3D Scene Reconstruction from a Single Viewport

Maximilian Denninger$^{1,2[0000-0002-1557-2234]}$ and
Rudolph Triebel$^{1,2[0000-0002-7975-036X]}$

^{1} German Aerospace Center (DLR), 82234 Wessling, Germany
{maximilian.denninger, rudolph.triebel}@dlr.de
^{2} Technical University Munich (TUM), 80333 Munich, Germany

Abstract. We present a novel approach to infer volumetric reconstructions from a single viewport, based only on an RGB image and a reconstructed normal image. To overcome the problem of reconstructing regions in 3D that are occluded in the 2D image, we propose to learn this information from synthetically generated high-resolution data. To do this, we introduce a deep network architecture that is specifically designed for volumetric TSDF data by featuring a specific tree net architecture. Our framework can handle a 3D resolution of 512^3 by introducing a dedicated compression technique based on a modified autoencoder. Furthermore, we introduce a novel loss shaping technique for 3D data that guides the learning process towards regions where free and occupied space are close to each other. As we show in experiments on synthetic and realistic benchmark data, this leads to very good reconstruction results, both visually and in terms of quantitative measures.

Keywords: Scene Reconstruction · 3D from Single Images · Space Compression

1 Introduction

One of the most fundamental tasks for visual perception systems - both natural and artificial - is the acquisition of the 3D environment structure from a given visual input, *e.g.* an image. The main challenge of this task is that this visual input is usually the result of a projection mapping from the 3D environment onto a lower-dimensional manifold and that this mapping is not bijective, *i.e.* it can not be inverted. Thus, mapping back to the 3D environment, which is denoted as the 3D reconstruction task, is an inverse problem. Nevertheless, humans and other living beings are capable of generating reasonably accurate representations of the true 3D structure, even when provided with only a single visual stimulus, which means that they are able to recover the information that was lost during the projection process. The key resource to achieve this are *experiences* made earlier, and this is our primary motivation to resort to machine learning techniques to solve the 3D reconstruction task for artificial systems such as robots, only from single images. The potential applications of such a technique are manifold. While most current approaches generating 3D environment models rely on the fusion of many images, which are acquired at different viewpoints. The single

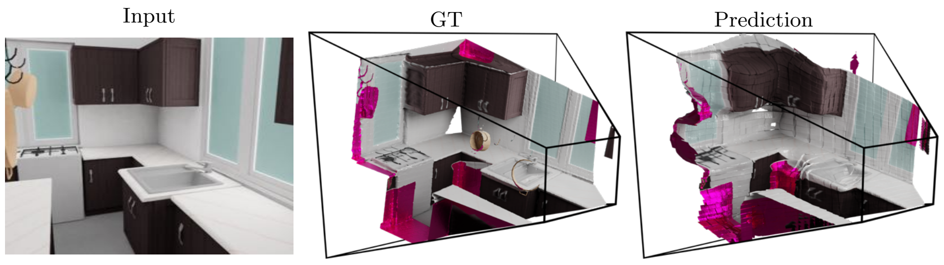

Fig. 1. 3D reconstruction from a single RGB image and a normal image (not shown). On the left the input color image is shown, in the middle the 3D ground truth scene and to the right our reconstruction is depicted. Note especially the reconstruction quality in areas where the 2D view caused occlusions (shown in pink).

image reconstruction has the benefit of producing a 3D representation fast and without having to move the camera. This can be very useful for mobile robots that need to explore unknown environments as it reduces the risk colliding with obstacles, and it can lead to denser and more accurate maps from less input data. Furthermore, it provides the ability to plan paths through the environment, *e.g.* to avoid occluded obstacles, even if only a single view is given.

The enormous attractiveness of these capabilities yet comes with a number of major challenges that need to be resolved. First and foremost, the curse of dimensionality is the major hurdle when dealing with 3D data, both in terms of memory requirements and regarding the algorithmic formulation. To address these issues, we propose both a novel network architecture that can reason efficiently on high resolution 3D data and a fast and efficient technique to generate and represent volumetric training data. We also introduce a specifically designed loss function for the training process. In summary, our main contributions are:

– A tree net architecture to reconstruct volumetric data.
– An autoencoder to efficiently compress TSDF volumes.
– A dedicated loss shaping technique for 3D reasoning.
– A framework to generate TSDF volumes from meshes.

In the following sections, we describe each of these contributions in more detail, after discussing previous works that are related to ours.

2 Related work

The four research topics that are most related to our work are shape completion, segmentation, depth reconstruction, and full scene reconstruction. In the following, we show the relations of these works to ours.

Shape completion There is some prior work that focusses on the reconstruction of single objects [18, 25, 26]. In particular, Wu *et al* [29] introduced 3D ShapeNets, which apply a deep belief network to a given shape database. This network can complete and generate shapes and also repair broken meshes. Later, Wu *et al* [27] used an autoencoder to convert color into normals and depth, and ultimately to a 3D scene with a resolution of 128^3. They extended this using an adversarially trained deep naturalness regularizer, which provides a solution to the problem of blurry mean outputs. In our approach, we also avoid this by training an autoencoder, which we use to compress the TSDF volumes. Tatarchenko *et al* [24] use an octree generative network to reconstruct objects and scenes. However, this relies on the assumption that the coarse prediction steps can always find even small details, which is often not justified. Therefore, we use a block-wise compression to benefit both from a high resolution and an efficient representation. The 3D-EPN approach introduced by Dai *et al* [4] can predict object shapes based on sparse input data. Park *et al* [16] showed an interesting approach, where instead of reconstructing a volume, they reconstruct for given points a certain SDF value. This however, struggles to generalize for complete scenes because of the missing spatial link between the input image and the output. Matryoshka Networks fuse multiple nested depth maps to a single volume [17], but the same struggle of generalizisation to full scenes appears.

