NICE-SLAM with Adaptive Feature Grids via Voxel Hashing

Arjun Bhardwaj
ETH Zurich
abhardwaj@ethz.ch
arbhardwaj98@gmail.com

Tanmay Goyal
ETH Zurich
tgoyal@ethz.ch
tanmaygoyal98@gmail.com

Takahiro Miki
ETH Zurich
tamiki@ethz.ch
takahiro.miki1992@gmail.com

Abstract

Dense 3D Simultaneous Localization and Mapping (SLAM) has been an active area of research. Recently, neural implicit representations, where a neural network implicitly represents a 3D structure, have been used in learning-based SLAM methods. Although, these methods are able to generate smooth 3D structures, they do not scale to larger environments. This is because of the limited representational capacity of the individual neural networks. NICE-SLAM (Neural Implicit Scalable Enoding for SLAM) demonstrated large scale mapping by leveraging multi-level grid-based features using a pre-trained implicit decoder. These feature vectors are stored in a hierarchical grid structure and optimized based on the RGB-D measurements. However, this grid structure is defined beforehand and therefore the memory requirement increases as the size of the map grows. This limits the size of the environment to be mapped.

In this project, we propose to integrate dynamically allotted adaptive grids into the NICE-SLAM architecture to reduce its memory requirements. The dynamic allocation of voxels is achieved using a memory and speed efficient data structure known as voxel hashing. This can result in an increased scalability of NICE-SLAM to large outdoor scenes because of significant reduction in runtime and storage memory requirements. The code for integrated NICE-SLAM can be found at `https://github.com/arbhardwaj98/3dv_nice_slam`

1. Introduction

Accurately localizing a camera position and mapping a 3D structure is crucial for various fields such as robotics, navigation, or AR etc. The problem is called SLAM as it requires localizing and mapping at the same time. Several different methods have been proposed [10, 15, 11] in the scope of dense visual SLAM where they often use RGB or RGBD images. These traditional methods track the camera position and create a dense 3D map by optimizing a photometric and geometric error. They can generate a high-quality result, however, the map cannot capture unobserved regions such as behind an obstacle due to occlusion. On the other hand, humans can roughly estimate how the structure would be even if the region is not observed. We can use our prior knowledge about the structure; for example, if we see a table, we can estimate the structure behind the panel because we already know the general structure of tables.

To add this feature, a learning-based approach has been researched. Neural implicit representation is a neural network that inputs a position and outputs occupancy, signed distance field, color etc., allowing a continuous representation with any topology [5, 13, 19]. This representation could only reconstruct a small-scale structure such as the shape of each object from ShapeNet [4]. To further extend to a larger scene, a hierarchical grid structure has been used [3, 28]. However, these methods do not consider the camera tracking problem. To also perform camera tracking in a learning-based approach, neural implicit representation and differentiable rendering were used to optimize the camera pose through depth rendering cost etc [24, 9, 39].

Nice-SLAM [39] introduced a hierarchical scene representation to efficiently store geometric features on multi-level voxels that is used by a pre-trained neural implicit decoder. The Hierarchical Scene Representation combines multi-level grid features named "fine", "middle" and "coarse", each representing different scale structures. This representation enabled more detailed reconstruction while reducing the memory consumption required with all fine-grained voxels, achieving large scene application. However, we had to initialize the multi-level features; therefore, we had to allocate all the feature's memory in the beginning. This limits the scalability where only some part of the scene is valid since we waste a large portion of the memory.

In this project, we aim to increase the memory efficiency by introducing a voxel hashing method [17] to allocate memory only at the needed voxels.

Key contributions. Here are our key contributions in this report.

1. We integrated voxel hashing method into NICE-

SLAM framework.

2. Our framework can reduce the number of voxels initialised (∼60%-90%) leading to reduced memory requirements.

3. Our framework keeps similar performance of the original NICE-SLAM even with reduced memory requirements.

Outline. In Section 2, we discuss related work regarding SLAM and improvemnts in the field. We present an overview of our solution in Section 3. Furthermore, a detailed evaluation of our framework on different datasets is presented in Section 4. We discuss our evaluations from this paper in Section 5. Finally, we conclude this paper by mentioning our contributions in Section 6.

