POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Degree Thesis

3D Indoor Environment Reconstruction for AR/VR Applications using a Smartphone Device

Supervisors

Prof. Andrea SANNA

Prof. Federico MANURI

Candidate

Alessandro Sergio MANNI

December 2021

Summary

Recent advancements in AR/VR technologies have created new ways of remote communication and collaboration. Computer-mediated shared virtual spaces are already used in many AR/VR applications: games, virtual meetings, navigation, interior design, training. Indoor environments, in particular, are well-suited for virtual representation: these are the places where people spend most of their time and where the majority of activities take place. Manually creating a computerized three-dimensional representation of these spaces is time-consuming and requires modeling skills that the average user doesn't have. Therefore, automatically reconstructing a digital version of real-world indoor environments could improve Extended Reality (XR) experiences. Following the success of Deep Learning, new approaches have been introduced: single view scene reconstruction relies on a single RGB photo of a scene to reconstruct the environment. The result depends on the complexity of the scene, as capturing all objects from a single RGB image would almost certainly result in heavily occluded objects with unpredictable shapes and pose. Moreover, this method does not solve the global scale depth ambiguity that arises when depth is inferred from a 2D image.

When evaluating the most efficient way to reconstruct a scene using a common smartphone, a full scan of the environment should be avoided, as this takes time and requires depth sensors that are only present on a few recent high-end devices. Furthermore, the virtual representation should be based on 3D CAD models as directly predicting meshes from images results in shapes that suffer from over-smoothing and tessellation issues. The requirements for a system performing such

automatic 3D reconstruction should include a straightforward process of data collection, as it should be performed by an average user, and the use of the most widely available device: a smartphone equipped with a single RGB camera.

This thesis presents a novel semi-automatic system that allows users to reconstruct the synthetic version of a real indoor scene using a smartphone. An Android ARCore-based app was developed in order to take photos of objects for which a 3D representation is to be included in the 3D scene reconstruction. The images, along with the depth information, are sent to a server for processing. This set of files is called snapshot. The output is the reconstructed scene, where for each physical object, the most similar 3D model is retrieved from a 3D model database, and its 7-DoF pose is estimated by the system: position, scale, and vertical rotation. When the user launches the Android app, the system starts to track the device's position and orientation; it predicts depth maps in real-time and detects horizontal planes in the scene. During the first stage of the reconstruction, when the user sees an object they want to include in the 3D environment reconstruction, they should target the object using the red pointer at the center of the screen and then tap the "Snap" button. When all the desired objects have been acquired, the user clicks the "Finish" button, instructing the app to send all the generated files to the server. All of the other stages are carried out automatically by the system. In the second stage, a Convolutional Neural Network (CNN) is used for classification and segmentation. Then the system: 1) generates a photo with only the object targeted by the user, 2) extracts the point cloud of the object from the raw point cloud of the scene. Then it proceeds by using a category-based approach: for small objects (e.g., mice, remote controls, bowls), computing a 3D oriented bounding box of such segmented point cloud is sufficient to estimate position and scale. On the other hand, large-size objects need further processing since they present some

difficulties: occlusion by smaller objects and outliers. In stage 3, a CNN is used to extract the features from the photo of the segmented object, these are compared with pre-computed features of 3D models to find the most similar 3D CAD model. In the last stage another CNN is used to estimate the vertical rotation of the object.

A Unity desktop application was developed to populate the virtual scene with instances of 3D models based on their estimated poses. This application can be used in online multiplayer mode together with the ARCore-based app and with another app developed for the VR headset Oculus Quest 2 to visualize the reconstructed scene.

A new dataset is introduced to test the system's accuracy. The dataset contains 500 snapshots taken from different viewing angles and distances of various objects organized into 13 semantic categories. The proposed solution achieves a maximum error of 18 percent for scale factor, less than 9 centimeters for the position, and less than 18 degrees for rotation.

This thesis presents a system capable of reconstructing an indoor environment using a smartphone with a single RGB camera. The results show that the proposed system can be used for XR applications, thus bridging the gap between real and virtual worlds. Future work could include a larger model dataset, which, when combined with non-trivial mesh deformation, may allow a more precise shape retrieval.