Segmentation The reconstruction of scenes is also sometimes covered in the field of semantic segmentation of 3D volumes. Using semantic information the reconstruction task can be improved, as the network knows for some objects what it is reconstructing [8, 21]. Song *et al* [21] showed in their work how to use pure depth data to generate semantic segmented volumetric predictions. Nonetheless, their work requires the use of a depth camera and the knowledge of all appearing objects in the scene to correctly classify them, whereas in our approach, we are free of such limitations. Additionally, instead of using a resolution of $240 \times 144 \times 240$, we work with 512^3. This is 16 times more data. Dai *et al* [3] showed how to complete a scene in several iterations on different resolutions by also predicting segmentation masks. However, their approach requires a rough 3D scene model, whereas we can start only with an RGB image. Also, our main focus is on scene reconstruction with as little extra knowledge as possible, thus semantic segmentation is not considered here.

Depth reconstruction In contrast to our approach, a large amount of prior research has been devoted to depth reconstruction on mono or stereo images. For example, multi-scale CNNs were used by Eigen *et al* [6] to generate robust depth estimations. A combination of CNN with CRF based regularization was shown by Liu *et al* [13,14], where they jointly learn CNN and CRF. Ma *et al* [15] showed how to generate depth images based solely on a few depth points and an input image. Kim *et al* showed that going from RGB images to TSDF works [12]. However, these are only 2.5D images of the scene and not a complete reconstruction of the occupation of the 3D space, which is the main objective of our approach.

Full scene reconstruction This area is mostly related to our work. For example, Firman *et al* [7] introduced Voxlets, which use random forests to predict unknown voxel neighborhoods. However, their approach only works on the local

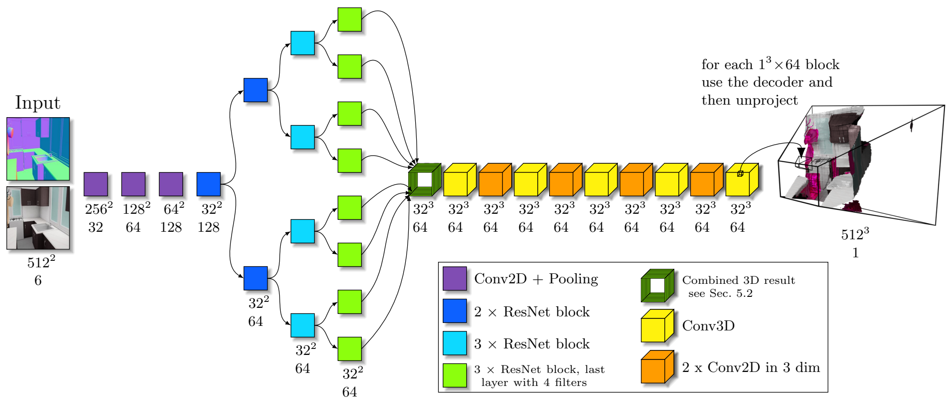

Fig. 2. Compressed form of our proposed architecture. On the left, an exemplary RGB and normal image are input to the network. From this, several convolution and pooling operations are done down to a size of 32^2. Then, we split the path of the network in two, where one represents the front and the other one the rear part of the depth channel. This split is done two more times, and the resulting depth slices are combined into a 3D structure with 64 channels. On that, we perform some 3D convolutions and use the autoencoder to decode the output of the tree net. Note that, our real model has one tree layer more, where the second layer is repeated once more.

neighborhood, which limits the prediction of bigger structures. There are methods, which reconstruct scenes by placing preexisting CAD models [11]. However, these are limited to the known CAD models, where we try to learn general shapes. In contrast to Silberman *et al* [20] who fill incomplete scenes using a novel CRF method, we build on the assumption that scenes are piece-wise planar and use deep learning to reconstruct a scene from one image. Also, many prior works focus on the reconstruction of small table scenes, where a majority of the objects are partly or entirely known. We believe the main reason for this is the lack of datasets to evaluate on. In order to not limit ourselves to such table scenes, we use the synthetic SUNCG dataset [21] and also the real-world Replica-dataset [22] to generate TSDF volumes on which we can measure our performance. Furthermore, we rely on the toolchain named BlenderProc [5] to generate realistic color and normal images.

3 Problem Description and General Approach

We formulate our problem as finding a mapping from 2D image coordinates $\mathbf{x}_{c,d} = (x_{c,d}, y_{c,d})$ to 3D scene coordinates $\mathbf{x}_s = (x_s, y_s, z_s)$. Our input is an RGB image $I_c\colon \Omega_c \to [0, 255]^3$ and a normal vector image $I_n\colon \Omega_d \to [-1, 1]^3$, where $\Omega_c \subset \mathbb{R}^2$ and $\Omega_d \subset \mathbb{R}^2$. The output is a high-resolution 3D truncated signed distance field (TSDF) $V\colon \Omega_v \to [-\sigma_{tsdf}, \dots, \sigma_{tsdf}]$ where $\Omega_v = \{0, \dots, 511\}^3$. This voxel grid represents free space with positive values and occupied areas with negative values. Absolute values are the distances to the closest surface.