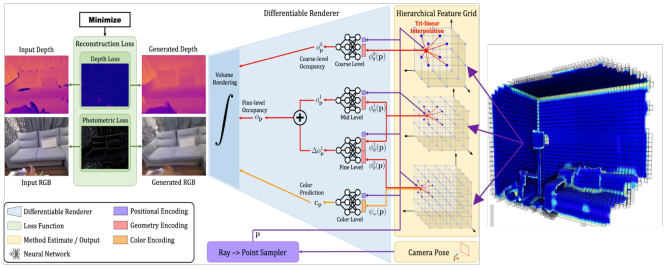

Figure 1: NICE-SLAM with Adaptive Feature Grids

2. Related Works

Our work in the project takes inspiration from two major concepts, *i.e*. Simultaneous Localization and Mapping and Dense Reconstruction. By combining methods from both fields, we build a SLAM framework with reduced memory requirements and increased scalability. Finally, we test our approach on several datasets to benchmark it against the current NICE-SLAM implementation. We summarize the related work as follows.

SLAM. The rapid increase in the applications of SLAM in autonomous driving and industrial use has led to numerous remarkable achievements in recent years. The recent development in the field is the introduction of **Dense Visual SLAM**. In it, the tasks are segregated into Tracking and Mapping. This is done to parallelize different tasks and increase the speed of the algorithms Mapping maps 3D geometry to specific keyframes, often represented as depth maps in the dense setting. Some of such examples with view-centric map represenation are DTAM [16], DeepTam [37], DeepV2D [29]. Some further extension in the field started using latent representations and their optimzation such as CodeSLAM [1], SceneCode [36] and NodeSLAM [26]. On the other hand, the world-centric map representation anchors the 3D geometry in uniform world coordinates, and can be further divided into surfels [22] and voxel grids, typically storing occupancies or TSDF values [32, 6]. Voxel grids have been used extensively in RGB-D SLAM, e.g., KinectFusion [14], SDF [2], BudleFusion [7] and Voxel Hashing [18]. In our proposed pipeline we also adopt the voxel-grid representation and voxel hashing.

Another major development in the field is the use of **Neural Implicit Representations**. They have proved promising in object geometry representation, scene completion, and generative modeling. They typically use an MLP to represent a 3D geometry that inputs position and outputs SDF [19, 23] or occupancy value [13, 20]. A few recent papers [27, 31, 35] attempt to predict scene-level geometry with RGB-(D) inputs, but they all assume given camera poses. Another set of works [12, 30] tackle the problem of camera pose optimization, but they need a rather long optimization process, which is not suitable for real-time applications. iMAP [25] is another important development in which, given an RGB-D sequence, they introduce a real-time dense SLAM system that uses a single multi-layer perceptron (MLP) to represent the entire scene compactly. Nevertheless, due to the limited model capacity of a single MLP, iMAP fails to produce detailed scene geometry and accurate camera tracking, especially for larger scenes. In contrast, NICE-SLAM [38] provide a scalable solution akin to iMAP, that combines learnable latent embeddings with a pretrained continuous implicit decoder.

Dense Reconstruction. Online 3D dense reconstruction is Determining 3D geometry of a static environment from sensor measurements. KinectFusion [10] was the earliest online 3D reconstruction. Earlier method primarily focused on using efficient data structures [21, 18, 34] for large-scale mapping without much emphasis on quality. BA were used to improve the quality of reconstruction and global consistency [33]. To improve the tracking and reconstruction, deep neural networks have been introduced. CodeSLAM [1] and NodeSLAM [26] performs Learned Probabilistic Reconstruction using Deep NN. The presence of stochastic sensor noise also motivates many works to consider the probabilistic distribution of the underlying geometry with either hand-crafted models or data-driven models. Implicit functions for geometric reconstruction started to be used to store surface information in sets of occupied voxels [6]. Its drawback due to discretization was also solved by mapping Gaussian Process and perform Bayesian map updates incrementally. They have been widely used in part segmentation, rendering, non-linear fitting, meta-learning, offline reconstruction etc. DI-Fusion [8] is among the first efforts to incorporate such an implicit representation with deep priors into an online 3D fusion framework.

