, we use one of the two hierarchy levels, namely the one using a voxel size of $0.09m$, which provides a middle ground between the other more coarse (voxel size $0.18m$) and fine grained (voxel size $0.045m$) hierarchy levels.

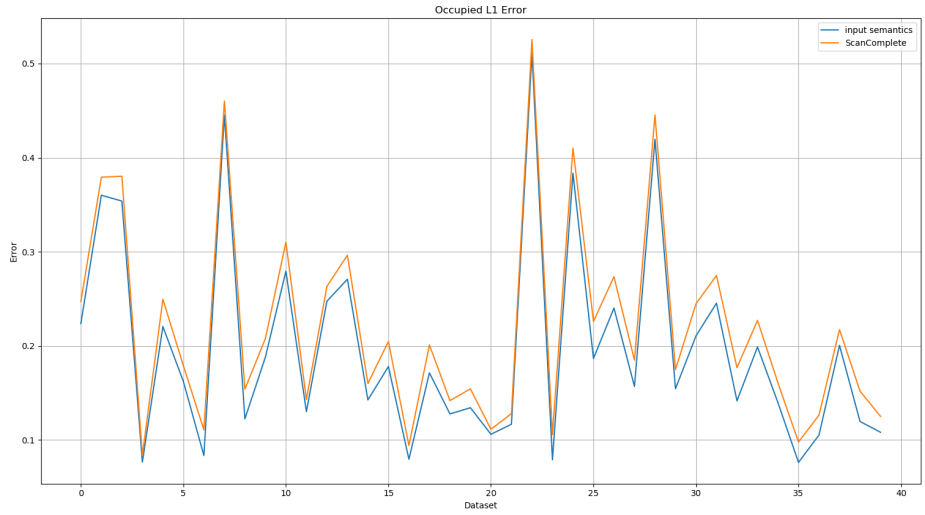
6.4.2.1 Influence of Additional Input Semantics

In this section we evaluate the effect of providing the network with additional input semantics. This extension will be denoted as *input sem.* in the performed experiments. From the evaluation of the L1 error in Table 6.1, when comparing *input sem.* (middle left) and *ScanComplete* [13] (middle right), it can be seen that *input sem.* achieves a lower overall L1 error as well as a lower L1 error on occupied voxels. These are likely the effects of giving the network additional context via input semantics. The differences are also observable in Figure 6.3, where we can see a decrease in error for *input sem.* on most datasets. Furthermore, it can be observed that the overall semantic accuracy increases, as seen in Table 6.1 and Figure 6.4, when comparing lines *input sem.* and *ScanComplete* [13], which means that the semantic initialization the network has been given was incorporated to provide better semantic predictions. An exemplified result for an improvement in the L1 error metric is shown in Figure 6.5, which depicts the geometric results for dataset 35. It can be observed, when comparing the results for *input sem.* and *ScanComplete* [13], that the reconstruction is more complete on the ceiling and partially on the walls. Furthermore, semantic accuracy was also improved as shown in Figure 6.7, when comparing the results for *input sem.* (middle left) and *ScanComplete* [13] (middle right). In the semantic prediction for the ScanComplete [13], approach voxels on the ceiling (red) were misclassified as floor (green) and the prediction contains a lot of noise. When looking at the result were input semantics were provided, this is not the case as the semantic initialization given to the network was expanded. From these observations made in Figure 6.5 and Figure 6.7, it can be seen that the additional semantic input provides useful context to the network. As a frame of reference, we have also provided the input and ground truth geometric and semantic data in Figure 6.5 (top part) and Figure 6.7 (top part), respectively.
An example, which shows a degradation in performance when using input semantics is dataset 20. When looking at the geometric prediction when using input semantics, which is visualized in Figure 6.9 as *input sem.* (bottom left), it can be seen that a lot of clutter was predicted around the couch object, which is also evident in the semantic prediction in Figure 6.10 (bottom left). This is likely because the network which was initialized with semantic information, knows that these objects belong to the furniture class. The network seems to think that the furniture extends to the surrounding area, which is not correct in this case. The result here is likely influenced by the lack of geometry around the couch, which is placed as a separate object outside of the building for this particular scenario. When using no input semantics the network does not use this context as also seen in the noisy semantic prediction of the *ScanComplete* [13] result (bottom right) in Figure 6.10. As an additional point of reference, the geometric and semantic input data and ground truth are provided in Figure 6.9 (top part) and Figure 6.10 (top part). Furthermore, we have also evaluated ScanComplete [13] when not using semantic prediction, however as the scaling factor for the semantic loss is set to 0.1 the results for ScanComplete [13] are not negatively impacted as seen in Table 6.1, when comparing *ScanComplete* [13] with *Scan-*

Complete [13] *no sem. pred.* and so we continue to compare with the *ScanComplete* [13] result using semantic prediction.



Overall L1 error metrics.



L1 error metrics on occupied voxels.

Figure 6.3: L1 error metrics for all 40 evaluation sets. It is observable that the *input sem.* approach, which is using additional input semantics yields lower L1 errors compared to ScanComplete [13] for most datasets.

6.4.2.2 Dilated Convolutions

In this section we evaluate the effects of using dilated convolutions [56] and additional input semantics, as described in Section 4.3.5.3. We denote this adaption as *input sem. + dilation* in our experiments. In Table 6.1, when comparing *input sem. + dilation* with *input sem.*, it can be seen that both of the L1 errors and both the semantic accuracy measures are improved. This is the result of extending the receptive field of the network, such that the filters are able to take into account information from a larger region. When looking at the results for dataset 38 shown in Figure 6.5 and comparing results for *input sem. + dilation* (bottom left) with results for *input sem.* (middle left), it is observable that even more of the ceiling has been completed. Furthermore, the semantic accuracy visualized for *input sem. + dilation* (bottom left) in Figure 6.7 has also improved, as there is less noise present compared to *input sem.* (middle left), which is evident on the front wall of the building.

6.4.2.3 Flipped TSDF

This section is dedicated to the evaluation of the flipped *TSDF* function, which is described in Section 4.3.5.2. We denote this method as *input sem. + flipped TSDF* in our experiments. As seen in Table 6.1, when comparing line *input sem. + flipped TSDF* with line *input sem.*, the effects of using a flipped *TSDF* input function in addition to semantic input provide no significant benefit compared the other tested methods using input semantics. Both the L1 error and semantic accuracy are on the same level as the *input sem.* method, which just uses additional semantic input. When comparing geometric results on dataset 38, denoted as *input sem. + flipped TSDF* (bottom right) and *input sem.* (middle left), in Figure 6.5, there are no significant differences visible. However, the semantic prediction contains a little more noise, specifically in the ceiling area, as seen in Figure 6.7 when comparing results for *input sem. + flipped TSDF* (bottom right) with *input sem.* (middle left).

6.4.2.4 Alternative Encoding for Semantic Input

In this section we evaluate the effects of using an alternative encoding for our semantic input as described in Section 4.3.5.1. It can be seen in Table 6.1, when comparing row *input sem. + encoding* with row *input sem.*, that using an alternative encoding for input semantics increases overall semantic accuracy, compared to just using input semantics. However, both the semantic accuracy metrics still fall slightly behind the variant which uses dilated convolutions. The L1 error does not significantly deviate from the results when using input semantics as well, however it becomes slightly worse. It can be seen in Figure 6.6, when comparing results for *input sem. + encoding* (middle right) and *input sem.* (middle left), that the geometric results for *input sem. + encoding* are more complete on the right side of the ceiling, however there is a small hole on the left side. Furthermore

there are some incorrect predictions for the semantics of the ceiling on the edges of the completed ceiling as seen in Figure 6.8, when looking at results for *input sem. + encoding* (middle right) and *input sem.* (middle left). On the other hand the noise in front of the wall present in *input sem.* is removed when using *input sem. + encoding*.

6.4.2.5 Augmenting Training Data with Synthetic Noise

This section is dedicated to evaluating the effect of augmenting the semantic training data with synthetic noise, while using additional semantic input data, as elaborated on in Section 4.3.5.4. It can be seen in Table 6.1, when comparing line *input sem. + noise* with line *input sem.*, that the overall evaluation metrics show minor improvements, however the metrics on occupied voxels are slightly worse compared to just using input semantics. This is the result of preventing wrong predictions, but being less correct on occupied voxels, which is also evident when looking at the semantic prediction results for *input sem. + noise* (bottom left) and *input sem.* (middle left) in Figure 6.8, as the noise in front of the building was removed for *input sem. + noise*, however the predictions on the ceiling are less correct. In terms of geometric completion, the approach also falls slightly behind as there is a hole present on the ceiling as seen in Figure 6.6 for *input sem. + noise* (bottom left), when comparing the result with *input sem.* (middle left). This means that the synthetic noise added during training prevents minor wrong predictions from being made, which results in only a slight overall improvement on synthetic data, however we will be further investigating the effects of this modification in our evaluation on real world data.

6.4.2.6 Augmenting Training Data with Synthetic Noise in addition to using Dilated Convolutions and the Alternative Semantic Encoding

In this section we evaluate the effects of combining dilated convolutions and the alternative encoding for input semantics, with augmenting the semantic input training data using synthetic noise. Although the alternative encoding yielded slightly worse results in terms of the L1 error metrics, the difference was not significant enough to outweigh the potential benefits of improved semantic predictions when using multiple hierarchy levels. The results for *input sem. + noise + dilation + encoding* in Table 6.1 show an improvement for the semantic accuracy metrics, when compared with any of the other methods. The L1 error on occupied voxels also improves, while the overall L1 error is not improved compared to *input sem. + dilation*. However, it does not deviate significantly from the *input sem. + dilation* result. These improvements could be the effects of a wider receptive field combined with the other two methods, which yielded more minor improvements. The semantic prediction for *input sem. + noise + dilation + encoding* (bottom right) seen Figure 6.8 contains less noise overall compared to *input sem.* (middle left), however it shows some incorrect classifications for floor (green) on the ceiling (red). This shows, that the network is able to prevent small amounts of noise from being propagated further in

the network. In terms of geometric completion, the results are improved as well, which can be seen in Figure 6.6, when comparing *input sem. + noise + dilation + encoding* (bottom right) with *input sem.* (middle left).

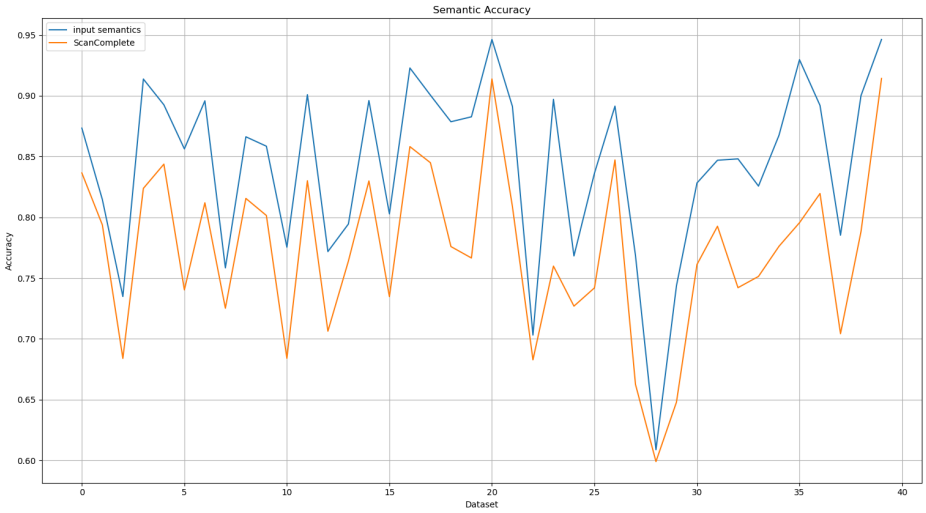
Furthermore these improvements also lead to an increase in geometric completion for objects contained in the synthetic scenes compared to ScanComplete [13], as seen with the chair in Figure 6.11, when comparing the *input sem. + noise + dilation + encoding* (bottom left) with the *ScanComplete* [13] (bottom right) result. This is also likely the result of an increase in overall semantic prediction accuracy, due to the provided semantic input for *input sem. + noise + dilation + encoding* (bottom left), as observed in Figure 6.12.

6.4.2.7 Conclusion

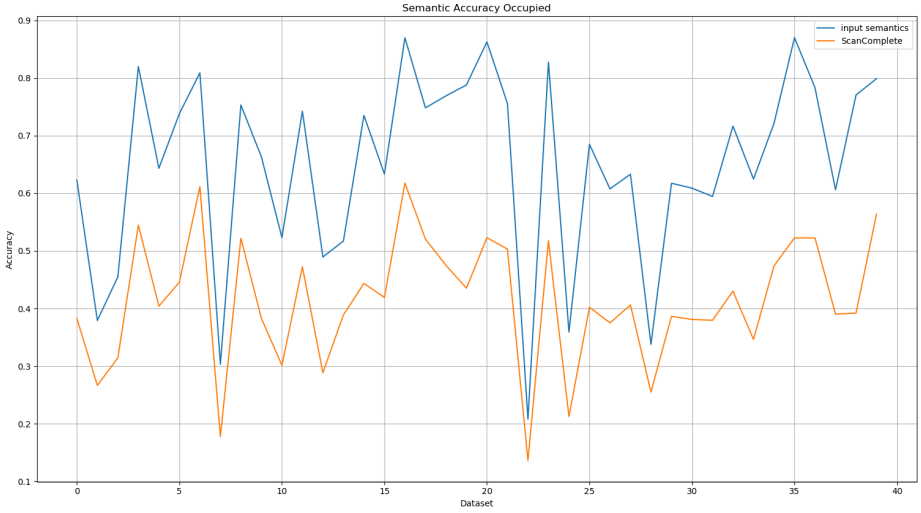
When looking at the overall results in Table 6.1, it can be observed that the *input sem. + noise + dilation + encoding* approach, considering there are only insignificant differences in terms of overall L1 error compared to *input sem. + dilation*, achieves better results than the other methods. This means that we will continue to evaluate the effects of the alternative encoding, dilated convolutions and synthetic noise in our subsequent experiments using two hierarchy levels and real world data.