To perform the 3D reconstruction, we propose to train a specifically designed deep network architecture on synthetic data, which can then be used to infer 3D reconstructions from new test images. An overview of our architecture is shown in Fig. 2, where the details will be presented in the following sections. Note that the input of this network consists of an RGB image and a normal vector image. Our motivation to use surface normals as an additional input is to provide continuity information so that planar surfaces can be reconstructed more precisely. Here, we take inspiration from Zhang *et al* [31] who also used normals for depth generation. In this paper, we focus on the 3D reconstruction part, and we use normals from a simulation pipeline named BlenderProc [5], which can generate RGB and normal images on the SUNCG dataset, as well as normal images on the Replica-dataset [21, 22]. Throughout this paper, such renderings were used to obtain training data, while during testing we use a U-net architecture [19] trained on soley SUNCG to generate normals (see Sec. 7).

For the design of our training procedure, we had to face three major challenges. First, we had to find a way to efficiently produce and represent the output training samples, which consist of voxel grids with $512^3 = 134,217,728$ voxels. Second, we had to design a network architecture that can represent 3D spatial information in hierarchical form. And third, we needed to find an appropriate loss function for the training process. All three parts will be described next.

4 Generating Synthetic 3D Training Data

Our output data consists of high-resolution 3D TSDF voxel grids. TSDF volumes offer in comparison to meshes or point clouds a dense representation, providing a deterministic reconstruction target, which we can align with the input domain. TSDF grids are widely used in computer vision, and there are several approaches to compute these volumes fast. However, most of them use approximations, because an accurate result is usually not needed and their test scenes have a smaller resolution than 512^3 [28]. We propose three steps to achieve an accurate result on such a resolution. First, we simplify the reconstruction task by aligning the output voxel grids with the camera frame and not the world frame, which is explained next. Then, we employ a fast algorithm to compute a TSDF voxel grid from a given 3D scene (see Sec. 4.2). In the end, we use a compression algorithm to store the voxel data with comparably low memory requirements (see Sec. 4.3).

4.1 Viewport Alignment

An important distinction between our work and most others in learning-based 3D reconstruction [4,27] is that for our training procedure, we use input RGB images and 3D voxel grids that are aligned within the same coordinate frame, namely the camera frame. For that, we transform vertices used for training from world coordinates $\mathbf{x}_w$ into the camera frame using the camera matrix C, *i.e.* $\mathbf{x}_s = C\mathbf{x}_w$. Then, a perspective projection P is applied to $\mathbf{x}_s$ such that the camera frustrum is mapped to a cubical 3D volume. The resulting projected points $\mathbf{x}_p = P\mathbf{x}_s$ are

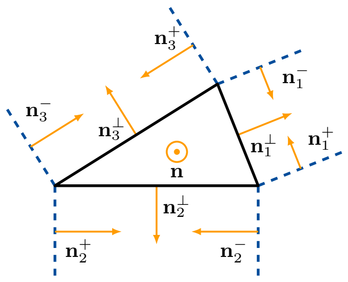

Fig. 3. Triangle t with all normals in orange that are used to efficiently compute $d(\mathbf{v}, t)$. The corresponding orthogonal planes are shown in blue and dashed.

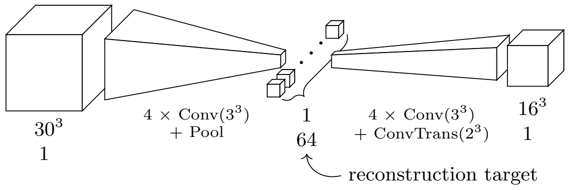

Fig. 4. Approximated structure of the autoencoder to compress TSDF volumes, which is applied on each 16^3 block plus padding ($=30^3$) on the 512^3 input space. The result of this is the encoded latent values in the middle, which we use as the reconstruction target in the tree network.

then in the range $[-1, 1]^3$. Now we voxelize this 3D volume with a resolution of 512. Then, the center point $\mathbf{x}_e$ of each voxel can be computed from its index $\mathbf{v}$ as $\mathbf{x}_e = (\mathbf{v}/512) \cdot 2 - 1$. The center $\mathbf{x}_e = (x_e, y_e, z_e)$ can now be directly mapped to the 2D image, which also has a resolution of 512, *i.e.* $\mathbf{x}_c = (x_e, y_e) \cdot 256 + 256$. Similarly, the inverse mapping from pixels $\mathbf{x}_c$ to points $\mathbf{x}$ in the 3D grid is done by setting $x = x_c, y = y_c$ and $z = (\pi(\mathbf{x}_c) - d_{min}) \,/\, (d_{max} - d_{min}) \cdot 2 - 1$, where π is the projected depth of the pixel position $\mathbf{x}_c$, and d_{min} and d_{max} are predefined values for the minimal and maximal depth range within which the voxel grid is defined. In our implementation, we use $d_{min} = 1\,m$ and $d_{max} = 4\,m$. In contrast to this inverse mapping, we predict the TSDF values along the camera ray.

This means that in our 3D reconstruction of the occupancy, we can directly link the input pixel values with the occupancies along the camera rays. This way, we can learn the transformation from a 2D image to a 3D space. For visualization, we project them back from the cube to the camera frustum.