Acknowledgements

First and foremost, I would like to express my gratitude to Professor Andrea Sanna for his continuous support and guidance. His method of conducting research and his invaluable supervision has constantly pushed me to do better. Thank you to my second supervisor, Professor Federico Manuri, who was so supportive that he personally delivered to my home the device I needed when the university was closed. I am also indebted to Dr. Damiano Oriti and Dr. Francesco De Pace for their precious expertise that greatly helped the publication of the article that derives from this dissertation. I would like to thank Luigi for his friendship and the precious time we spent together working on projects for university courses. I would like to express my gratitude to my high-school and university colleagues Jacopo, Luca R. and Luca T., for all the years we spent together. My heartfelt gratitude goes to my family. Words cannot express how grateful I am to my mother and father, for all of your sacrifices made on my behalf, you made sure I had everything I needed even if that meant depriving you of something. To my brother for always believing in me and in my potential. To my aunt Carla who, despite the distance, always has a thought for me. To Giannina who is always present for me.

To Antonio, Cinzia, Maria Rita, Edoardo for their strength and for

having always had a word of encouragement for me. A special thanks to my friend Gianluca, who is always supported me and shown me how to face the difficulties of life. I would like to thank also my friends Francesco, Luca, Matteo, Marco, Mirko, Riccardo, Stefano for being present and helpful during these years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

In 1992, American author Neal Stephenson published Snow Crash, a science fiction novel in which he introduced the concept of the metaverse, where the prefix "meta" means "beyond" and "verse" means "universe". In Stephenson's concept of the metaverse, people use avatars to interact with each other in an immersive three-dimensional graphical environment, essentially living an alternative life. This virtual world is the result of the collaborative creation of its users, who can design and build complex social structures, including businesses, clubs, and cities. Although this concept of the metaverse appears dystopian and far from reality, recent advances in hardware, computer graphics, and artificial intelligence have opened up new possibilities for remote communication and collaboration with a sense of presence via customized avatars and immersive spaces. For example, Microsoft Mesh enables shared mixed reality experiences with the HoloLens headset, Facebook Horizon Workrooms is used for remote VR collaboration, Rec Room is a social VR application used by millions of users to play multiplayer games using VR headsets. The use of shared virtual spaces is rapidly increasing, and indoor environments, in particular, are well-suited for virtual representation: these are the places where people spend the majority of their time and where the majority of activities take place. Indoor scenes can be found in games as well as in Virtual Reality

(VR) and Augmented Reality (AR) applications such as interior design, navigation, and virtual meetings. Manually creating a computerized three-dimensional representation of these spaces requires modeling skills that the average user lacks, and the process is usually time-consuming. Therefore, automatically reconstructing a digital version of real-world indoor environments could improve Extended Reality (XR) experiences. The requirements for a system performing such automatic 3D reconstruction should include a straightforward process of data collection, as it should be performed by an average user, and the use of the most widely available device: a smartphone equipped with a single RGB camera.

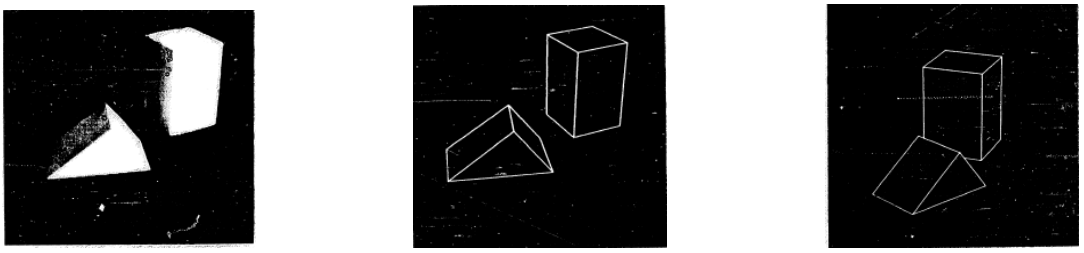
Figure 1.1: Microsoft Mesh

1.2 Problem definition

3D indoor environment reconstruction is a fundamental problem in computer graphics and computer vision. Traditional methods, such as photogrammetry, could be used to create the digital equivalent of a scene. There are several software applications that accomplish this task, but they require the user to take numerous photographs of the object

from various angles, basically forcing them to circle it. It is a slow process that is not suitable for large objects or for objects that cannot be circled. Additionally, this technique alone does not provide a method for determining the position of each object in space in order to recreate the entire scene. Scanning solutions that employ light detection and ranging (LIDAR) scanners or time-of-flight (ToF) camera tend to suffer from missing geometry, noise and mismatches. When evaluating how to quickly reconstruct a scene using a common smartphone, a full scan of the environment should be avoided, as this takes time and requires depth sensors that are only present on a few recent high-end devices.