3. Methodology

In this section, we present an overview of the methodology of our proposed framework for solving the limited scalability of the NICE-SLAM framework.

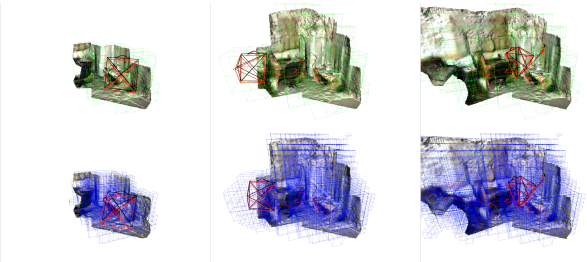

Figure 2: Visualization of valid voxels updated through voxel hashing. The top row shows middle grid in a green color and bottom row shows fine grid with blue.

3.1. Input Stream

The NICE-SLAM framework takes a continuous sequence of RGB-D images as the input. The ground truth RGB-D image is used to optimize the estimated camera pose through a differentiable rendering of the camera's view. The estimated camera position and the ground truth depth image are used to extend the map, the details for which can be found in Section 3.3.

3.2. NICE-SLAM Framework

We built up on the NICE-SLAM [39] framework. We modify the framework to work with a dense voxel array instead of a pre-initialized voxel grid. A brief explanation of the working of NICE-SLAM is given in this section.

An RGB-D stream is given as input to the NICE-SLAM framework. The geometry is encoded in three grids with different resolutions. This framework stores a 3-level geometric representation of the scene and its corresponding pretrained MLP decoders. The geometry decoders are pretrained as a part of ConvONet and are fixed. Using a differentiable rendering, NICE-SLAM generates a depth image and an RGB image. These renderings are compared with the input to generate a loss which can be used to optimize the latent representations. The differentiable rendering process integrates the predicted occupancy and colors from latent representation to produce images for the scene. For this, it samples N points along the back-projected ray of each pixel. It then calculates the occupancy by passing the latent representation through the decoder. These occupancy values are used to calculate the weight for each of the sampled points in the integration. Thus by sampling points along a viewing ray and querying the network, NICE-SLAM can render both depth and color values of this ray. Finally, by minimizing the re-rendering losses for depth and color images, NICE-SLAM is able to optimize both the camera pose and the scene geometry in an alternating fashion for selected keyframes. To optimize the scene representation, NICE-SLAM uniformly samples total M pixels from the current frame and the selected keyframes. Next, it perform optimization in a staged fashion to minimize the geometric loss $\mathcal{L}_g^l$ in eq. (1) and photometric loss $\mathcal{L}_p$ in eq. (2).

$$\mathcal{L}_g^l = \frac{1}{M} \sum_{m=1}^{M} \left| D_m - \hat{D}_m^l \right|, \quad l \in \{c, f\} \tag{1}$$

$$\mathcal{L}_p = \frac{1}{M} \sum_{m=1}^{M} \left| I_m - \hat{I}_m \right| \tag{2}$$

NICE-SLAM also runs in parallel camera tracking to optimize the camera poses of the current frame, i.e., rotation R and translation t. It samples M_t pixels in the current frame and apply the same photometric loss $\mathcal{L}_p$ in eq. (2) but use a modified geometric loss $\mathcal{L}_{g\_var}^l$ (3).

$$\mathcal{L}_{g\_var} = \frac{1}{M_t} \sum_{m=1}^{M_t} \frac{\left| D_m - \hat{D}_m^c \right|}{\sqrt{\hat{D}_{var}^c}} + \frac{\left| D_m - \hat{D}_m^f \right|}{\sqrt{\hat{D}_{var}^f}} \tag{3}$$

3.3. Dense Map

This section explains in-depth the integration of Adaptive voxel grids into NICE-SLAM. Our paper's major contribution is explained here. First, we describe the most important attributes of our data structure followed by its useful methods.