4.2 Fast Generation of TSDF Voxel Data

To produce synthetic 3D training samples, we start with a set $\mathcal{T}$ of 3D triangles, which we map into the camera frame using a predefined transform $\tau = P \cdot C$. Then, for the center point $\mathbf{x}$ of each voxel $\mathbf{v} = (v_x, v_y, v_z)$ we need to compute the distance d_x to the closest point on a triangle $t \in \mathcal{T}$ and truncate the absolute distance at a maximum value σ_{tsdf}, i.e.

$$V[\mathbf{v}] = d_x = \max\left(-\sigma_{tsdf}, \min\left(\sigma_{tsdf}, \min_{\forall t \in \mathcal{T}} \{d(\mathbf{x}, t)\}\right)\right), \forall \mathbf{v} \in \Omega_v. \qquad (1)$$

To achieve that, we developed a very fast technique that transforms the triangles and computes $d(\mathbf{x}, t)$ for each voxel $\mathbf{v}$. It uses a combination of flood filling, octrees, and a fast distance computation. With this, we can process the 134 million voxels in the order of seconds. A more detailed description

Algorithm 1 In this distance calculation algorithm we use three different variable colors, which correspond to the main, the edge and the border planes.

```
1: procedure CalculateDistance($Point\ p$)
2:     $plnDist \leftarrow mainPln.distTo(p)$
3:     for $nr \in [1, 2, 3]$ do
4:         if $edgePln[nr].distTo(p) < 0$ then            ▷ Outside, check border planes
5:             if $borderPln[nr][1].distTo(p) < 0$ then          ▷ Dist to left point
6:                 return $sgn(plnDist) \cdot \|p - borderPln[nr][1].p\|_2$
7:             else if $borderPln[nr][2].distTo(p) < 0$ then    ▷ Dist to right point
8:                 return $sgn(plnDist) \cdot \|p - borderPln[nr][2].p\|_2$
9:             else                                                ▷ Dist to edge
10:                return $sgn(plnDist) \cdot edgeLine[nr].distTo(p)$
    return $plnDist$
```

of all individual steps is given in the supplementary material, and we also refer to our implementation, which is online: https://github.com/DLR-RM/SingleViewReconstruction. The key component here is the fast computation of $d(\mathbf{x}, t)$ using modern hardware. For this, we first precompute 10 vectors for each triangle t, namely the normal vector $\mathbf{n}$ of the triangle plane $\mathfrak{P}$, the vectors $\mathbf{n}^{\perp}$ that are orthogonal to the edges of t and lie inside $\mathfrak{P}$, as well as the vectors $\mathbf{n}^{+}$ and $\mathbf{n}^{-}$ that are parallel to the edges of t (see Fig. 3). Next, we compute the distance $d(\mathbf{x}, \mathfrak{P})$ between $\mathfrak{P}$ and $\mathbf{x}$ and check whether its projection onto $\mathfrak{P}$ is inside t, using the normals $\mathbf{n}^{\perp}$. If so, $d(\mathbf{x}, t)$ is equal to $d(\mathbf{x}, \mathfrak{P})$, otherwise $\mathbf{x}$ is closer to an edge or vertex than to the surface. For the final check, we use the normals $\mathbf{n}^{+}$ and $\mathbf{n}^{-}$ of the planes. If the distances of the planes are positive, then the distance can be calculated towards the edge, and if one of them is negative, the closest distance is to one of the points, see Algorithm 1. This distance calculation has to be done for all voxels and all polygons. Finally, we quantize the TSDF volume to 16 bit and compress them with gzip, which reduces the size by a factor of ten.

4.3 Spatial Compression

A straightforward implementation of our high-resolution TSDF volume V with 512^3 voxels would require 536.87 MB per scene, which renders the training process on current hardware infeasible. Therefore, we employ a block-wise compression of V to a size of $64{\times}32^3$. This results in 8.38 MB per scene with a compression factor of 64. The compression is done with an autoencoder as shown in Fig. 4. First, we use 3D convolutions in combination with valid padding on a larger input than the output, thereby shrinking the input size from 30^3 to 64 and then up again to 16^3. Second, we balance the input to the autoencoder so that the much more likely empty voxels are mostly removed to focus on the ones with surfaces. Third, we add loss shaping to focus on the reconstruction of the surfaces, with:

$$loss\,(x, y) = \|x - y\|_1 \cdot \left(1 + \mathcal{N}\left(0, \frac{\sigma_{tsdf}}{4}\right)(y) \cdot \frac{4}{\sigma_{tsdf}}\right) \qquad (2)$$

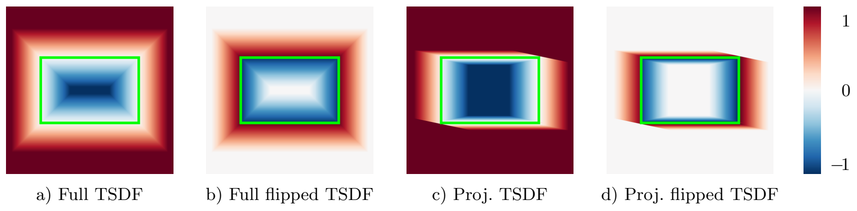

Fig. 5. The two on the left represent a full TSDF, the two on the right are projected TSDF volumes, which uses a camera projecting beams into the scene. Both also have a flipped version, where the empty and occupied space is zero.