Method	∅ L1	∅ L1 occ.	∅ Sem. acc.	∅ Sem. acc. occ.
input sem. + noise + dilation + encoding	0.0674	**0.1721**	**0.8611**	**0.6965**
input sem. + noise	0.0762	0.2021	0.8445	0.6246
input sem. + encoding	0.0774	0.2048	0.8529	0.6536
input sem. + flipped *TSDF*	0.0759	0.1982	0.8419	0.6467
input sem. + dilation	**0.0668**	0.1752	0.8600	0.6684
input sem.	0.0767	0.1974	0.8429	0.6498
ScanComplete [13]	0.0821	0.2202	0.7727	0.4134
ScanComplete [13] no sem. prediction	0.0841	0.2205	-	-

Table 6.1: L1 error: lower means better. Semantic accuracy: higher means better. Overview of evaluation metrics for experiments performed on 1 level with synthetic data. In general, the proposed methods outperform both of the *ScanComplete* [13] approaches. It can be seen that *input sem. + noise + dilation + encoding* yields the best performance for every metric except overall L1 error. However the result for this metric does not significantly deviate from the best result with *input sem. + dilation*. Furthermore *input sem. + encoding* and *input sem. + noise* yields improvements for the semantic accuracy metrics. The input *sem. + flipped TSDF* result does not provide a significant advantage compared to *input sem.* The *input sem. + encoding* method improves on the semantic accuracy metrics compared to *input sem.*

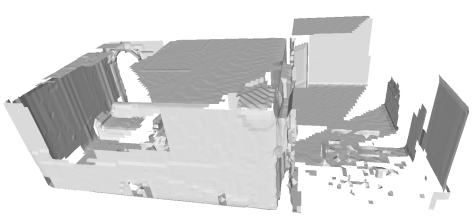


Overall semantic accuracy metrics.

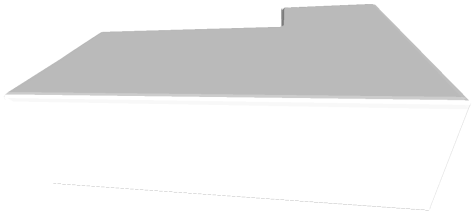


Semantic accuracy metrics on occupied voxels.

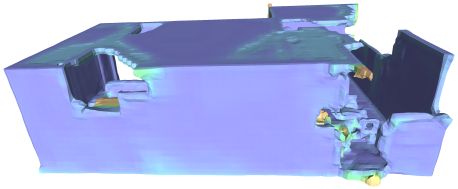
Figure 6.4: Semantic accuracy metrics for all 40 evaluation sets. It can be seen that using a semantic initialization in the *input sem.* approach, improves semantic accuracy as the network can extend this semantic prediction.



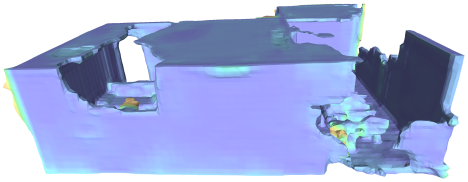
input used



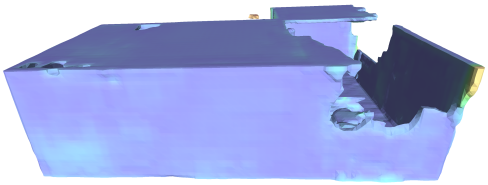
ground truth



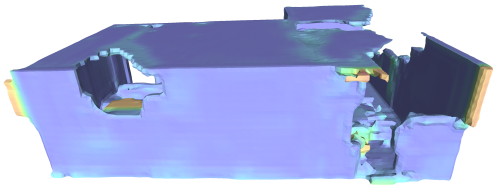
input sem.



ScanComplete [13]

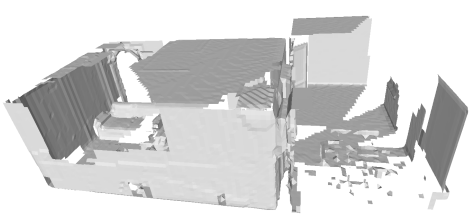


input sem. + dilation



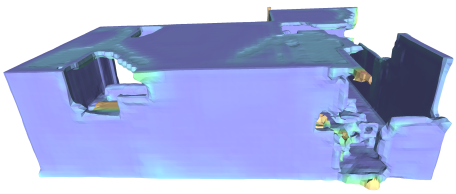
input sem. + flipped *TSDF*

Figure 6.5: Geometric network output for various methods on dataset 38. The geometric input used contains holes on the left and right side of the ceiling as well as the front wall. It can be observed, when comparing the *ScanComplete* [13] result with *input sem.*, that more of the geometry is completed, specifically in the ceiling area on the left side as well as the front wall. Using the *input sem. + dilation.* method, it can be seen that even more of the missing building structure is filled as the network can take more of the global context into account. The *input sem. + flipped TSDF* variant does not provide significant benefits over the *input sem.* method.

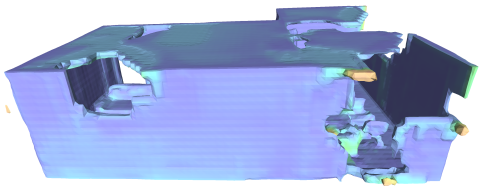


input used

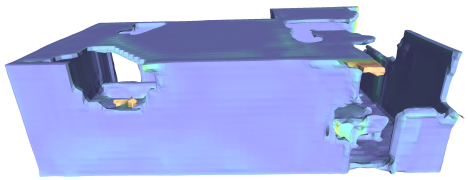
ground truth



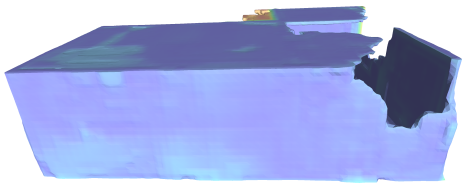
input sem.



input sem. + encoding

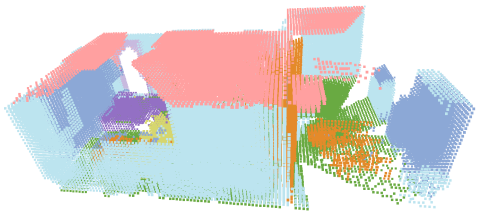


input sem. + noise

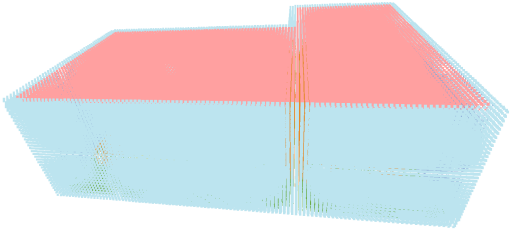


input sem. + noise + dilation + encoding

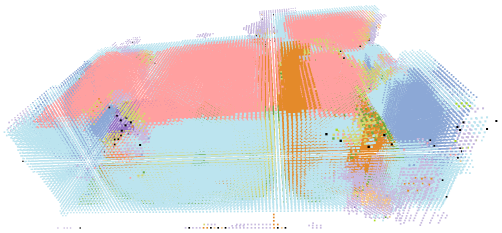
Figure 6.6: Geometric network output for various methods on dataset 38. It can be seen that out of the presented methods *input sem. + noise + dilation + encoding* provides the most complete results, as more of the missing geometry in the ceiling and the hole remaining in other results on the left side of the front wall area of the building is filled compared to *input sem. + encoding* and *input sem. + noise.*



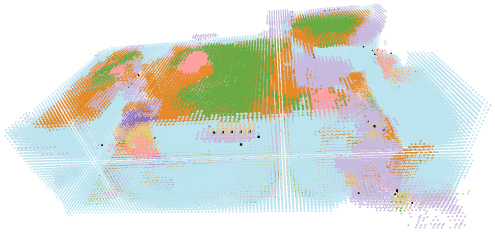
input used



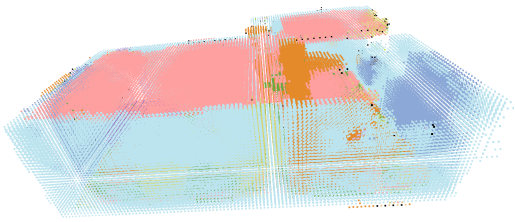
ground truth



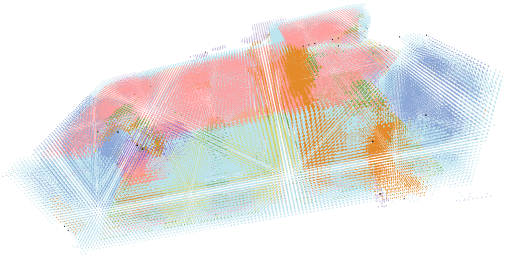
input sem.



ScanComplete [13]



input sem. + dilation



input sem. + flipped *TSDF*

Figure 6.7: Semantic predictions for various methods on dataset 38. Providing an initialization for the semantic prediction for the network to expand upon with the *input sem.* approach provides a better overall result than the original ScanComplete [13]. Furthermore, the method *input sem. + dilation provides* better results which contain less noise, than *input sem.* and *input sem. + flipped TSDF*.

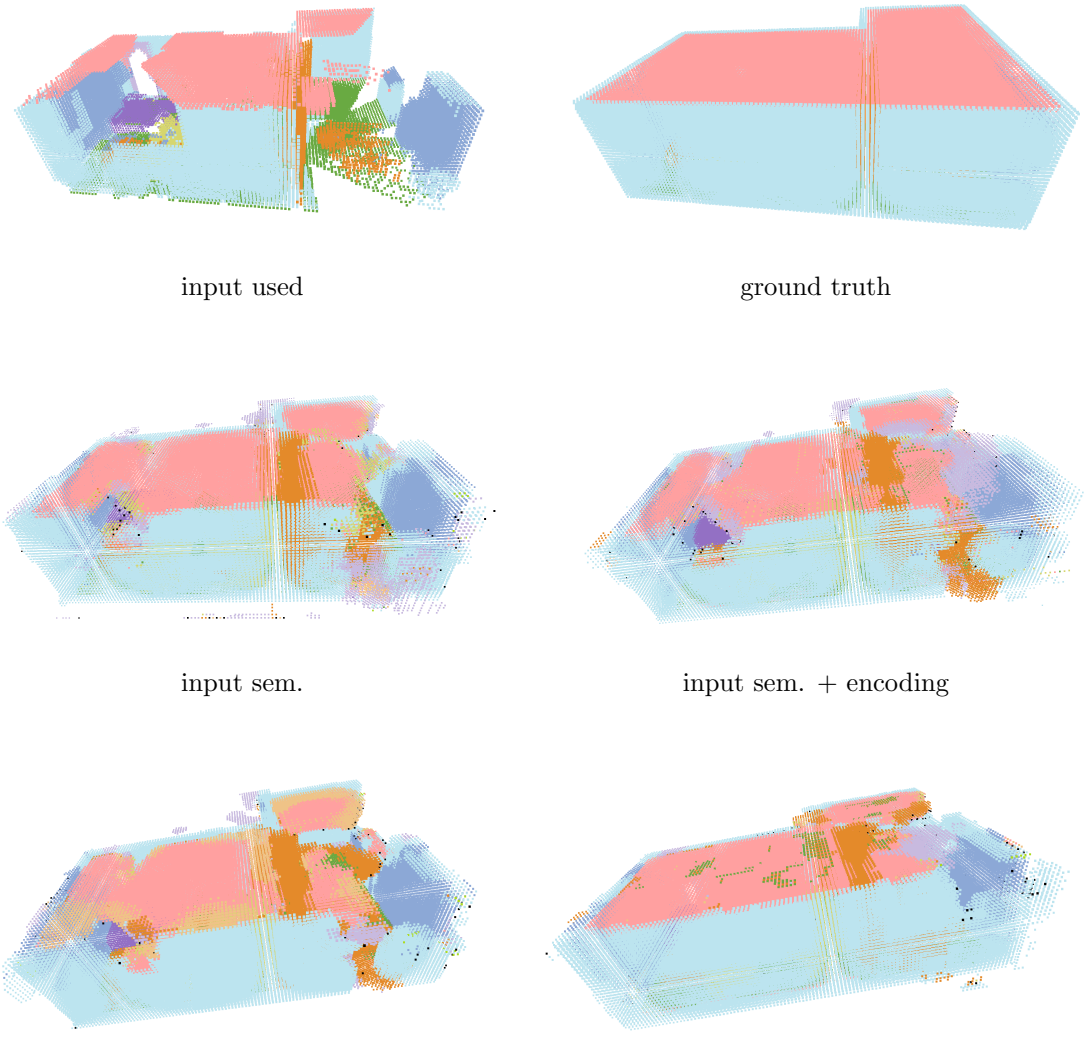
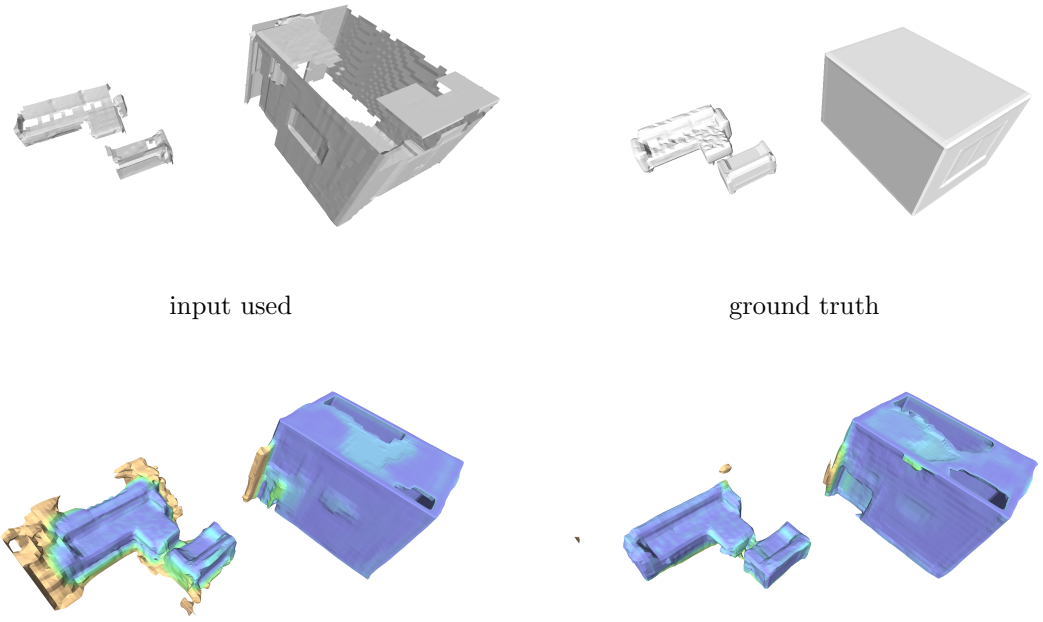