3.3.1 Attibutes

This section presents the efficient data structure used in our framework for implementing voxel hashing and efficiently storing and retrieving encodings and 3D point coordinates. It consists following elements -

1. **n_occupied:** It is an integer which denotes the number of occupied voxels in our framework.
2. **indexer:** It is a list of integer, which is initialised by -1 values. For each 3D voxel, we convert its location to an index using Equation 4. At this index, we store the location of the corresponding latent vector in 'latent_vecs' list.
3. **latent_vecs:** It is a list of shape (N, dim), where N is total number of voxels and dim is size of encoding. It stores the latent representations for the initialized voxels.
4. **latent_vecs_pos:** It is a list of shape $(N,)$, where N is total number of voxels. It stores indices corresponding to a points' location in indexer.

3.3.2 Point Cloud Generation

Our first target is to generate a point cloud from the depth images we get as an input. We generated it using the camera's intrinsic parameters and back projecting the pixels into 3D. Then, using the estimated camera pose, we transform the points from the camera coordinate frame to the world coordinate frame. We need this 3D point cloud to calculate the volumes in a scene with clusters of points.

3.3.3 Dynamic Voxel Allocation

After getting the point cloud, we can get the clusters of points in the scene. Each such cluster of points corresponds to an important geometric feature. Thus, we dynamically allocate voxels only in areas with point clusters and skip the allocation of voxels in areas with no or sparse 3D points. In our implementation, after getting 3D points, we pass these 3D points and allocate new voxels only in the volume around these points. As in a scene, most of the space is empty, a tiny fraction of the voxels are initialized per keyframe, and as we get new keyframes, we keep updating the list of voxels.

3.3.4 Voxel Feature Update

Once we get our dense voxel array, we need to update the information stored in it i.e., update the latent vectors. We need to train the encodings to output the correct scene representation to get optimized latent representations. . For tuning these encodings, we use the geometric loss calculated using depth, RGB ground truth and predicted images explained in Section 3.2. We use this loss for performing back-propagation and use these gradients for updating latent vectors.

3.3.5 Voxel Retrieval

Once we get the coordinates of each voxel, we need an efficient data structure for querying and storing the 3D points. We use voxel hashing technique for this. We first initialize an empty array. Then we take a 3D point, and pass the 3 coordinates to a fixed formula (4) to get a single output. Here, n_x, n_y and n_z denote the total number of voxels along with the x, y and z axis, respectively.

$$id(x, y, z) = z + n_z \times y + (n_z \times n_y) \times x \qquad (4)$$

We then use this output as the index for the 'indexer' array. It has indices of the latent vector corresponding to that point stored as the value. So in this way, we can easily get the point latent vectors from their 3D coordinates in $\mathcal{O}(1)$ efficiency.

3.3.6 Tri-Linear Interpolation

The voxels have a fixed size and can only provide a finite resolution while rendering. To this effect, we use trilinear interpolation to estimate the latent representation for any point in the 3D space. For each 3D point, we first find the surrounding voxel centers, which will be used for interpolation. In case the surrounding voxels have not been initialized, this means that the region does not contain any surfaces and hence the occupancy for that point can be set to zero. Once we have the surrounding voxels, we can trilinearly intepolate the latent repsresentation for the point based on its axial distances to the voxel centers.