Here x is the prediction, and y is the label. The scaling value of the Gaussian was determined experimentally. We use a complete TSDF, not a projected or a flipped TSDF, see Fig. 5. We found that a projected TSDF volume can generate hard cuts in the resulting output volumes, which means that moving the input by one voxel generates a big loss at these boundaries. With full TSDF volumes this does not happen, so that the network can learn the three dimensional representation of an object in space. Auto encoders trained on the flipped TSDF performed considerably worse after training, we didn't investigate this further.

5 Proposed Network Architecture

The major challenge of our framework is to represent the 3D occupation information of the voxel grid in a deep neural network. In order to solve this, we propose a special architecture that is based on a tree structure, which helps us to transform a 2D image into a volumetric 3D space, which is described next. To perform the 3D reconstruction task from a single image, we designed a network architecture that can split the input data along the depth dimension. One way of doing this is to use the feature channels as the depth dimension at some point within the network. This, however, requires the network to transform the 2D input to 3D in one step, which failed in our experiments for complex scenes.

5.1 Tree Network

To address this problem, we propose a tree architecture, where each level in the binary tree splits the depth dimension into a front and back part. That means the first tree node splits the scene into foreground and background, where those are defined by the distance to the camera. In Sec. 4.1, we showed that our input images are aligned with the output frame, which makes this splitting possible. We repeat the splitting process three more times so that the leaves of the tree contain small slices of the depth dimension while still representing the full spatial resolution. These slices are then combined into a 3D volume and processed by further 3D convolutions to remove small artifacts.

In Fig. 6 such a tree is depicted. The first node is fed with the output of some convolutional and pooling layers to scale the input from 512^2 down to

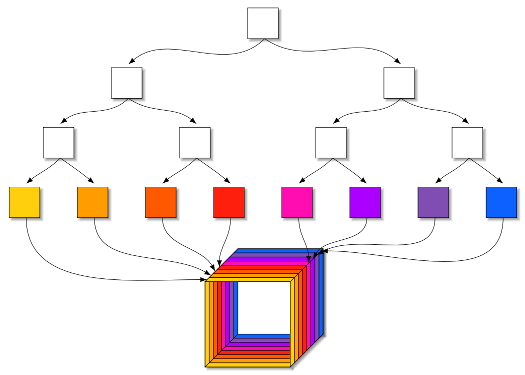

Fig. 6. The basic tree architecture which generates a 3D volume based on a 2D input. Each layer is thought to perform a split through the depth dimension, and in the end all single paths of the tree are concatenated to create the third dimension.

32^2. Then, in this image, it is split three times, and the resulting colored leaves are combined into a 3D tensor. The resulting feature channels in the leaf nodes can then only have four channels to obtain the desired depth of 32. Here each path builds different CNN parts to learn the size of different objects at different scales. Instead of a single track sequential model, where at some point the feature channels could be mapped to the depth dimension, our network has several layers to capture the relationship between the input and the depth dimension.

5.2 Multipath

The proposed tree network has a bottleneck when forming the 3D volume from the 2D features channels. As there are only 32 leaf nodes with just two feature channels each, the combination leads only to one 3D volume, whereas the compressed output has 64. We address this by increasing the output of the leaf nodes to 128 and then create 64 3D volumes out of it. This is achieved by splitting up each leaf node's feature results and using two feature channels per created 3D volume. In Fig. 7, we show this for two 3D volumes with only eight leaf nodes, which means that each leaf node has a eight feature channels. Thereby, the first half of each node is used in the left 3D volume and the second in the right.

5.3 General architecture

Inspired by He *et al* [10] we use ResNet blocks, where additionally each block uses dilated convolution in an inception fashion [23, 30]. This means that the input per ResNet step is given to three different convolutional blocks, where the dilation rate differs. This dilation inception step was done twice in each of the ResNet blocks. In our experiments a dilation rate of $1, 2, 4$ with a split of $50\%, 25\%, 25\%$ over the desired filter channels performed best. These three are

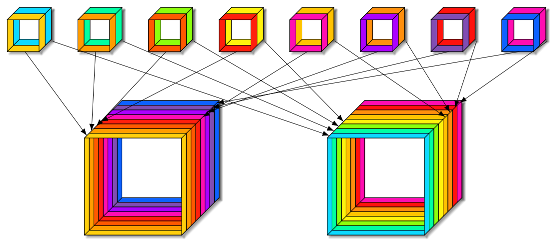
Fig. 7. The upper row represents the leaves of our tree architecture. In this example each node has eight feature channels, which are evenly split over the resulting volumes.

then concatenated again and used as an input to the next layer. Our tree uses two ResNet blocks in the first two layers and three in the last two layers.

After performing the multipath joining explained in Sec. 5.2, we perform several 3D convolutions on the joined result. This smoothes out errors that were introduced by paths in the tree, which performed worse than the others. We use 9 layers of a sequence of normal convolutions and separable convolutions to save memory [2]. We alternate between one normal 3D convolution followed by two 2D convolutions performed in all three axes. All of them use 64 filters, where we split in each over four paths. These also use again dilations with rates of $1, 2, 4$, and 8, where the splitting for the filters is $32, 16, 8$, and 8.

6 Loss Shaping

An essential part of our pipeline is our loss shaping, which we use to focus the attention of the network to parts of the TSDF volume that are more relevant for a correct reconstruction. We distinguish two kinds of loss shaping, one is related to the voxel space and one to the tree net structure. Both are described next.