Figure 6.8: Semantic predictions for various methods on dataset 38. It can be seen that the *input sem. + encoding* and *input sem. + noise* methods, while reducing smaller amounts of noise, specifically on the right side of the front wall of the building, overall also add some wrong predictions, specifically in the ceiling area. However in general these modifications still provide improvements in semantic prediction as seen in Table 6.1. When adapting the network architecture to combine these smaller improvements with a larger receptive field with *input sem. + noise + dilation + encoding*, the noise is further reduced and the overall prediction accuracy is increased.



input used ground truth

input sem ScanComplete [13]

Figure 6.9: Example for an increase in L1 error when using *input sem.* for dataset 20. The scene includes a building with missing geometry and a couch area. It can be seen that a lot of clutter is predicted around the couch area. This is likely due to missing geometry in the ground truth around the couch area, which is separated from the building structure.

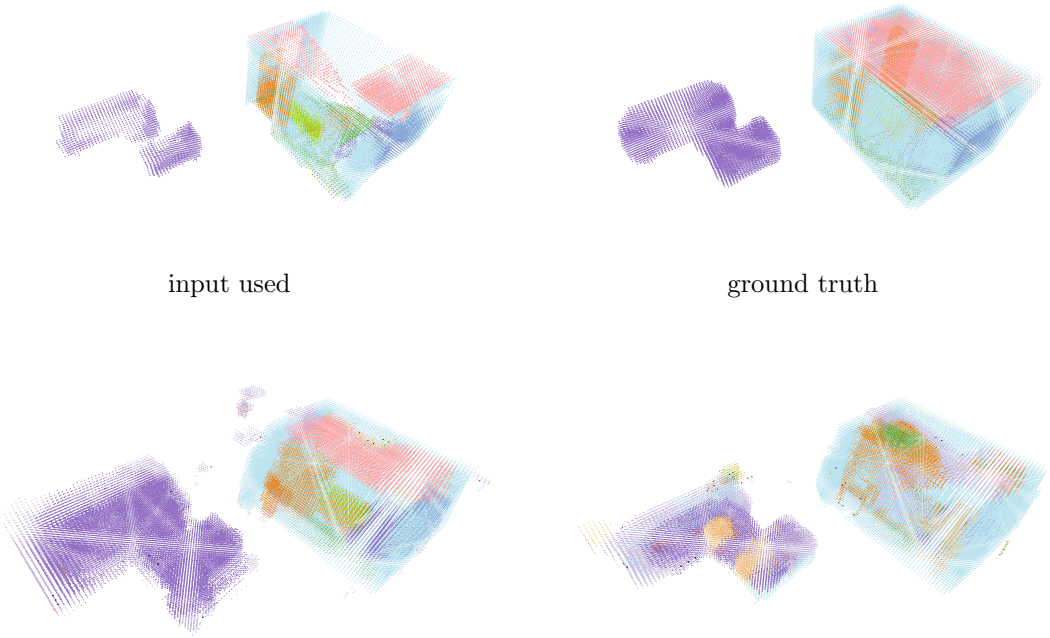


Figure 6.10: Example results for semantic accuracy for dataset 20. It can be observed that the couch area is extended outside of its original bounds when using *input sem*, in this case likely due to missing geometry in the ground truth in this area.

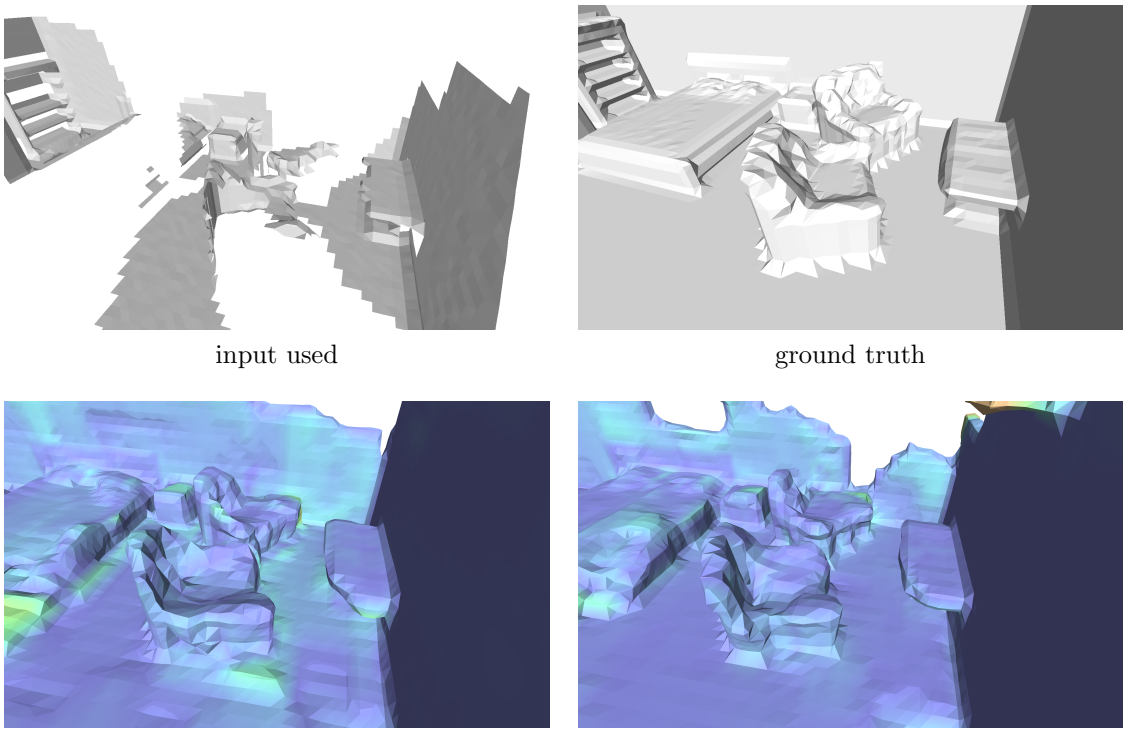
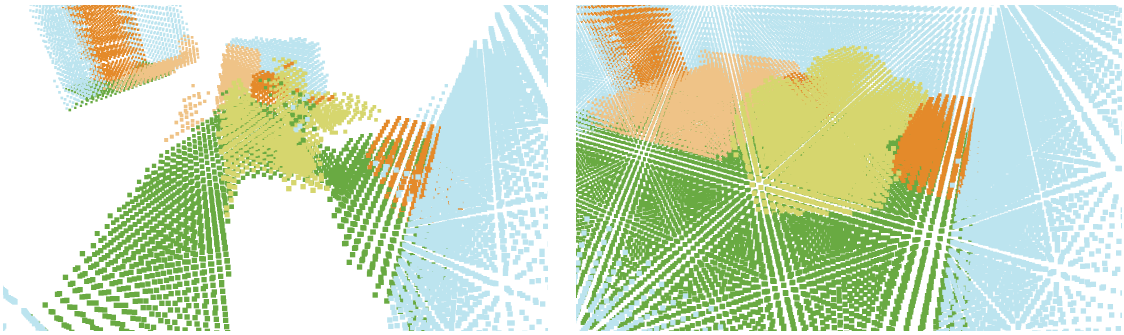
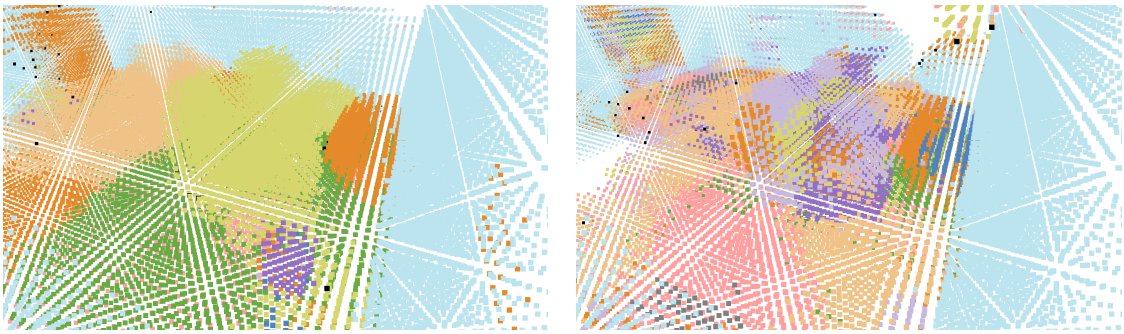


Figure 6.11: Example in dataset 2, for an increase in geometric prediction quality for objects when using *input sem. + noise + dilation + encoding* compared to *ScanComplete* [13], which can be seen on the chairs in this case. It can be observed that more of the chairs has been filled up using the *input sem. + noise + dilation + encoding* approach.



input used ground truth



input sem. + noise + dilation + encoding ScanComplete [13]

Figure 6.12: Example in dataset 2, for an increase in semantic prediction quality for objects when using *input sem. + noise + dilation + encoding* compared to ScanComplete [13], which can be seen on the chairs. It can be observed that for the proposed *input sem. + noise + dilation + encoding* approach, the semantic input provides an initialization for the semantic prediction which is expanded by the network.

6.4.3 Detailed Evaluation for Two Hierarchy Levels

In this section we will evaluate different training methods and the use of dilated convolutions, when using two of the hierarchy levels. Due to memory constraints on the graphics card, it was only possible to train 2 levels of the hierarchy for our proposed method without having to reduce the number of convolutional filters significantly. When using dilated convolutions on 2 levels we only apply them on the second level as initial results showed that this could lead to artifacts being present in the scene, which have a negative influence on the result of the more fine grained hierarchy level.

There are two types of training procedures for multiple hierarchy levels suggested in ScanComplete [13]. The first is to use the ground truth data from the coarser level, when training the hierarchy level directly below it. However, the inputs from the coarser level are highly unlikely to always be perfect when inferencing on evaluation data. This is why it was suggested by Dai. et. al. [13] to first infer the needed geometric and semantic input for the training data on the coarse level and then train the level directly below it using this input. This section is dedicated to investigating the effects of this training procedure on our approach. The inferred training data from the coarser level for ScanComplete [13] is generated using 1 level with semantic prediction enabled, while the data for our methods is generated using additional semantic input with semantic prediction enabled. It can be seen in Table 6.2, when comparing line *2 levels + ScanComplete [13] + inferred train* and line *2 levels + ScanComplete* [13], that training on inferred data from the coarser level leads to a reduction in L1 error and an increase in semantic accuracy for ScanComplete [13], as already established by Dai. et. al [13].

When applying this approach to our *2 levels + input sem. + inferred train* method, we also see a decrease in L1 error and an increase in semantic accuracy for occupied voxels in Table 6.2. The overall semantic accuracy decreases very slightly, likely as a result of wrong predictions being made outside of occupied voxels. When comparing the *2 levels + ScanComplete* [13] *+ inferred train* (middle right) and the *2 levels + input sem. + inferred train* (middle left) method, which are both trained using inferred predictions from the coarser level, it can be seen that the geometric results shown in Figure 6.13 are improved when using the *2 levels + input sem. + inferred train* method, which provided additional input semantics. This is evident on the roof of the left building and the front wall of the right building. Furthermore, there are less incorrect completions performed for the door on the left building for *2 levels + input sem. + inferred train* when compared with *2 levels + ScanComplete* [13] *+ inferred train*. An interesting effect is that the side wall of the left building is closed for both methods, however as seen in the ground truth this should be kept empty. This is the case, as there are no structures in the input data to suggest this and the network was not aware of this. Moreover, the wall on the right building is extended beyond the building itself. The additional input semantics used in the *2 levels + input sem. + inferred train* approach (middle left) cannot prevent this effect, as there are no voxels from any intersecting walls close by as seen in Figure 6.14.