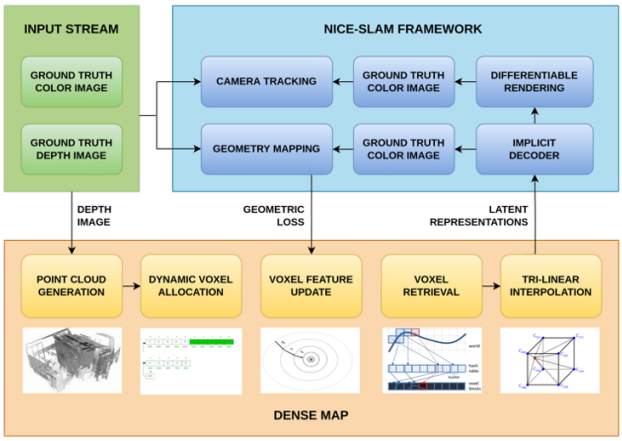

Figure 3: Implementation of NICE-SLAM Framework with Adaptive Feature Grids

4. Experimental Evaluation

This section analyzes and evaluates our framework extensively on different datasets. In section 4.1 we present how using adaptive feature grids via Voxel Hashing significantly reduces the memory requirements compared to the original NICE-SLAM. In section 4.2, we show the reduction in computational requirements. In section 4.3, we compare the quality of camera tracking, and in 4.4, we compare the surface reconstruction quality using our framework.

4.1. Memory Requirements

To compare the memory requirements, we tested our framework on two different datasets - ScanNet and Replica. As in our framework, we add new voxels only when a new frame is explored. So for both the experiments, we stored the number of voxels initialized whenever a new frame was added. In NICE-SLAM, the voxels in the entire space are initialized at the start, and doesn't change with the exploration of new frame. Thus the memory reduction can be

compared by comparing the reduction in the number of voxels initialized.

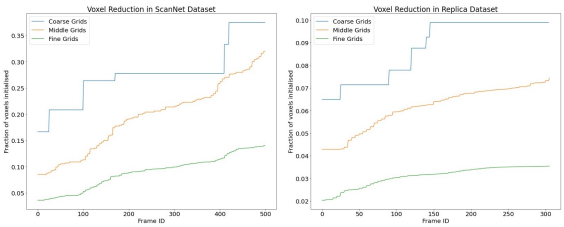

Figure 4: Fraction of voxels initialized compared to NICE-SLAM vs no. of frames added a) ScanNet and b) Replica datasets

In Figure 4, fraction of voxels initialized are plotted on Y-axis. For the ScanNet dataset, we can see that our framework achieved a reduction in the number of voxels initialized by 87.5% for fine grids, 70% for middle grids, and 62.5% for coarse grids. For the Replica dataset, our framework achieved a reduction in the number of voxels initialized by 97% for fine grids, 92.5% for middle grids and 90% for coarse grids. Seeing these results, the strength of our framework in reducing the memory requirements can be demonstrated for large-scale outdoor scenes.

We also present the comparison of the above-outlined methods in Table 1.

Framework	**Fine**	**Middle**	**Coarse**
NICE-SLAM	21525	2400	72
Ours	3399	711	29

Table 1: Comparison of voxels initialised in our framework against NICE-SLAM on a scene from ScanNet dataset

4.2. Computational Requirements

To compare the computational requirements, we report the number of iterations performed per second in our framework compared to NICE-SLAM. As seen in Table 2, our implementation requires more time to perform a similar number of iterations i.e. to similar number of frames. This can be attributed to additional time spent on voxel hashing and Dense map optimization. But as can be seen between the results of different datasets, we can infer that this computational time is independent of the size of the dataset used. Hence for larger datasets, the benefits of reduced memory requirements shown in Section 4.1 outweighs the increased time spent per frame and using our implementation, we can increase scalability to larger datasets with similar devices at the cost of increased time.

Framework	**Iterations per second**	
	ScanNet	Replica
NICE-SLAM	2.38	2.18
Ours	4.56	4.85

Table 2: Comparison of Iterations per second on ScanNet and Replica Datasets

4.3. Camera Tracking

As explained in Section 3.2, we sample points along the ray. Thus to sample points that are reasonable estimates and can help us in optimization, we need to have the camera at accurate positions. This can affect the quality of estimated RGB and depth images. To measure the quality of camera tracking, we find Absolute Trajectory Error RMSE values, which give an estimate of the difference between points of the true and the estimated trajectory. Thus smaller the value, the more accurate the tracking is. To benchmark, we report the ATE RMSE values of both NICE-SLAM and our framework. We benchmark tracking of our framework against NICE-SLAM, and the results are shown in Table 3.

It can be seen in the table that our framework can perform camera tracking, which is comparable to the NICE-SLAM. Thus for larger datasets, the benefits of reduced memory requirements again outweigh the slightly increased error values. This error can further be reduced by fine-tuning the hyper parameters, which can be an extension of our work. Using a good estimate of these hyper parameters produces promising results.