6.1 Output loss shaping

When we know where the surface in the TSDF volume is, we can increase the loss around and on the surface by a factor $\sigma_{Surface}$ to make sure that these encoded latent values are correctly regressed. The same is done for the free space before an object occurs. This value σ_{Free} is selected to be smaller than $\sigma_{Surface}$. Additionally, the free space behind objects receives an increased loss factor to make sure that those areas, which are reachable but not visible from the camera point of view, are reconstructed well. The distance to the closest visible and free voxel determines the strength of the factor. It decreases from σ_{Free} to a fixed value of $\sigma_{NonVisibleFree}$. This decline is done at most for 7% of the space size, which we found gives sufficiently good results.

In Fig. 8, the loss factors for a 2D scene with two objects (in blue) are shown. The camera is on the left side of the frame and is oriented towards the right. All voxels with circles in them are free. The stars are used for the area around and below the surface of an object, and the rectangles depict the areas, which are not

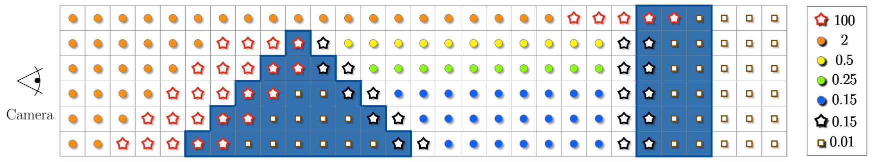

Fig. 8. In this top-down 2D map of a scene two objects are depicted one on the left as a blue pyramid and one as a wall on the right, which give the used loss factors as seen from the left. In the legend we show the weight values that are used in our approach.

reachable. It is important to note here that the factor for the first hit onto an object is 100 to make sure that this surface is regressed correctly. The surfaces behind this only receive an increased factor if they can be reached from the free space. To determine these back surfaces we used a flood filling algorithm. Using this loss shaping, our network is able to focus on the more relevant parts of the reconstruction, and neglects the parts, which are deemed less important. This improves the reconstruction performance, see Sec. 7.

6.2 Tree loss shaping

To speed up the training, we enforce the splitting of the depth dimension already in the tree by comparing the output of each node with the average of the corresponding depth range. This means for the first split we take the left node result and branch into a 1×1 convolution to change the number of feature channels so that they match the target output, see Fig. 9. Then we take the target output and use only the first half of it, average it in the depth dimension and compare this slice with the branched output. This process is repeated in every node, where every time the corresponding depth slice is averaged and compared with the branched version. All these losses are combined and weighted, where the second layer in the tree receives a lower loss than the leaf nodes. We used the values $[0.2, 0.3, 0.5, 0.8]$ from top to bottom for our tree, which has a height of 5. Finally, we scale this weighted tree value with a factor of 0.4 and add it to the final loss. Additionally, before reducing the difference between all these losses, we multiply our averaged loss map introduced in Sec. 6.1. This again helps to focus on the relevant surfaces of the TSDF volume.

7 Experiments

7.1 Test setup

The evaluation of our approach is first done on the synthetic dataset SUNCG [21], from which we already used one split for training. The second evaluation is done on the real-world Replica-dataset [22], which is the only dense, hole-free dataset available. It stands in contrast to datasets like Matterport 3D [1], where holes introduced through the scanning process have not been filled manually.

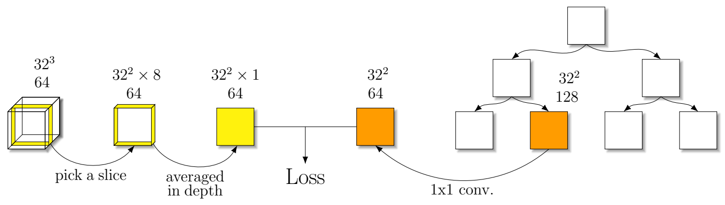

Fig. 9. For each node the corresponding depth layers from the output are averaged in the depth and then compared to a reshaped tensor from the node. This already enforces in the tree a sense of the encoded 3D structure.

As described in Sec. 4.2 for the training we first generate the TSDF volumes for the sampled camera positions from SUNCG. Then we create the corresponding loss volumes and finally the RGB images using BlenderProc [5]. We tested both with the normal generation and with the synthetic normal images to see how our network can deal with the limitations of the normal generation. All tests were performed with models that were exclusively trained on the generated data from BlenderProc on the SUNCG dataset. For the training, we used around 130,000 image pairs. We did not finetune on the Replica-dataset, nor did we finetune with the generated normals to show the lower bound of this approach. The reconstruction network was evaluated on 500 image scene pairs from the SUNCG dataset. For the Replica-dataset, we sampled as in SUNCG ten cameras per scene, which resulted in 180 image pairs. The creation of the encoded scene from a color and normal image takes around 0.11 seconds. However, the reconstruction to a full scene takes around 5.1 seconds, with the decoder.

7.2 Qualitative results

In Fig. 10, we show some qualitative results on the real-world Replica-dataset. As in previous images, the areas in pink are invisible to the camera and did not get assigned a color. For failed reconstructions, these areas are too far away from the true surface to get the correct color. The scene in the lower left corner, for example, was reconstructed well, without ever seeing this room before nor being able to recognize that this texture belongs to a bed. It also indicates that it learned some kind of semantic understanding of this object type, without us providing the additional label “bed”. In the right lower corner in Fig. 10 is in contrast to that a failed reconstruction, as the network could not reconstruct the surface of the thin chair and nearly hidden table.