One of the first researchers to address the problem of 3D perception from 2D images was Prof. Lawrence Roberts in his Ph.D. thesis in 1963. In his thesis, Prof. Lawrence presented a method that was able to construct the 3D representation of known objects from a photograph by recognising which object was in the scene and by inferring its position. The problem that affects Prof. Lawrence's work and today's solutions is that it is not possible to know a priori the exact objects that might be present in a given photograph of a scene. The variety of objects with different shapes in an indoor environment is extremely large, and occlusions and lighting are also a major problem. In recent years deep



(a) Original Picture. **(b)** Differentiated Picture. **(c)** Rotate view.

Figure 1.2: Inferring 3D objects from a photo. Prof. Lawrence Roberts's Ph.D. thesis, 1963.

learning (DL) methods have had great results in computer vision tasks such as image classification [1], object detection [2] and segmentation, [3] so researchers have been applied these techniques to 3D tasks [4, 5]. They can reconstruct a scene from a single RGB image by estimating room layout, object location, pose, and shape, they can add missing parts of a partial scan or retrieve and align 3D CAD models of objects to the scan to solve low resolution problem. DL add semantics. This means that it is possible for a machine to interpret and understand the physical world in a way that is closer to that of humans: for example, a reconstructed 3D environment with semantics allows a VR user to interact with the reconstructed objects as the user would in the real world. On the other hand, an AR user may wish to share the physical environment with a remote VR user in order to move and interact in the same space. In this case, scene understanding is necessary to provide both users with spatial and semantic information about the objects, resulting in a better experience.

When considering a fast scene reconstruction for AR/VR applications using a widely available smartphone, a full scan of the environment should be avoided because it takes time and requires depth sensors that are present on just few recent high-end devices. The position and scale of the objects are critical, so DL methods using a single RGB image for scene reconstruction are insufficient because they do not address the global scale ambiguity that occurs when inferring depth from a 2D image. It is, in fact, an ill-posed problem and an open challenge. Furthermore, reconstructing the whole scene from a single RGB image would almost certainly result in heavily occluded objects with unpredictable shapes and poses. Based on these considerations, this thesis presents a novel semi-automatic system that allows users to reconstruct the digital version of a real indoor scene using a smartphone.

1.3 Contribution

This paper presents a novel semi-automatic system that allows users to reconstruct the synthetic version of a real indoor scene using a smartphone. An Android ARCore-based has been developed, and it is used to take pictures of objects for which a 3D representation will be retrieved from a database. The images are then sent to a server for processing, along with depth information. The system is semi-automatic, as the user still must perform some manual tasks: 1) capturing a single image for each object and 2) specifying a point that belongs to the framed object. All of the other steps in the process are carried out automatically by the server. The first step is object classification and segmentation, which assigns a semantic label to each pixel and enables the system to differentiate between objects in the frame. After generating the photo with only the segmented object, the system finds the most similar 3D CAD model available in a model database. Lastly, the object pose and scale factor are estimated, and the object rotation and position in the smartphone's world reference system are computed. The object is scaled based on the computed scale factor, with no other mesh deformation. A desktop Unity3D application has been developed in order to populate the virtual scene with instances of 3D models based on their estimated poses. This application can be used in online multiplayer mode along with the ARCore-based app and with another version developed for the VR headset Oculus Quest 2 to visualize the reconstructed scene.