Framework	**ATE RMSE (cm)**
NICE-SLAM	4.1
Ours	5.23

Table 3: Comparison of ATE RMSE on ScanNet Dataset

4.4. Surface Quality

In this section, we compare the quality of 3D reconstruction, reconstructed RGB images, and depth images with the NICE-SLAM results. We present a qualitative comparison of the reconstruction quality. Figure 5 shows the quality of generated depth and RGB images. We can infer from the results that generated images are very similar to the original images. We show that our framework achieves a comparable image generation quality even with the reduced voxels.

Figure 6 shows the 3D reconstruction quality of ScanNet dataset using our framework. We can visually see that the side table, chair and study table have been reconstructed and can be easily identified and distinguished from surround-

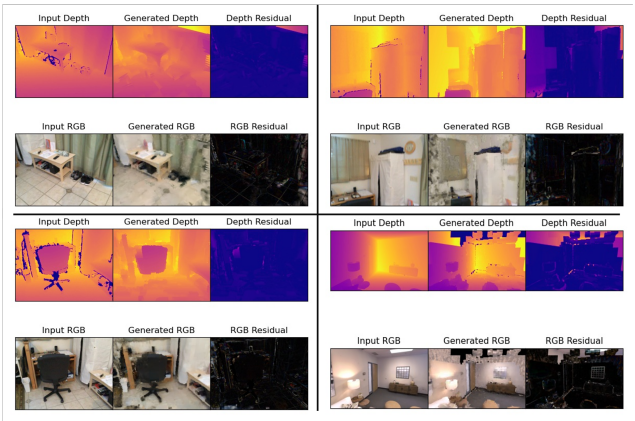

Figure 5: We show reconstruction results of our framework. For each scene, we show Input image (left), Generated image (middle) and Residuals (right). The datasets used are a) ScanNet, b) ScanNet, c) ScanNet and d) Replica

ing objects. Thus we can show that our framework can achieve comparable 3D reconstruction of the scenes with much lesser voxels also.

Figure 6: 3D reconstruction of scene from ScanNet dataset

5. Discussion

In this report, we demonstrated the effectiveness of integrating adaptive grids in NICE-SLAM architecture by benchmarking our approach against NICE-SLAM on a given Demo (ScanNet) dataset. We achieved lower memory requirements by dynamically allocating voxels using memory and speed-efficient data structure known as voxel hashing. By comparing on the Demo dataset, we show that our framework initialized 84.2% fewer voxels than NICE-SLAM for fine grids, and 67.9%, 59.7% reduction for middle and coarse grids respectively. As the number of initial grids corresponds to the dominating memory requirements, we relate this to a more memory efficient pipeline.

Our results above highlight that our framework can function even in devices with limited resources, which is a clear benefit of our approach over NICE-SLAM. This also proves that using similar resources, our approach can scale to larger outdoor environments.

Future work would be to further evaluate our pipeline on a larger dataset such as KITTI dataset. In addition, using a more efficient memory structure would be also an interesting direction. An octree can be used or the voxel size can be dynamically defined to further increase the memory efficiency.

6. Contributions

Algorithm Adaptation: Our framework builts upon the NICE-SLAM framework and our dense map data structure is adapted from DI-Fusion's implementation. The main backend behind our implementation is of NICE-SLAM. The function to be used for Voxel Hashing has been adopted from DI-Fusion from framework. We integrated the Voxel Hashing into NICE-SLAM framework by implementing or modifying following -

1. Creating the dense map data structure class *DenseIndexedMap* which contains the code for initialization, storage and retrieval of voxels. The *DenseIndexedMap* was adapted from Di-Fusion [8]
2. Replacing the voxel grid map with the dense map and creating necessary functions throughout the framework, i.e., for tracking, mapping, rendering , visualization and logging.
3. Modifying the ConvOnet decoder to work with the dense map for efficient forward and backward propagation.
4. Performing extensive evaluation on the benchmark datasets to understand the benefits and limitations of the implemented approach