As a result the network thinks the wall extends beyond the building. Analogous to using 1 level, the semantic prediction is also improved upon when using the *2 levels + input sem. + inferred train* approach and contains less noise and incorrect predictions than the *2 levels + ScanComplete* [13] *+ inferred train* method.

Another interesting result observable in Table 6.2, is that the *1 level + input sem. + noise + dilation + encoding* approach, outperforms the approach using input semantics and two hierarchy levels trained on inferred results from the coarse level denoted as *2 levels + input sem. + inferred train*. As a result, one could complete scenes using only a single level, which means inferring on only one network and save inference time. This is however not entirely the case when comparing the *1 level + input sem. + noise + dilation + encoding* method with *2 levels + input sem. + inferred train + encoding + dilation + noise*. It can be seen in Table 6.2, that while the overall L1 and semantic performance metrics for *2 levels + input sem. + inferred train + encoding + dilation + noise* are worse compared to *1 level + input sem. + noise + dilation + encoding*, the results on occupied voxels are improved. This means that the network made a lot of incorrect predictions in non-occupied space, however the predictions on occupied voxels. where data should be predicted are better. The geometric structures the network predicts are natural extensions of the already existing building geometry and are potentially meaningful, however in this case they are not present in the ground truth. An example for this is shown in Figure 6.13 and Figure 6.14 with the *1 level + input sem. + noise + dilation + encoding* (bottom) result, where the building walls have been almost completely restored, however the geometry on the floor has also been extended, which is not present in the ground truth.

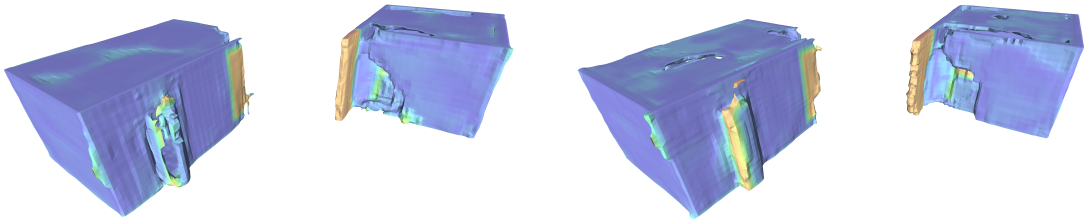
6.4.3.1 Conclusion

In our evaluation procedure using two hierarchy levels, it was observable that both ScanComplete [13] and the *input sem.* approach improved when training on inferred data from the coarser level. However, in terms of overall performance the *1 level + input sem. + noise + dilation + encoding* approach, which utilizes 1 level in combination with dilated convolutions, semantic input, the alternative encoding and noise for the semantic input during training, yields better results when comparing the overall L1 error. This is due to a lot of additional predictions being made when using the *2 levels + input sem. + inferred train + encoding + dilation + noise* method. On the other hand, the L1 error on occupied voxels is lower when using *2 levels + input sem. + inferred train + encoding + dilation + noise*, which means that the geometry that should have been predicted by the network is more correct. This concludes that using *1 level + input sem. + noise + dilation + encoding* offers good performance, while not having to train two networks and infer on two networks.

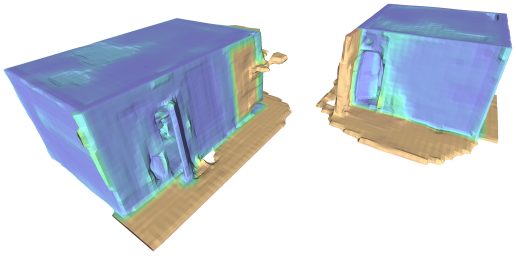
Method	∅ L1	∅ L1 occ.	∅ Sem. acc.	∅ Sem. acc. occ.
2 levels + input sem. + inferred train + encoding + dilation + noise	0.0803	**0.1593**	0.8195	**0.7166**
2 levels + input sem. + inferred train	0.0758	0.1737	0.8189	0.6456
2 levels + input sem.	0.0962	0.2400	0.8219	0.5950
1 level + input sem. + noise + dilation + encoding	**0.0674**	0.1721	**0.8611**	0.6965
2 levels + ScanComplete [13] + inferred train	0.0781	0.2012	0.7661	0.4310
2 levels + ScanComplete [13]	0.1019	0.2622	0.7651	0.4217

Table 6.2: L1 error: lower means better. Semantic accuracy: higher means better. Average error metrics using different training procedures for two hierarchy levels. It can be seen that the *1 level + input sem. + noise + dilation + encoding* approach using all improvements outperforms all of the other methods in overall error metrics. However, on occupied voxels the *2 levels + input sem. + inferred train + encoding + dilation + noise* approach yields better results, but infers a lot of wrong predictions for non occupied voxels.

input used ground truth

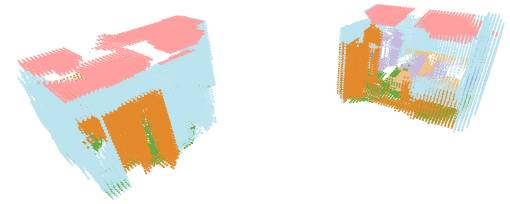


2 levels + input sem. + inferred train 2 levels + ScanComplete [13] + inferred train

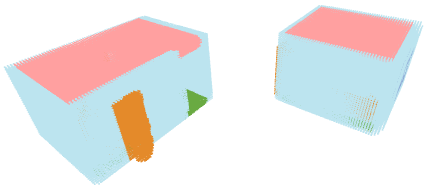


2 levels + input sem. + inferred train + encoding + dilation + noise

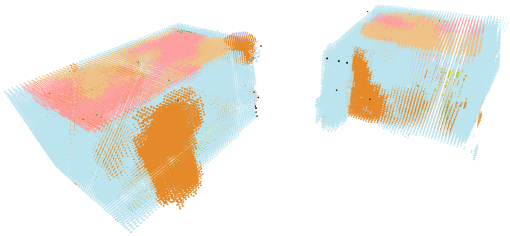
Figure 6.13: Geometric results on dataset 39 for various methods using 2 levels. An initial comparison can be made when looking at *2 levels + input sem. + inferred train* and *2 levels + ScanComplete [13] + inferred train*. It is observable that, when using *2 levels + input sem. + inferred train*, the building is more complete, especially on the ceiling of the left building and that there are less prediction errors on the door of the left building. Furthermore a result for *2 levels + input sem. + inferred train + encoding + dilation + noise* is included, where it is observable that while the reconstruction is more complete overall, some prediction errors have been made, where the network has inferred geometry which is not present in the ground truth, such as the added floor in front of the building. It is noteworthy however, that this is not arbitrary clutter, but a natural extension of the building.



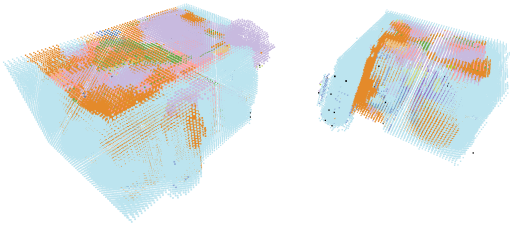
input used



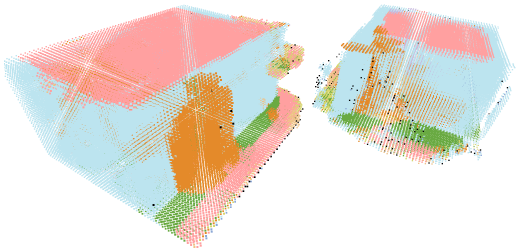
ground truth



2 levels + input sem. + inferred train



2 levels + ScanComplete [13] + inferred train



2 levels + input sem. + inferred train + encoding + dilation + noise

Figure 6.14: Semantic prediction results on dataset 39 for various methods using 2 levels. When comparing *2 levels + input sem. + inferred train* with *2 levels + ScanComplete [13] + inferred train*, it can be seen that the semantic prediction contains less noise and is overall more correct when using additional input semantics. Moreover, a result for *2 levels + input sem. + inferred train + encoding + dilation + noise* is provided, where it is observable that additional semantic information not present in the ground truth has been predicted on the floor of the left building and on the wall of the right building.

6.4.4 Comparison with other Methods on Synthetic Data

In this section, we will compare the proposed method with two other approaches, namely SSCNet [47] and Poisson-Surface Reconstruction [26] as implemented in MeshLab [10]. These methods have been chosen as SSCNet [47] is a another neural network based method and Poisson-Surface Reconstruction [26] represents a more traditional algorithmic approach. This allows us to compare performance with two different underlying approaches. We compare the proposed method with the original ScanComplete [13] architecture trained on our training set and with the two other methods using the L1 error metric.

6.4.4.1 Evaluation Procedure for SSCNet

In this section we elaborate on generating comparable results for evaluation using SSCNet [47]. Because SSCNet [47] is restricted to completing data from single depth images, we apply the pre-trained network, which was trained on SUNCG [47] data to the synthetically generated input depth maps. We then integrate all of the voxel grid outputs generated by the network into a single voxel grid. This is done on a resolution of 8cm as SSCNet [47] works with input data on a resolution of 2cm and then outputs a grid with 8cm resolution due to an included pooling layer.

6.4.4.2 Evaluation Procedure for Poisson Surface Reconstruction

In order to generate results used for evaluation with Poisson Surface Reconstruction [26], we first apply a MeshLab [10] filter script to the input reconstruction, which was generated using Open3D [65] for each evaluation set. The filter applies a Poisson Surface Reconstruction [26] using the vertices from the input mesh as points. The normals are calculated from the faces of the mesh as Poisson Surface Reconstruction [26] requires points and corresponding normals as an input. We then use SDFGen [9] to generate the respective distance function representation, using a voxel size of $0.09m$ for evaluation on the L1 error metric.

6.4.4.3 Results

In this section we compare results generated using SSCNet [47] and Poisson Surface Reconstruction [26] with our own methods and ScanComplete [13]. When looking at the results in Table 6.3, it can be seen that SSCNet [47] is the worst performing method overall, in terms of overall L1 error and L1 error on occupied voxels. This is also evident when looking at the plots in Figure 6.17, where the method yields a higher overall L1 error for all datasets, except for one and a higher error on occupied voxels for all datasets. Because this method is only able to work on single depth images, it is not able to complete large missing patches where no depth images were available, as seen by the missing geometry on the back side of the left building in Figure 6.15 for the SSCNet [47] (bottom left) result and thus it performs worse in general.

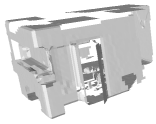
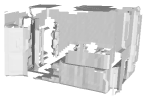
Poisson Surface Reconstruction [26] still achieves quite good results on the overall L1 error, as it is on the same level as *2 levels + ScanComplete* [13] *+ inferred train*. This is likely because most of the generated surface is very close to points of the input data. This means that completely wrong inferences such as the extended wall seen for *1 level + input sem. + noise + dilation + encoding* (bottom) in Figure 6.13, are not an issue with this method. Moreover, the results using Poisson Surface Reconstruction [26] are very smooth, such that the general building shapes become very round. On the other hand this method does not take general geometric properties of the input data into account and as such the buildings and objects contained in them are reconstructed very coarsely, which increases the error on occupied voxels significantly. This is also evident, when comparing error metrics in Figure 6.17. It can be seen that the L1 error on occupied voxels is higher for every dataset, while the overall L1 error is even lower than our proposed *1 level + input sem. + noise + dilation + encoding* approach and *2 levels + ScanComplete* [13] *+ inferred train* for some evaluated scenes. As mentioned above, this is a result of wrong inferences made by the neural network based methods. This is especially evident on smaller scenes such as the room shown in Figure 6.16 when comparing the *Poisson Surface Reconstruction* [26] result (bottom right) with the other methods shown.

Method	∅ L1	∅ L1 occupied
SSCNet [47]	0.1568	0.4828
Poisson Surface Reconstruction [26]	0.1019	0.3743
2 levels + input sem. + inferred train + encoding + dilation + noise	0.0803	**0.1593**
2 levels + input sem. + inferred train	0.0758	0.1737
2 levels + input sem.	0.0962	0.2400
1 level + input sem. + noise + dilation + encoding	**0.0674**	0.1721
2 levels + ScanComplete [13] + inferred train	0.0781	0.2012
2 levels + ScanComplete [13]	0.1019	0.2622

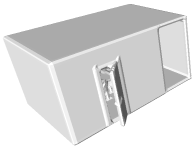
Table 6.3: L1 error: lower means better. Error metrics comparison with SSCNet [47] and Poisson Surface Reconstruction [26]. It can be seen that our *1 level + input sem. + noise + dilation + encoding* approach outperforms all other listed methods in terms of overall L1 error. However it falls behind the *2 levels + input sem. + inferred train + encoding + dilation + noise* method in terms of L1 error on occupied voxels. Furthermore, Poisson Surface Reconstruction [26] performs better than SSCNet [47].