The diagram below depicts a high-level overview of the system Figure 1.3. Several techniques for overcoming the limitations imposed by single-view snapshots and low-resolution depth maps have been described. To assess the system's accuracy, a dataset of over 500 snapshots was introduced. To the best of this author's knowledge, this is the first system using a smartphone equipped with a single RGB

camera capable of estimating the 7-DoF pose (3-DoF position, 3-DoF scale and 1-DoF rotation around the longitudinal axis) of objects from single views, to reconstruct an indoor environment. The contributions

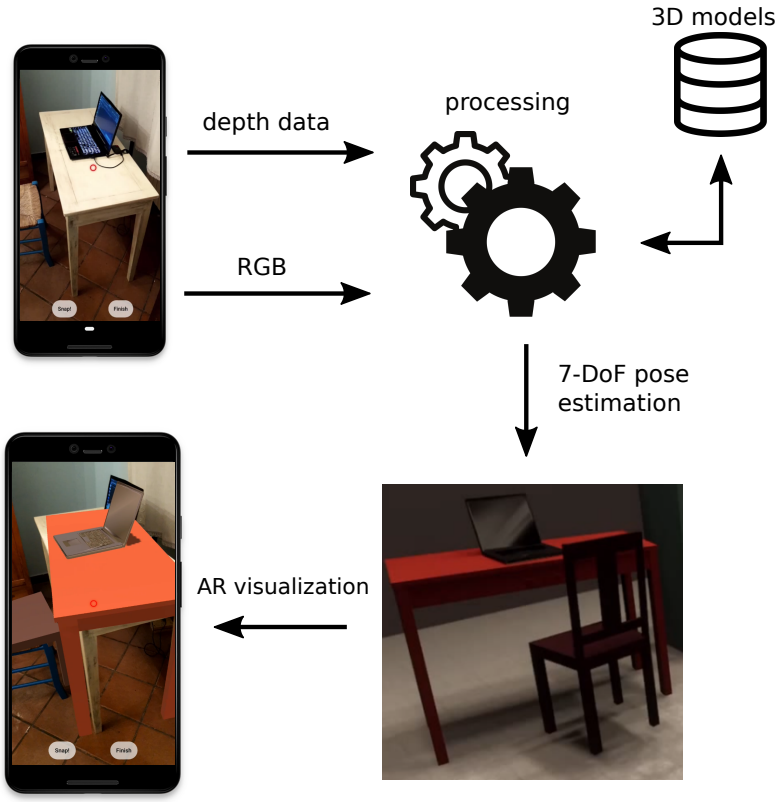


Figure 1.3: The proposed system. A single RGB image of each object is captured using an Android smartphone and augmented with depth information provided by Google ARCore. Then, a server uses these data to classify the object in the image and retrieve the most similar synthetic 3D object from a database; the model's position, vertical orientation, and scale factor are estimated and applied. Finally, an AR and a desktop/VR application can be used to visualize the reconstructed scene.

made with this work led to the following publication:

Alessandro Manni, Damiano Oriti, Andrea Sanna, Francesco De Pace, Federico Manuri, Snap2cad: 3D indoor environment reconstruction for AR/VR applications using a smartphone device, Computers Graphics, Volume 100, 2021, Pages 116-124, ISSN 0097-8493, https://doi.org/10.1016/j.cag.2021.07.014.

1.4 Thesis outline

The thesis is organized as follows:

- Chapter 1 is an introduction to the thesis, setting out the motivations that led to the development of the system presented. It also gives an overview of the problem addressed and the limitations of the currently available solutions. At the end of the chapter, the contributions made by this work are presented.
- Chapter 2 discusses the three main types of Extended Reality: Virtual Reality, Augmented Reality, and Mixed Reality, as well as the differences between them.
- Chapter 3 examines standard solutions for 3D indoor environment reconstruction, followed by a review of state-of-the-art Deep Learning-based solutions that includes 3D data and datasets. At the end of the chapter, a brief description of methods for image-based 3D shape retrieval is provided, as this is a critical component of the broader field of 3D reconstruction from RGB image.
- Chapter 4 elaborates the proposed solution starting from the system's specifications and overview and then moving on to a description of core modules. Following that, the main stages of the system pipeline are described.

- Chapter 5 details the procedures used to determine the system's accuracy. The main outcome of the experiments are showed.
- Chapter 6 presents a discussion of the results and the conclusions.