7.3 Quantitative results

We evaluate the precision, recall, and IOU over the occupied voxel on both datasets. This shows directly how much of the space was correctly classified as occupied, for that the predicted TSDF volumes are converted into binary occupation grids. This process means that some of the resolution is lost. Because of that we also evaluate the mean and RMS Hausdorff distance (HD) [9], between

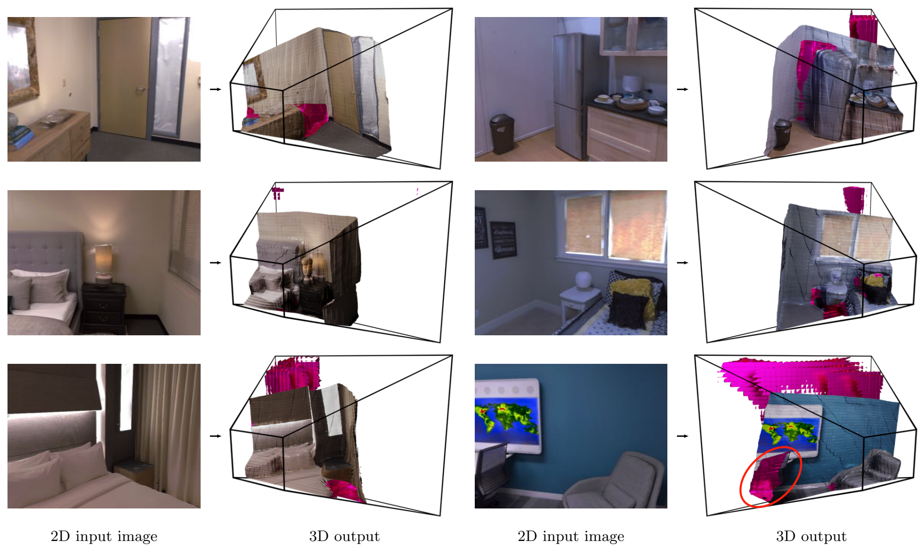

Fig. 10. Results on the Replica-dataset for six scenes. Only the generated normal and color images were provided to create the full scene reconstruction. In the top left corner, our network could separate the commode from the wall and detect the end of it, too. Areas in pink are so far away from the true reconstruction to assign a color. The red ellipse highlights, the failed reconstruction of the thin chair and table.

the true and the predicted mesh. This mean is calculate by averaging over the distances of each true mesh vertex to the closest point in the test mesh.

Table 1 shows the results for four different cases, where two are with SUNCG and two with the Replica-dataset. Both are tested with the normals from BlenderProc (woNG) and with the generated once from the U-Net (wNG). We tested with four different configurations, first we alter the amount of layers in the tree from four to six, where five is our default. By only copying or removing the second layer in Fig. 2 and also report results when no loss shaping was used.

Even though our network has never seen real scenes, the performance on the real-world Replica-dataset is better than on the SUNCG dataset. As our network performs particularly well in predicting large structures, which are more commonly found in Replica, so the performance on Replica is higher. As a lot of the SUNCG scenes are cluttered with thin small objects. This also relates to the fact that the scenes in the Replica-dataset are more structured than in SUNCG. We observed that it might happen that unusual combinations of objects are randomly placed in a SUNCG scene. It is also interessting to see that not using the loss shaping, introduced in section 6, increases the IOU performance, however, decreases strongly the HD performance, so that rough shapes can still be reconstructed, but the finer details are mostly lost.

In order to demonstrate the relative performance of our approach, we included the results from Firman *et al* and Song *et al* [7, 21]. Note here, that they use output spaces with less resolution and other datasets. They did not report

Table 1. The comparison on the synthetic SUNCG dataset and the real-world Replica-dataset. It was tested with the ground truth (woNG) and the generated normals (wNG).

Dataset	Method	Precision	Recall	⌀IOU	⌀HD	RMS HD
SSCNet joint [21] SUNCG+NYU		75.0	96.0	73.0	-	-
Voxlets [7]		58.5	79.3	65.8	-	-
SUNCG woNG	default	**85.05**	72.96	65.10	0.0416	0.0670
SUNCG woNG	height 4	84.47	76.59	68.05	0.0395	0.0644
SUNCG woNG	height 6	81.08	78.06	66.58	**0.0390**	**0.0620**
SUNCG woNG	no loss sh.	83.51	**81.21**	**70.49**	0.0745	0.1117
SUNCG wNG	default	**83.65**	69.65	61.56	0.0509	0.0794
SUNCG wNG	height 4	83.06	73.41	64.41	0.0488	0.0769
SUNCG wNG	height 6	80.93	74.19	63.48	**0.0487**	**0.0750**
SUNCG wNG	no loss sh.	82.70	**77.80**	**67.61**	0.0835	0.1232
Replica woNG	default	**86.19**	84.34	73.97	**0.0387**	0.0521
Replica woNG	height 4	85.31	91.40	**78.76**	0.0393	**0.0518**
Replica woNG	height 6	81.67	93.39	76.81	0.0457	0.0569
Replica woNG	no loss sh.	83.33	**94.03**	78.36	0.0614	0.0766
Replica wNG	default	**87.75**	72.94	65.86	0.0562	0.0745
Replica wNG	height 4	84.59	80.00	69.40	**0.0518**	**0.0691**
Replica wNG	height 6	83.46	78.33	67.27	0.0563	0.0735
Replica wNG	no loss sh.	83.37	**88.38**	**73.49**	0.0766	0.0959

the HD for their reconstructions. Nonetheless, the given precision, recall, and IOU values indicate that our approach performs equally well, even though we do not have any depth data, nor do we do any semantic segmentation. Using our novel tree net architecture we can reconstruct scenes well without the additional information of depth or semantic segmentation.