6.4.4.4 Conclusion

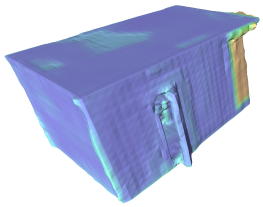
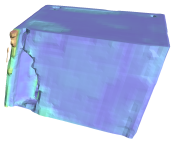
During our comparison with the two selected methods, Poisson Surface Reconstruction [26] achieved a lower L1 error on occupied voxels compared to SSCNet [47], as it was able to take more of the global context into account. Furthermore the results for Poisson Surface Reconstruction [26] do not deviate a lot from the original input point cloud and as such the overall L1 error is lower as well. However, compared to *2 levels + ScanComplete* [13] + *inferred train* , as well as the proposed *1 level + input sem. + noise + dilation + encoding* method, Poisson Surface Reconstruction [26] yields worse results due to not being able to incorporate the semantic or geometric context of the input geometry.

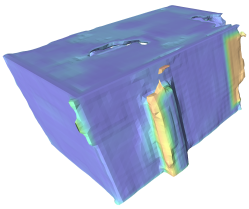
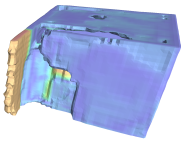
input used



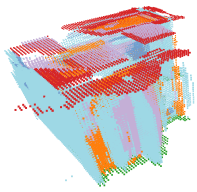
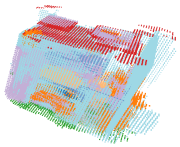

ground truth

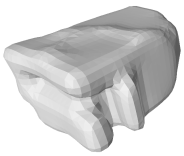
1 level + input sem. + noise + dilation + encoding

2 levels + ScanComplete [13] + inferred train

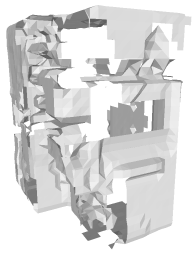



SSCNet [47]

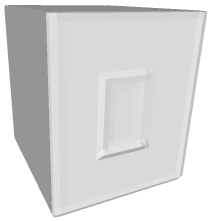



Poisson Surface Reconstruction [26]

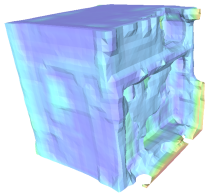
Figure 6.15: Comparison of geometric results of *2 levels + ScanComplete [13] + inferred train* and *1 level + input sem. + noise + dilation + encoding* with Poisson Surface Reconstruction [26] and SSCNet [47] on dataset 39. It can be seen in the SSCNet [47] result, that larger parts of the building on the left are missing. In general Poisson Surface Reconstruction [26] yields a very smooth and round result, which does not really fill the wall and floor correctly. Overall *1 level + input sem. + noise + dilation + encoding* outperforms the other methods shown as it fills more of the geometry on the ceiling and walls.



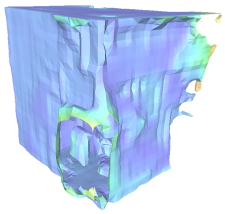
input used



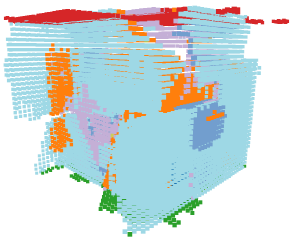
ground truth



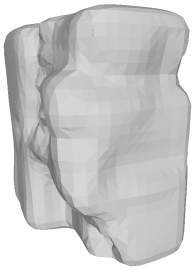
1 level + input sem. + noise + dilation + encoding



2 levels + ScanComplete [13] + inferred train

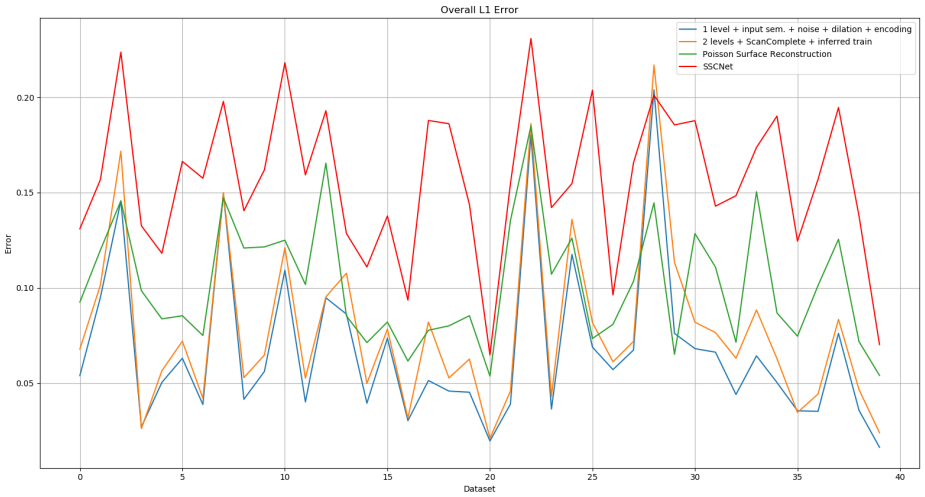


SSCNet [47]

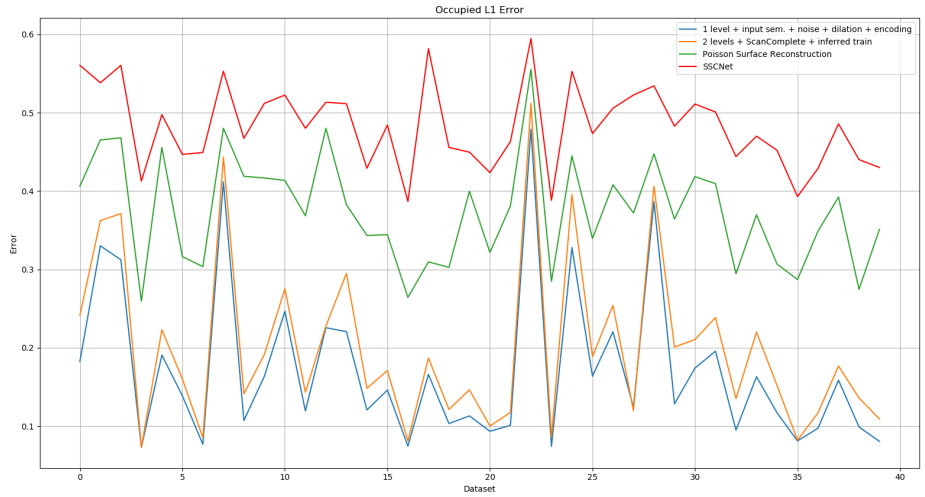


Poisson Surface Reconstruction [26]

Figure 6.16: Comparison of geometric results of *2 levels + ScanComplete [13] + inferred train* and *1 level + input sem. + noise + dilation + encoding* with Poisson Surface Reconstruction [26] and SSCNet [47] on dataset 29. It can be seen that Poisson Surface Reconstruction [26] yields a surface which closely follows the shape of the original building, while the *1 level + input sem. + noise + dilation + encoding* approach and *2 levels + ScanComplete [13] + inferred train* introduce some artifacts which deviate from the ground truth. Note that the *1 level + input sem. + noise + dilation + encoding* shows the other side of the result as more artifacts are visible there. The SSCNet [47] result also shows that the walls extend beyond the original ground truth boundaries.



Overall L1 error metrics.



L1 error metrics on occupied voxels.

Figure 6.17: L1 error metrics for all 40 evaluation sets evaluated using Poisson Surface Reconstruction [26] and SSCNet [47] compared with *2 levels + ScanComplete [13] + inferred train* and *1 level + input sem. + noise + dilation + encoding*. It can be seen that, while Poisson Surface Reconstruction [26] manages to outperform the other methods on some datasets, *1 level + input sem. + noise + dilation + encoding* yields better results in general.

6.4.5 Experimental Results on Real-World Image-Based Reconstructions

In this section we evaluate our proposed approach on datasets from the ETH3D [45] benchmark. As described in Section 6.1.3, we evaluate using the F1 score metric and use the input poses provided by the benchmark to perform a dense image-based reconstruction using COLMAP [44] and Open3D [65]. Note that we also had to limit the maximum depth values used during integration in order to keep the reconstructed scenes small in terms of voxel grid dimensions, such that we do not exceed the memory limits of our test system. Moreover, we have also changed the truncation value for visualizing occupied voxels of semantic predictions to 0.15 for these datasets, which yields more comprehensible visualizations as there is a lot of clutter present for some scenes. In the following sections we present results for the datasets *office*, *pipes*, *relief* and *terrains*. We compare our approach with the best performing methods evaluated in Section 6.4.4, which are ScanComplete [13] and Poisson Surface Reconstruction [26].

6.4.5.1 Experimental Results

In this section we perform a detailed analysis in terms of F1-score with the best performing methods from the evaluation on synthetic data. Furthermore, we also evaluate the influence of applying synthetic noise to the semantic input during training. It can be seen in Table 6.4, that the F1 score on the *pipes* and *office* datasets is improved for our proposed *1 level + input sem. + noise + dilation + encoding* and *input sem.* approaches compared to *1 level + ScanComplete* [13] and *2 levels + ScanComplete* [13] + *inferred train*, however on the *relief* and *terrains* datsets the F1 score of our approaches falls behind *1 level + ScanComplete* [13]. As seen in Table 6.5, this is due to a decrease in accuracy for the proposed methods on these datasets. The completeness on the other hand is universally improved for our approaches as seen in Table 6.6. It is also noteworthy that the decrease in accuracy is stronger for methods which in infer more geometry and thus also make more wrong predictions, such as the methods which use dilated convolutions like *1 level + input sem. + noise + dilation + encoding*. Moreover the synthetic noise inserted into the semantic input data during training seems to increase the accuracy in some cases, for instance when comparing the results for *1 level + input sem. + noise* with *input sem.* on the *relief* and *terrains* datasets in Table 6.5. An example for this decrease in clutter due to noise is also visualized in Figure 6.25 when comparing *1 level + input sem. + noise* (bottom left) with*1 level + input sem.* (bottom right).

A significant increase in completeness is achieved using our *2 levels + input sem. + inferred train + encoding + dilation + noise* approach, which uses 2 levels and all improvements combined. This is also visualized in Figure 6.18 (middle left) and the completeness visualization in Figure 6.24 (bottom left). Furthermore the quality of the semantic prediction is improved visually as seen in Figure 6.19 (top right), Figure 6.23 (top

right) and Figure 6.21 (top right), when comparing the *2 levels + input sem. + inferred train + encoding + dilation + noise* approach with the *2 levels + ScanComplete* [13] *+ inferred train* (bottom left) method and the *1 level + ScanComplete* [13] (bottom left) approach. Because there is no semantic ground truth evaluation data included in the ETH3D [45] benchmark, we are only able to compare these results visually.

Overall, we can conclude that the approaches using additional semantic input, combined with other improvements are able to use the provided context to their advantage, by completing more scene geometry which has been labeled correctly. However, the increased completeness in this case results in worse accuracy metrics by potentially wrong additional predictions.

The decrease in accuracy for our proposed approaches can be visualized in Figure 6.24, by comparing results from the *2 levels + input sem. + inferred train + encoding + dilation + noise* (top left) approach and *1 level + ScanComplete* [13] (top right). It can be seen that, while our *2 levels + input sem. + inferred train + encoding + dilation + noise* approach predicts significantly more surface area, especially on the floor, the overall accuracy is decreased, as more wrong predictions are introduced, which is evident in the ceiling area of the dataset. The completeness measure however is increased, which can also be seen in Figure 6.20 when comparing the *2 levels + input sem. + inferred train + encoding + dilation + noise* result (middle left) with the other presented results.

In general Poisson Surface Reconstruction [26] achieves worse overall results compared to the proposed methods and ScanComplete [13] approaches, in terms of completion. However, in terms of accuracy the results are even better than the two neural network based methods on the *terrains* dataset as seen in Table 6.5. This is an analogous development to the evaluation on synthetic data, as the surface generated by Poisson Surface Reconstruction [26] is always close to the input reconstruction and thus yields better accuracy values in some scenarios. The *terrains* dataset represents an ideal case for this as it is a long hallway where the Poisson surface can wrap around, which is visualized in Figure 6.22 (bottom right). The additional clutter generated by the neural network based approaches decreases the accuracy and thus they fall slightly behind Poisson Surface Reconstruction [26].

In general, it can be seen that additional input semantics provide benefits in areas where the semantic input predictions are mostly correct such as the floor and wall. In these areas the network can use the additional context to provide more complete reconstructions. More complex objects with noisy semantic predictions on the other hand can lead to a decrease in accuracy when being completed by the network.

F1 score evaluation	Office	Pipes	Relief	Terrains
2 levels + input sem. + inferred train + encoding + dilation + noise	0.8272	0.4608	0.6014	0.5885
1 level + input sem. + encoding + dilation + noise	**0.8334**	0.5064	0.6245	0.6146
2 levels + input sem. + inferred train + noise	0.7961	0.5140	0.6781	0.6261
2 levels + input sem. + inferred train	0.8061	0.4867	0.6550	0.6430
1 level + input sem. + noise	0.7929	0.4861	0.6786	0.6550
1 level + input sem.	0.7978	**0.5156**	0.6793	0.6623
2 levels + ScanComplete [13] + inferred train	0.7619	0.4785	0.6866	0.6727
1 level + ScanComplete [13]	0.7591	0.4750	**0.7127**	**0.6901**
Poisson Surface Reconstruction [26]	0.4717	0.2212	0.1566	0.3359

Table 6.4: F1 score: higher means better. F1 score results for different methods and datasets. It can be seen that our proposed *1 level + input sem. + noise + dilation + encoding* and *input sem.* approaches achieve higher scores on *office* and *pipes*, but the results are worse than *1 level + ScanComplete* [13] on *relief* and *terrains*. This is due to a decrease in accuracy for these datasets when using our proposed approaches, while the completeness is still superior.