Work Distribution: The distribution of the work between individual team members was as follows:

1. **Arjun Bhardwaj:** Implementation of voxel hashing data structure, Functions for integration of dense map, Code debugging, Code cleaning, Framework testing, Euler cluster setup, Documentation
2. **Tanmay Goyal:** Code implementation for voxel hashing and framework results' evaluation, Code implementation for meshing and scene reconstruction, Code debugging, Framework Testing, Benchmarking Framework on datasets, AWS server setup, Euler cluster setup, Documentation
3. **Takahiro Miki:** Code implementation of voxel hashing for visualization, visualization of allocated voxels, Euler cluster setup, Code Debugging, Documentation

References

[1] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J. Davison. Codeslam - learning a compact, optimisable representation for dense visual SLAM. *CoRR*, abs/1804.00874, 2018.

[2] Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. 06 2013.

[3] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *European Conference on Computer Vision*, pages 608–625. Springer, 2020.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015. cite arxiv:1512.03012.

[5] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[6] Brian Curless and Marc Levoy. 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, page 303–312, New York, NY, USA, 1996. Association for Computing Machinery.

[7] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Trans. Graph.*, 36(3), may 2017.

[8] Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. Di-fusion: Online implicit 3d reconstruction with deep priors. *CoRR*, abs/2012.05551, 2020.

[9] Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. Di-fusion: Online implicit 3d reconstruction with deep priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[10] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.

[11] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106. IEEE, 2013.

[12] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: bundle-adjusting neural radiance fields. *CoRR*, abs/2104.06405, 2021.

[13] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[14] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[15] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[16] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011.

[17] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[18] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6), nov 2013.

[19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[20] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, pages 523–540. Springer, 2020.

[21] Abdul N. Raouf, Osama Alluhaibi, Stewart Birrell, Matthew D. Higgins, and Simon Brewerton. A probabilistic octree fusion model for analytical-based observer fault detection in lsavs. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–7, 2020.

[22] Thomas Schoeps, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. pages 134–144, 06 2019.

[23] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[24] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6229–6238, 2021.

[25] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison. imap: Implicit mapping and positioning in real-time. *CoRR*, abs/2103.12352, 2021.

[26] Edgar Sucar, Kentaro Wada, and Andrew Davison. Neural object descriptors for multi-view shape reconstruction. *CoRR*, abs/2004.04485, 2020.

[27] Jiaming Sun, Yiming Xie, Linghao Chen, Xiaowei Zhou, and Hujun Bao. Neuralrecon: Real-time coherent 3d reconstruction from monocular video. *CoRR*, abs/2104.00681, 2021.

[28] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11358–11367, June 2021.

[29] Zachary Teed and Jia Deng. Deepv2d: Video to depth with differentiable structure from motion. *CoRR*, abs/1812.04605, 2018.

[30] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf-: Neural radiance fields without known camera parameters. *CoRR*, abs/2102.07064, 2021.

[31] Silvan Weder, Johannes L. Schonberger, Marc Pollefeys, and Martin R. Oswald. Neuralfusion: Online depth fusion in latent space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3162–3172, June 2021.

[32] Diana Werner, Ayoub Al-Hamadi, and Philipp Werner. Truncated signed distance function: Experiments on voxel size. volume 8815, pages 357–364, 10 2014.

[33] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34:598–626, 04 2014.

[34] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion. *Int. J. Rob. Res.*, 35(14):1697–1716, dec 2016.

[35] Zike Yan, Yuxin Tian, Xuesong Shi, Ping Guo, Peng Wang, and Hongbin Zha. Continual neural mapping: Learning an implicit scene representation from sequential observations. *CoRR*, abs/2108.05851, 2021.

[36] Shuaifeng Zhi, Michael Bloesch, Stefan Leutenegger, and Andrew J. Davison. Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations. *CoRR*, abs/1903.06482, 2019.

[37] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. *CoRR*, abs/1808.01900, 2018.

[38] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. NICE-SLAM: neural implicit scalable encoding for SLAM. *CoRR*, abs/2112.12130, 2021.

[39] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.