8 Conclusion

We have demonstrated that the difficult task of reconstructing a full indoor scene based on just one single color image is possible. To achieve that, we introduced a tree net architecture that enables the splitting in different depth layers. We combined this with an autoencoder approach to increase the resolution of the used TSDF volumes. Furthermore, we showed the importance of loss shaping during training to focus the attention of the network on the relevant parts. For some applications, the quality of our results is likely to be sufficient, especially in the domain of map generation for mobile robot navigation.

We furthermore conclude that our 3D reconstruction approach is realized with a network that is solely trained on synthetic data, and it can still adapt to a real scenario. Finally, we showed that the complete scene reconstruction is possible without depth data or any auxiliary task like semantic segmentation.

References

1. Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3d: Learning from rgb-d data in indoor environments. International Conference on 3D Vision (3DV) (2017)
2. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1251–1258 (2017)
3. Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., Nießner, M.: Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4578–4587 (2018)
4. Dai, A., Ruizhongtai Qi, C., Nießner, M.: Shape completion using 3d-encoder-predictor cnns and shape synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5868–5877 (2017)
5. Denninger, M., Sundermeyer, M., Winkelbauer, D., Zidan, Y., Olefir, D., Elbadrawy, M., Lodhi, A., Katam, H.: Blenderproc. arXiv:1911.01911 (2019)
6. Eigen, D., Puhrsch, C., Fergus, R.: Depth map prediction from a single image using a multi-scale deep network. In: Advances in neural information processing systems. pp. 2366–2374 (2014)
7. Firman, M., Mac Aodha, O., Julier, S., Brostow, G.J.: Structured prediction of unobserved voxels from a single depth image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5431–5440 (2016)
8. Hane, C., Zach, C., Cohen, A., Angst, R., Pollefeys, M.: Joint 3d scene reconstruction and class segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 97–104 (2013)
9. Hausdorff, F.: Grundzüge der mengenlehre, leipzig. de Gruyter & Co **1927**, 1935 (1914)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
11. Izadinia, H., Shan, Q., Seitz, S.M.: Im2cad. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5134–5143 (2017)
12. Kim, H., Moon, J., Lee, B.: Rgb-to-tsdf: Direct tsdf prediction from a single rgb image for dense 3d reconstruction. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 6714–6720. IEEE (2019)
13. Liu, F., Shen, C., Lin, G.: Deep convolutional neural fields for depth estimation from a single image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5162–5170 (2015)
14. Liu, F., Shen, C., Lin, G., Reid, I.: Learning depth from single monocular images using deep convolutional neural fields. IEEE transactions on pattern analysis and machine intelligence **38**(10), 2024–2039 (2015)
15. Mal, F., Karaman, S.: Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 1–8. IEEE (2018)
16. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. arXiv:1901.05103 (2019)
17. Richter, S.R., Roth, S.: Matryoshka networks: Predicting 3d geometry via nested shape layers. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1936–1944 (2018)

18. Rock, J., Gupta, T., Thorsen, J., Gwak, J., Shin, D., Hoiem, D.: Completing 3d object shape from one depth image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2484–2493 (2015)
19. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI). LNCS, vol. 9351, pp. 234–241. Springer (2015), http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a, (available on arXiv:1505.04597 [cs.CV])
20. Silberman, N., Shapira, L., Gal, R., Kohli, P.: A contour completion model for augmenting surface reconstructions. In: European Conference on Computer Vision. pp. 488–503. Springer (2014)
21. Song, S., Yu, F., Zeng, A., Chang, A.X., Savva, M., Funkhouser, T.: Semantic scene completion from a single depth image. arXiv:1611.08974 (2016)
22. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., et al.: The replica dataset: A digital replica of indoor spaces. arXiv:1906.05797 (2019)
23. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
24. Tatarchenko, M., Dosovitskiy, A., Brox, T.: Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2088–2096 (2017)
25. Thanh Nguyen, D., Hua, B.S., Tran, K., Pham, Q.H., Yeung, S.K.: A field model for repairing 3d shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5676–5684 (2016)
26. Varley, J., DeChant, C., Richardson, A., Ruales, J., Allen, P.: Shape completion enabled robotic grasping. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2442–2447. IEEE (2017)
27. Wu, J., Zhang, C., Zhang, X., Zhang, Z., Freeman, W.T., Tenenbaum, J.B.: Learning shape priors for single-view 3d completion and reconstruction. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 646–662 (2018)
28. Wu, Y., Man, J., Xie, Z.: A double layer method for constructing signed distance fields from triangle meshes. Graphical models **76**(4), 214–223 (2014)
29. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1912–1920 (2015)
30. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. arXiv:1511.07122 (2015)
31. Zhang, Y., Funkhouser, T.: Deep depth completion of a single rgb-d image. arXiv:1803.09326 (2018)