Accuracy Evaluation	Office	Pipes	Relief	Terrains
2 levels + input sem. + inferred train + encoding + dilation + noise	**0.7417**	0.4022	0.5189	0.4752
1 level + input sem. + encoding + dilation + noise	0.7346	0.5720	0.5602	0.5485
2 levels + input sem. + inferred train + noise	0.7126	0.6098	0.6455	0.5572
2 levels + input sem. + inferred train	0.7250	0.5745	0.6022	0.5917
1 level + input sem. + noise	0.7292	0.6148	0.6635	0.6394
1 level + input sem.	0.7118	0.6334	0.6520	0.6391
2 levels + ScanComplete [13] + inferred train	0.7252	0.6400	0.6899	0.6763
1 level ScanComplete [13]	0.7144	**0.6642**	**0.7379**	**0.7009**
Poisson Surface Reconstruction [26]	0.6742	0.5593	0.4956	**0.7114**

Table 6.5: Accuracy: higher means better. Accuracy results for different methods and datasets. It is observable that the proposed approaches achieve a high accuracy only on the *office* dataset, while the other results are comparable to *1 level + ScanComlete* [13], but worse in general. This is due to the fact that by completing additional scene geometry, clutter is introduced by the proposed methods, which reduces the accuracy.

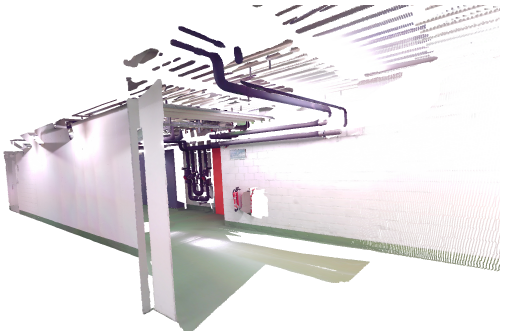
Completeness Evaluation	Office	Pipes	Relief	Terrains
2 levels + input sem. + inferred train + encoding + dilation + noise	0.9349	**0.5393**	0.7150	**0.7729**
1 level + input sem. + encoding + dilation + noise	**0.9628**	0.4543	0.7055	0.6989
2 levels + input sem. + inferred train + noise	0.9019	0.4442	0.7141	0.7145
2 levels + input sem. + inferred train	0.9077	0.4222	**0.7178**	0.7041
1 level + input sem. + noise	0.8687	0.4019	0.6944	0.6714
1 level + input sem.	0.9074	0.4347	0.7090	0.6873
2 levels + ScanComplete [13] + inferred train	0.8025	0.3820	0.6833	0.6692
1 level ScanComplete [13]	0.8099	0.3698	0.6892	0.6796
Poisson Surface Reconstruction [26]	0.3628	0.1379	0.0930	0.2199

Table 6.6: Completeness: higher means better. Completeness results for different methods and datasets. Overall the completeness metric is superior for our proposed methods on every dataset evaluated. It can be seen that proposed method utilizing dilated convolutions and 2 levels, denoted as *2 levels + input sem. + inferred train + encoding + dilation + noise* achieves the best on pipes and terrains. For *office 1 level + input sem. + noise + dilation + encoding* performs better, however there are only minor differences to *2 levels + input sem. + inferred train* on relief.

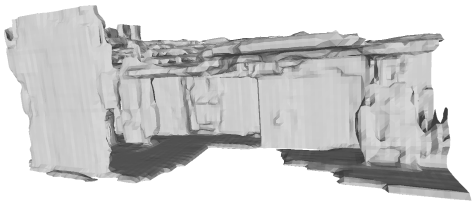
Average Evaluation	∅ F1	∅ Accuracy	∅ Completeness
2 levels + input sem. + inferred train + encoding + dilation + noise	0.6195	0.5345	**0.7405**
1 level + input sem. + encoding + dilation + noise	0.6447	0.6038	0.7053
2 levels + input sem. + inferred train + noise	0.6536	0.6313	0.6937
2 levels + input sem. + inferred train	0.6477	0.6234	0.6879
1 level + input sem. + noise	0.6531	0.6617	0.6591
1 level + input sem.	**0.6637**	0.6591	0.6846
2 levels + ScanComplete [13] + inferred train	0.6499	0.6828	0.6343
1 level ScanComplete [13]	0.6593	**0.7043**	0.6371
Poisson Surface Reconstruction [26]	0.2964	0.6101	0.2034

Table 6.7: Averaged results for different methods and all datasets. In terms of average F1 score our *input sem.* method performs best, as it provides a good balance between completion and accuracy. However, in terms of overall accuracy *1 level + ScanComplete* [13] yields better results. The highest completeness metric is achieved with our *2 levels + input sem. + inferred train + encoding + dilation + noise* approach, which uses 2 levels and all improvements.

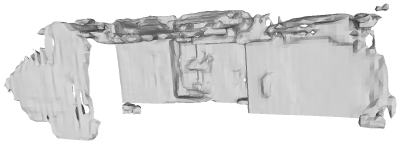
input used



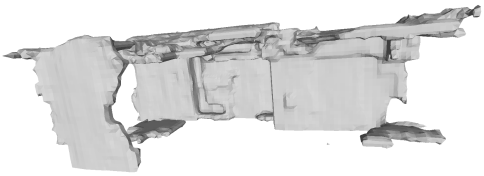
ground truth



2 levels + input sem. + inferred train + encoding + dilation + noise



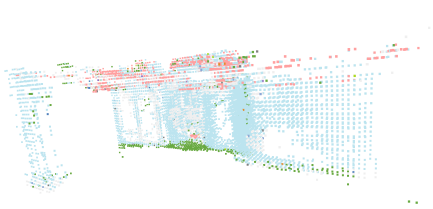
2 levels + ScanComplete [13] + inferred train



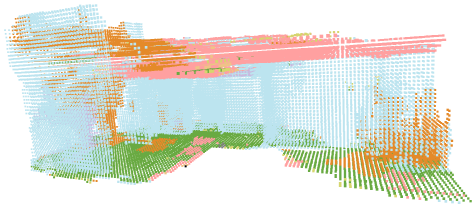
1 level + input sem.

Poisson Surface Reconstruction [26]

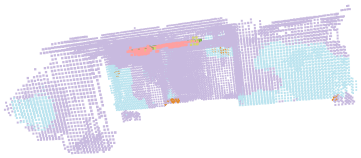
Figure 6.18: Geometric result comparison for the pipes dataset. It is observable that the *2 levels + input sem. + inferred train + encoding + dilation + noise* method yields the most complete result, while also introducing additional clutter in the ceiling area and on the outer right side of the wall. The one *level + input sem.* approach yields a more complete result compared to the *2 levels + ScanComplete [13] + inferred train* method, which can be seen on the floor and the wall to the left, while not introducing a large amount of additional clutter. The result for Poisson Surface Reconstruction [26] completes most of the wall and ceiling, but does not respect the geometry of the original input and produces very round and smoothed results. Note that for the Poisson Surface Reconstruction [26] the camera view is from inside the model to show the same environment as in the other results.



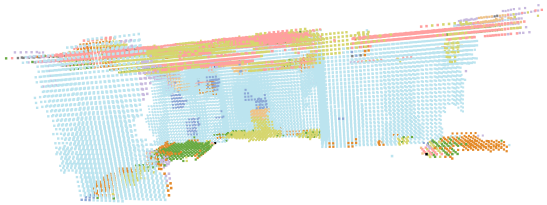
input used



2 levels + input sem. + inferred train + encoding + dilation + noise

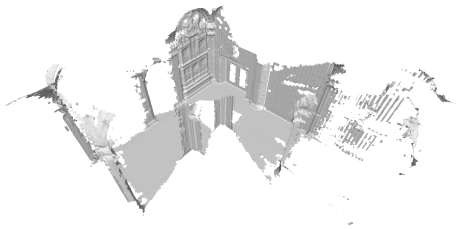


2 levels + ScanComplete [13] + inferred train

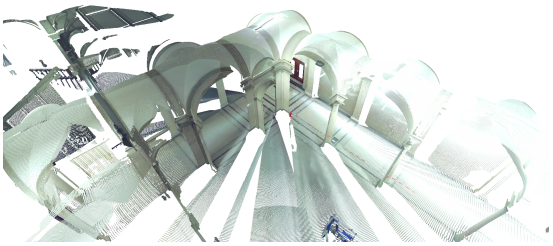


1 level + input sem.

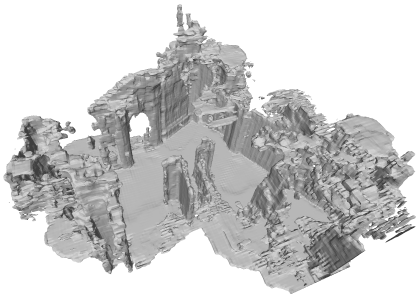
Figure 6.19: Semantic prediction comparison for pipes dataset. It can be seen that the *2 levels + input sem. + inferred train + encoding + dilation + noise* method yields the most complete semantic prediction with predictions for wall (blue), floor (green) and ceiling (red), which do not contain a lot of noise. The *2 levels + ScanComplete [13] + inferred train* approach was not able to identify walls, floor and ceiling correctly consistently. This is likely due to not being provided additional semantic input. The *level + input sem.* method yields a result where floor, ceiling and wall have been predicted correctly in large areas, however there is still some noise present specifically in the ceiling area.



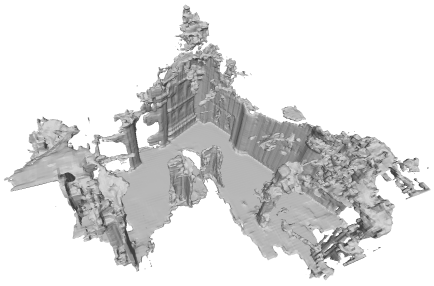
input used



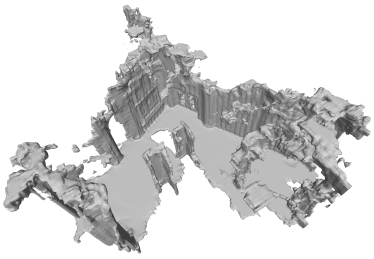
ground truth



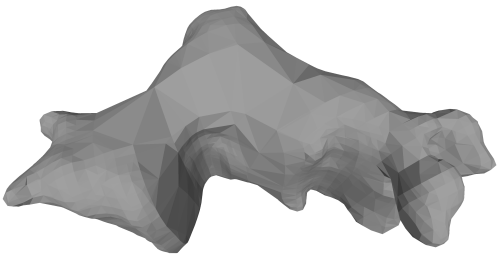
2 levels + input sem. + inferred train + encoding + dilation + noise



1 level + ScanComplete [13]



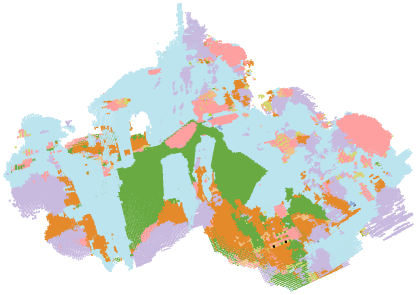
1 level + input sem. + noise



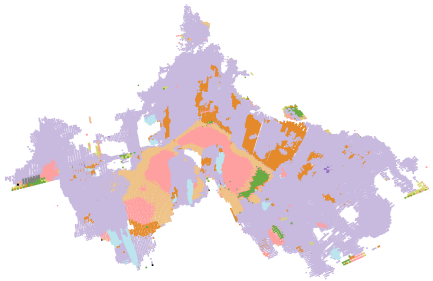
Poisson Surface Reconstruction [26]

Figure 6.20: Geometric result comparison for the relief dataset. It can be seen that the *2 levels + input sem. + inferred train + encoding + dilation + noise* approach yields the most complete result compared to others, while also introducing a lot of clutter, specifically in the area of the ceiling. The result for Poisson Surface Reconstruction [26] does not respect most of the more fine grained geometric structures present in the input reconstruction.

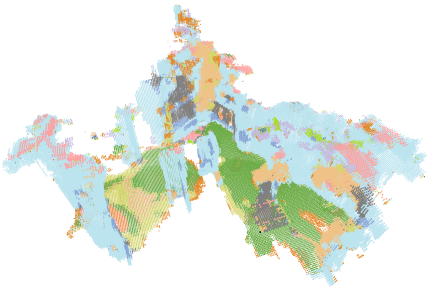
input used



2 levels + input sem. + inferred train + encoding + dilation + noise

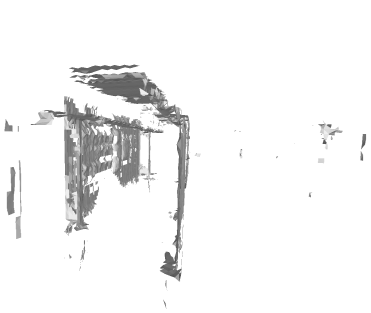


1 level + ScanComplete [13]

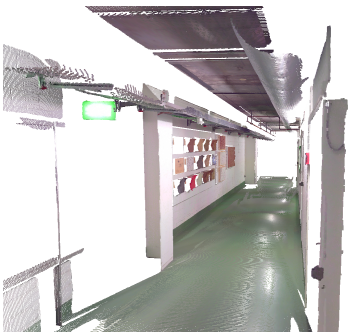


1 level + input sem. + noise

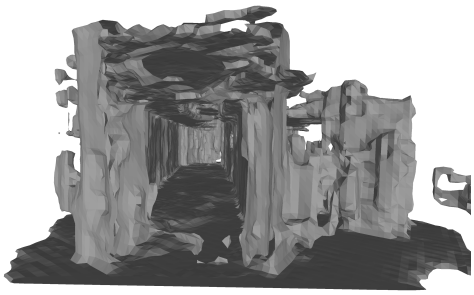
Figure 6.21: Semantic prediction comparison for relief dataset. It is observable that the *2 levels + input sem. + inferred train + encoding + dilation + noise* method yields the best overall result, but still includes some wrong predictions and noise in the areas of the walls and floor.



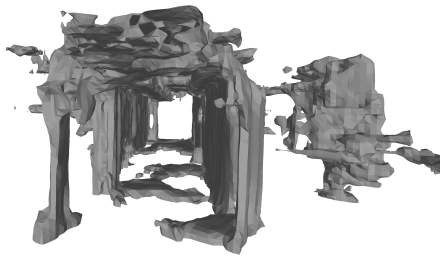
input used



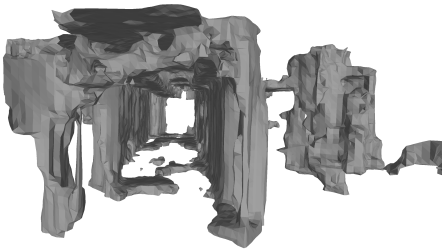
ground truth



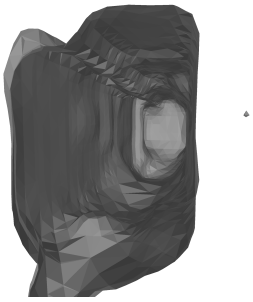
2 levels + input sem. + inferred train + encoding + dilation + noise



1 level + ScanComplete [13]

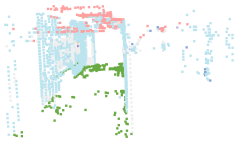


1 level + input sem. + noise

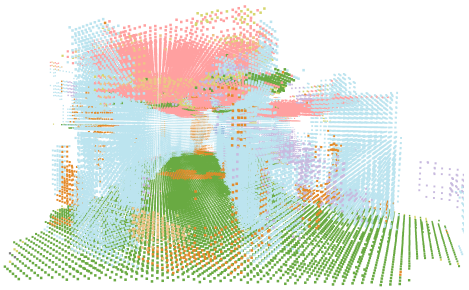


Poisson Surface Reconstruction [26].

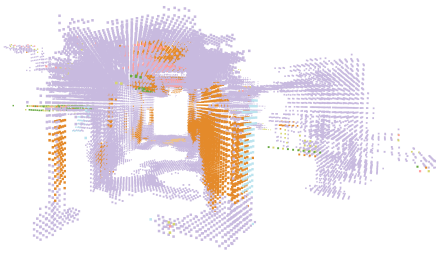
Figure 6.22: Geometric results for the *terrains* dataset. It can be seen that for the Poisson Surface Reconstruction [26] result, the surface fits the outer walls and not a lot of clutter is present. Note that for the Poisson Surface Reconstruction [26] the camera view is from inside the model to show the same environment as in the other results. The *2 levels + input sem. + inferred train + encoding + dilation + noise* result is more complete than *1 level + input sem. + noise* and *1 level + ScanComplete* [13] results, however a lot of additional predictions are also introduced in the ceiling area.



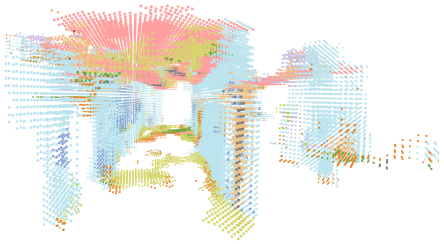
input used



2 levels + input sem. + inferred train + encoding + dilation + noise

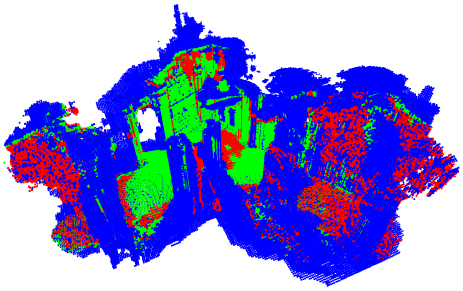


1 level + ScanComplete [13]

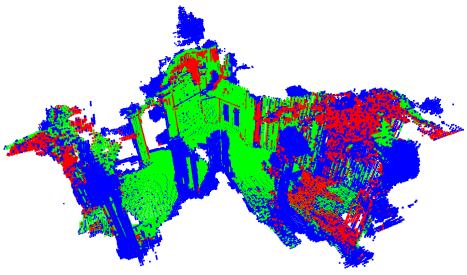


1 level + input sem. + noise

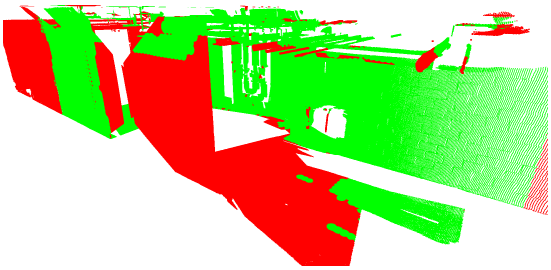
Figure 6.23: Semantic prediction results for the *terrains* dataset. It can be seen that the *2 levels + input sem. + inferred train + encoding + dilation + noise* result contains less noise than the *1 level + input sem. + noise* result and was able to make a more correct prediction of the floor (green) than *1 level + input sem. + noise* and *1 level + ScanComplete* [13].



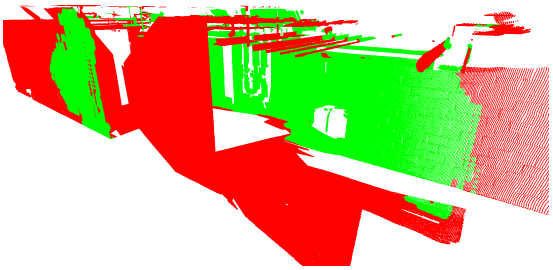
Accuracy visualization for 2 levels + input sem. + inferred train + encoding + dilation + noise



Accuracy visualization for 1 level + ScanComplete [13]



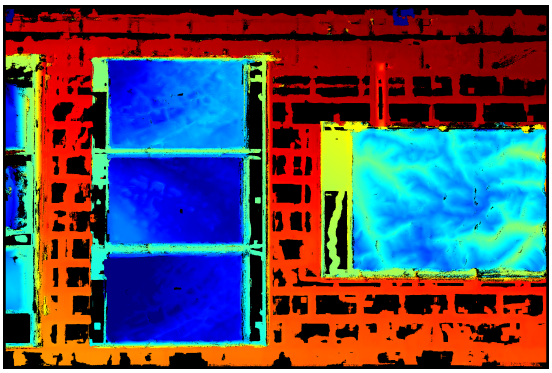
Completeness visualization for 2 levels + input sem. + inferred train + encoding + dilation + noise



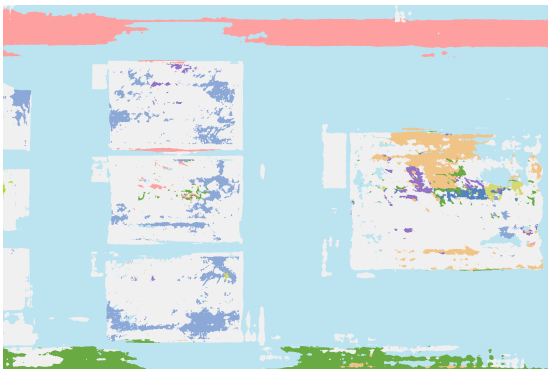
Completeness visualization for 2 levels + ScanComplete [13] + inferred train

Figure 6.24: Accuracy comparison for the relief dataset and completeness comparison on the pipes dataset. For the accuracy visualization, it can be seen that the additional clutter and completed data added in the ceiling area for *2 levels + input sem. + inferred train + encoding + dilation + noise* leads to more inaccurate points (red) in contrast to accurate points (green), when comparing the result with *1 level + ScanComplete* [13]. The blue points represent points outside of the laser scanner range. The completeness visualization shows an increase in completeness for the *2 levels + input sem. + inferred train + encoding + dilation + noise* approach compared to *2 levels + ScanComplete [13] + inferred train*, which can be seen by the increased amount of green points (complete), in contrast to red points (incomplete).

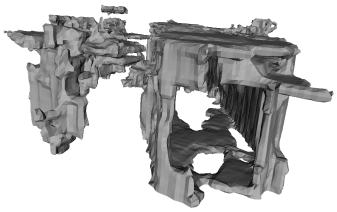
Example for an RGB image for which a depth map and semantically segmented image have been created in the *terrains* dataset.



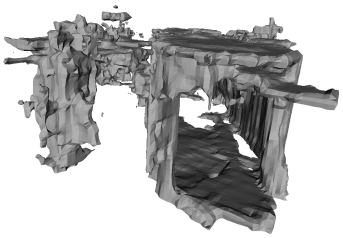
Example for an input depth map used for *terrains*.



Example for a semantically segmented image used for *terrains*.



1 level + input sem. + noise



1 level + input sem.

Figure 6.25: Comparison for geometric results when adding noise to the input semantics during training. It can be seen that the result which was trained on data containing noise reduces some of the clutter outside of the wall introduced by incorrect input depth maps. The color in the depth image is chosen such that larger depth values yield a color which has a red tone, while a blue tone encodes a smaller depth value. It is observable that a lot of the wall contains large depth values, which are not valid in this case and lead to noise when combining the depth maps in Open3D [65]. It can also be seen that incorrect values appear in areas where there is almost no texture, such as the wall.

6.4.5.2 Conclusion

While evaluating on real world image based reconstructions, we have seen that giving the network additional context via input semantics leads to more complete results. However, as the network tries to infer more of the geometry, additional clutter is added which decreases accuracy. This effect is amplified when using 2 levels. Furthermore, adding synthetic noise to the semantic input data during training improved on the accuracy, as it removed some of the clutter present due to imperfect input depth maps and input semantic information.

6.5 Conclusions

In this section we will summarize the results from the performed experiments and draw conclusions about strengths, weaknesses and potential improvements of our approaches.

In Section 6.4.2, we showed that providing additional input semantics results in a performance increase, both in terms of L1 error and semantic accuracy. We also showed that using dilated convolutions yields the largest further improvement of our approach, while combining it with synthetic noise inserted into input semantic during training and an alternative semantic encoding lead to even further improvement.

When evaluating on 2 levels in Section 6.4.3, we discovered that using 1 level with dilated convolutions yields comparable results for prediction on occupied voxels and better overall error results, than when using our best performing approach with two hierarchy levels. This means it is possible to save time by training only one network and inferring on only one network and still generate good results.

The evaluation with other methods performed in Section 6.4.4, showed that in general our proposed approach yields the best overall results compared to SSCNet [47] and Poisson Surface Reconstruction [26], while ScanComplete [13] improves significantly over the other two methods as well.

Our experiments on real world image based reconstructions, showed a general increase in completeness for our proposed approach compared to ScanComplete [13] and Poisson Surface Reconstruction [26]. However this also meant a decrease in accuracy on some datasets as the network made wrong predictions. In general, it could be seen that providing additional semantic information provides benefits, especially for wall and floor regions. Furthermore we can conclude that the method which provides a good trade off between completion and accuracy is the *1 level + input sem. + encoding + dilation + noise* approach.

7 Conclusion

Contents

In this chapter, we summarize the main implications of this thesis, draw conclusions and elaborate on limitations of our method, as well as possible future work which could solve these problems.

7.1 Summary and Conclusions

In this work, we explored a *CNN* based technique to recover the oftentimes incomplete geometry of image-based reconstructions, which are a commonly used technique of generating 3D data from real world environments. Because acquiring large amounts of ground truth data from real world scenes presents a lot of difficulties, we trained the network on synthetic data generated from the SUNCG dataset [47]. As the basis for our work we use the neural network architecture proposed in ScanComplete [13], which takes as an input a regular voxel grid encoding geometric information as a *TSDF* and outputs distance function values in a regular voxel grid, which encode the completed geometry. We presented an architecture, which extends the approach proposed in ScanComplete [13] in multiple ways, with the goal of increasing the performance of the network in terms of completion. In order to measure the impact of our changes, we first evaluated these modifications on a synthetic evaluation set, before applying them to real-world image-based reconstructions.

As elaborated on in Section 4.3, we provide the network with additional input semantic information, which gives the network more context when completing the geometry of a scene. Our experiments on synthetic data in Section 6.4.2.1 showed, that this additional input leads to improved performance in terms of how much structure and geometry was completed. This extension also resulted in an improvement for semantic prediction, as the

network was given an initialization for semantic labels, which it could expand upon.

Furthermore, we used dilated convolutions when inferring on one hierarchy level, which also yields improvements both in terms of geometric and semantic prediction on synthetic data as described in Section 6.4.2.2. We observed that providing the network with more global context using dilated convolutions, which increases the receptive field of the network, leads to a more substantial increase in overall performance metrics than using an additional coarser hierarchy level and no dilated convolutions. This implies that one can reduce the amount of time used for training and inference, by using only a single hierarchy level. However, it is possible to improve the results on occupied voxels with two hierarchy levels, by adding dilated convolutions to the second level and using the alternative semantic encoding and noise during training.

We also evaluated an alternative encoding for the additional semantic input provided to the network, similar to the encoding proposed by Garbade et. al. [18], on synthetic data, which is elaborated on in detail in Section 4.3.5.1. The quantitative results obtained in our performed experiments in Section 6.4.2.4, showed an increase in the accuracy of the semantic predictions, however no improvements could be observed in terms of geometric prediction. When augmenting the training data, such that synthetic noise is included in the input semantic information, a slight improvement could be observed in preventing wrong semantic predictions from being made outside the bounds of the ground truth. However, the semantic prediction accuracy on the ground truth model was more inaccurate, as seen in our evaluation on synthetic data in Section 6.4.2.5.

The combination of all proposed modifications on one hierarchy level also lead to a further performance increase on synthetic data, as leveraging more of the global context was combined with the other less substantial improvements. This was explored in Section 6.4.2.6. The approach using two hierarchy levels yields better results in some aspects, however it is prone to make more wrong predictions outside the bounds of the ground truth and as such, we recommend using one level which represents a good middle ground between performance and inference time.

Over the course of our evaluation, we also compared the proposed method with two other techniques for completing 3D geometry. For this evaluation, we chose a traditional approach with Poisson Surface Reconstruction [26] and another *CNN* based approach with SSCNet [47]. The comparisons showed that these two approaches performed worse in comparison to the proposed approach and ScanComplete [13].

Applying the proposed extensions on real world image based reconstructions of the ETH3D benchmark [45] showed that while improvements were made in terms of how much of the missing geometry was recovered, the results became less accurate with respect to the ground truth than the results generated by the architecture proposed in ScanComplete [13]. It cloud be observed that a lot of additional clutter was generated as well, which could be slightly reduced by incorporating synthetic noise into the semantic input data used during training. This means that there was a trade off between completing more of the scene and being more accurate with respect to the original ground truth. Although this shows that

the method has the potential to improve image based reconstructions and create more complete reconstructions for applications such as virtual reality and augmented reality, there are still additional steps which would need to be taken, to make this a more viable method for these use cases.

7.2 Future Work

In this section we explore the limitations of the proposed approach and discuss potential ways of solving or reducing the impact of these issues. A big restriction of the proposed approach is that the large amount of memory consumed when inferencing and training, limits the method in terms of the maximum possible underlying resolution of the geometry encoded in the voxel grid for scenes with larger dimensions. The issue here is, that the approach uses a regular voxel grid, which stores the truncated values of the distance function as well as the non truncated ones. However, the number of voxels containing truncated values has a larger magnitude than the voxels containing non-truncated values for most of the encoded scenes. Thus, all of these voxels could be represented by a single large voxel, which also stores the truncated value. This concept was proposed by Riegler et. al. [40] for neural networks working on 3D data. The incorporation of this more memory efficient method with our approach would yield further improvements in terms of recovering finer detail and being able to work on larger scenes in general.

Another area, where the proposed method could be improved, is to create training data which resembles real world image-based reconstructions more closely. One of the main problems for image based reconstructions are walls, floors and ceiling areas which are poorly textured. It would be possible to use the 2D semantic information of the synthetic dataset to generate noise on the depth maps in these areas specifically.

Another way to improve on this aspect would be to execute a *MVS* algorithm on images taken from photo-realistic synthetic scenes. These scenes could simulate difficult conditions for image-based reconstructions, for instance brightness differences or poorly textured surfaces. Such photo-realistic images from synthetic scenes can also be generated with the SUNCG dataset [47] by using physically based rendering [59]. However, it is unclear how viable this approach is, in the sense how close the results are to real world *MVS* results. Furthermore, these photo-realistic images could also be used to generate semantic predictions which are closer to real world data, by applying the DeepLabV3 [7] (with parameters/weights from an already trained network using ade20k [62] [63]) network on this data. In this case, it would also be possible to train the network on the used synthetic dataset beforehand.

A

List of Acronyms

BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
CRF	Conditional Random Field
FAST	Features from Accelerated Segment Test
LIDAR	light detection and ranging
MVS	Multi-View Stereo
RELU	rectified linear unit
SFM	Structure from Motion
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
TSDF	truncated signed distance function

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. (page 46, 52, 54)

[2] Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. (page 11)

[3] Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. (2000). Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 417–424, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. (page 2, 29)

[4] Bleyer, M., Rhemann, C., and Rother, C. (2011). Patchmatch stereo - stereo matching with slanted support windows. In *British Machine Vision Conference (BMVC)*. (page 15)

[5] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. (page 52, 54)

[6] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *European Conference on Computer Vision (ECCV)*, ECCV'10, pages 778–792, Berlin, Heidelberg. Springer-Verlag. (page 11)

[7] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision (ECCV)*. (page 3, 48, 54, 109)

[8] Cherabier, I., Schönberger, J., Oswald, M., Pollefeys, M., and Geiger, A. (2018). Learning priors for semantic 3d reconstruction. In *European Conference on Computer Vision (ECCV)*, Cham. Springer International Publishing. (page 33)

[9] Christopher Batty (2015). SDFGen. `https://github.com/christopherbatty/SDFGen`. (page 58, 85)

[10] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association. (page 64, 85)

[11] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA. ACM. (page 15)

[12] Dai, A., Qi, C. R., and Nießner, M. (2017). Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 33, 36)

[13] Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., and Nießner, M. (2018). Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 2, 3, 33, 34, 41, 42, 44, 46, 47, 48, 49, 58, 63, 64, 65, 66, 67, 70, 72, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 105, 107, 108)

[14] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159. (page 45)

[15] Dzitsiuk, M., Sturm, J., Maier, R., Ma, L., and Cremers, D. (2016). De-noising, stabilizing and completing 3d reconstructions on-the-go using plane priors. *CoRR*, abs/1609.08267. (page 28)

[16] Firman, M., Aodha, O. M., Julier, S., and Brostow, G. J. (2016). Structured Completion of Unobserved Voxels from a Single Depth Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 33)

[17] Gallup, D., Frahm, J., Mordohai, P., Yang, Q., and Pollefeys, M. (2007). Real-time plane-sweeping stereo with multiple sweeping directions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. (page 15)

[18] Garbade, M., Sawatzky, J., Richard, A., and Gall, J. (2018). Two stream 3d semantic scene completion. *CoRR*, abs/1804.03550. (page 3, 33, 35, 47, 108)

[19] George, P. L. and Seveno, E. (1994). The advancing-front mesh generation method revisited. *International Journal for Numerical Methods in Engineering*, 37(21):3605–3619. (page 27)

[20] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. (page 16, 20, 21)

[21] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in*

Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc. (page 32)

[22] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition. (page 5)

[23] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. (page 43)

[24] Heinly, J., Schöunberger, J. L., Dunn, E., and Frahm, J. (2015). Reconstructing the world* in six days. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3287–3295. (page 1)

[25] Holzmann, T., Maurer, M., Fraundorfer, F., and Bischof, H. (2018). Semantically aware urban 3d reconstruction with plane-based regularization. In *European Conference on Computer Vision (ECCV)*. (page 28, 31, 32)

[26] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. (page 27, 28, 29, 36, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 99, 101, 105, 108)

[27] Kim, B.-S., Kohli, P., and Savarese, S. (2013). 3d scene understanding by voxel-crf. In *International Conference on Computer Vision (ICCV)*, ICCV '13, pages 1425–1432, Washington, DC, USA. IEEE Computer Society. (page 33)

[28] Kim, Y. M., Mitra, N. J., Yan, D.-M., and Guibas, L. (2012). Acquiring 3d indoor environments with variability and repetition. *ACM Transactions on Graphics*, 31(6):138:1–138:11. (page 28)

[29] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980. (page 45)

[30] Kurbiel, T. and Khaleghian, S. (2017). Training of deep neural networks based on distance measures using rmsprop. *CoRR*, abs/1708.01911. (page 45)

[31] Lepetit, V., Moreno-Noguer, F., and Fua, P. (2008). Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155. (page 12)

[32] Li, Y., Dai, A., Guibas, L., and Nießner, M. (2015). Database-assisted object retrieval for real-time 3d reconstruction. In *Computer Graphics Forum*, volume 34. Wiley Online Library. (page 2, 28, 29, 31, 36)

[33] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA. ACM. (page 16)

[34] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 11)

[35] Mitra, N. J., Guibas, L. J., and Pauly, M. (2006). Partial and approximate symmetry detection for 3d geometry. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 560–568, New York, NY, USA. ACM. (page 28)

[36] Nister, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770. (page 12)

[37] Okutomi, M. and Kanade, T. (1993). A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363. (page 14)

[38] Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., and Guibas, L. J. (2008). Discovering structural regularity in 3d geometry. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 43:1–43:11, New York, NY, USA. ACM. (page 28)

[39] Reed, S. E., van den Oord, A., Kalchbrenner, N., Colmenarejo, S. G., Wang, Z., Belov, D., and de Freitas, N. (2017). Parallel multiscale autoregressive density estimation. *CoRR*, abs/1703.03664. (page 42)

[40] Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 33, 109)

[41] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407. (page 23)

[42] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*. (page 11)

[43] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747. (page 45)

[44] Schönberger, J. L., Zheng, E., Frahm, J.-M., and Pollefeys, M. (2016). Pixelwise view selection for unstructured multi-view stereo. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *European Conference on Computer Vision (ECCV)*, pages 501–518, Cham. Springer International Publishing. (page 16, 53, 91)

[45] Schöps, T., Schönberger, J. L., Galliani, S., Sattler, T., Schindler, K., Pollefeys, M., and Geiger, A. (2017). A multi-view stereo benchmark with high-resolution images and

multi-camera videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 3, 53, 59, 63, 64, 91, 92, 108)

[46] Sipiran, I., Gregor, R., and Schreck, T. (2014). Approximate symmetry detection in partial 3d meshes. *Comput. Graph. Forum*, 33(7):131–140. (page 28)

[47] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic scene completion from a single depth image. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 3, 33, 34, 35, 36, 37, 47, 52, 55, 56, 57, 58, 60, 65, 85, 86, 87, 88, 89, 90, 105, 107, 108, 109)

[48] Sorkine, O. and Cohen-Or, D. (2004). Least-squares meshes. In *Proceedings of Shape Modeling International*, pages 191–199. IEEE Computer Society Press. (page 27)

[49] Strecha, Tuytelaars, and Gool, V. (2003). Dense matching of multiple wide-baseline views. In *International Conference on Computer Vision (ICCV)*, pages 1194–1201 vol.2. (page 14)

[50] Sun, J., Zheng, N.-N., and Shum, H.-Y. (2003). Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787–800. (page 14)

[51] The MathWorks, Inc. (2017). Matlab R2017a. `https://de.mathworks.com/products/matlab.html`. (page 64)

[52] Thrun, S. and Wegbreit, B. (2005). Shape from symmetry. In *International Conference on Computer Vision (ICCV)*, ICCV '05, pages 1824–1831, Washington, DC, USA. IEEE Computer Society. (page 28, 30, 32, 37)

[53] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000). Bundle adjustment — a modern synthesis. In Triggs, B., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg. Springer Berlin Heidelberg. (page 13)

[54] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016). Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328. (page 42)

[55] Wu, J., Zhang, C., Zhang, X., Zhang, Z., Freeman, W. T., and Tenenbaum, J. B. (2018). Learning shape priors for single-view 3d completion and reconstruction. *CoRR*, abs/1809.05068. (page 33)

[56] Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. In *ICLR*. (page 25, 47, 48, 68)

[57] Zeng, H., Wu, J., and Furukawa, Y. (2018). Neural procedural reconstruction for residential buildings. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *European Conference on Computer Vision (ECCV)*, pages 759–775, Cham. Springer International Publishing. (page 33)

[58] Zhang, J., Zhao, H., Yao, A., Chen, Y., Zhang, L., and Liao, H. (2018). Efficient semantic scene completion network with spatial group convolution. In *European Conference on Computer Vision (ECCV)*. (page 33)

[59] Zhang, Y., Song, S., Yumer, E., Savva, M., Lee, J.-Y., Jin, H., and Funkhouser, T. (2017). Physically-based rendering for indoor scene understanding using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 109)

[60] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334. (page 9, 10)

[61] Zhao, W., Gao, S., and Lin, H. (2007). A robust hole-filling algorithm for triangular mesh. In *2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 22–22. (page 27)

[62] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2016). Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*. (page 3, 48, 54, 109)

[63] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2017). Scene parsing through ade20k dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 3, 48, 54, 109)

[64] Zhou, Q.-Y. and Koltun, V. (2013). Dense scene reconstruction with points of interest. *ACM Trans. Graph.*, 32(4):112:1–112:8. (page 16, 42)

[65] Zhou, Q.-Y., Park, J., and Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*. (page 18, 42, 52, 54, 57, 85, 91, 104)