Chapter 2

Extended Reality Technologies

Extended reality (XR) encompasses all physical and virtual environments in which humans and machines interact through computer software and hardware. It includes Augmented Reality (AR), Virtual Reality (VR), Mixed Reality (MR) and all in between. In the Reality–virtuality continuum proposed in 1994 by Milgram and Kishino [6] all the type of environments starting from real and ending with virtual are depicted on a continuous scale in order to represent all the possible variations and compositions in the range. Figure 3.5 shows the Reality–virtuality continuum.

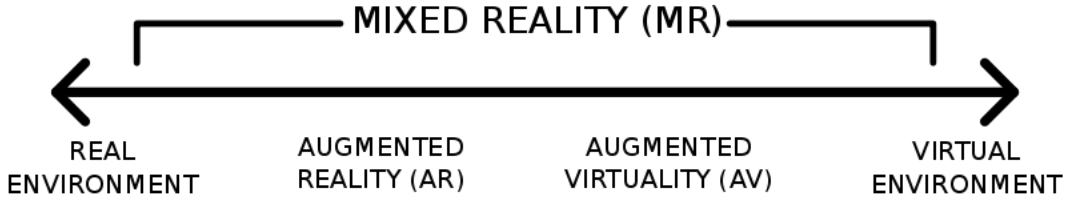


Figure 2.1: Reality–virtuality continuum.

2.1 Virtual Reality

Virtual reality is a computer-generated environment that can be classified into two types: non-immersive virtual reality, in which the user's

sense of presence is limited by the devices used to display the synthetic environment (laptop, desktop screens), and immersive virtual reality, wherein head mount displays (HMD) isolate the user from the real world and provide an experience that gives the user the sense of being in the virtual world. In most cases, the users control their virtual environment and are able to interact with digital objects. While virtual reality has existed for a long period of time, it has gained popularity recently as a result of the development of VR headsets such as the Oculus Rift, Quest, Valve Index, and HTC Vive. These headsets enhance the user's virtual reality experience by blocking out the outside world, presenting a more realistic virtual environment with a wide field of view. Virtual reality can be used for a variety of purposes, including training [7], gaming, and entertainment.

2.2 Augmented Reality

Augmented reality is a technology that allows people to interact with virtual content that is overlaid on the real world. The user see their real-world surroundings as well as computer-generated synthetic objects. AR enhance real-world experience by adding a new layer of perception through virtual elements. Azuma [8] defined some characteristics of an AR system: 1) it combines the real and the virtual, 2) it is interactive in real time, and 3) it is 3D registered. AR has grown in popularity in recent years as a result of the widespread availability of AR-capable smartphones and AR games. Additionally, retail companies use AR for smartphones to enable their clients to select products from a catalog and place them in their home to see how they fit.

2.3 Mixed Reality

Various definitions of Mixed Reality have been proposed by researchers over the years. It can be conceived as a more advanced form of AR (because the user is aware of their surroundings) or as a hybrid of AR and VR. Because of environmental understanding, interactions between virtual and real-world objects are possible in MR. Moreover, hands and head are tracked and instinctive interaction is accomplished through gestures recognition. Holographic devices such as Microsoft HoloLens and Magic Leap 1 have been used in manufacturing, remote support, inspection, and surgery [9].

Chapter 3

Literature Review

In this chapter, state-of-the-art solutions for 3D indoor environment reconstruction are described. Following a description of standard scanning methods, an overview of deep learning methods and the most commonly used datasets is provided. Deep learning methods take various approaches, and these methods can be classified based on the input data they require and the output they produce. Single view scene reconstruction refers to reconstruction solutions that take a single RGB photo of the scene as input and output either a scene reconstructed by predicting the mesh of each object detected in the scene or 3D CAD models retrieved from a database whose poses are predicted based on the physical counterpart of the photo. Multiple photos are used as input in multiple view scene reconstruction, and the two types of output are the same as in the previous method. An RGB-D scan of the scene is used as input in reconstruction from RGB-D methods, i.e., a camera paired with a depth sensor scans the environment. The output can be the same scan with the missing geometry predicted, or it can be a scene with 3D objects retrieved from a database and aligned to those scanned. A brief review of the literature about Image-Based 3D Shape Retrieval is given at the end of this chapter since it is an important sub-module of 3D indoor environment reconstruction and is part of the system proposed in this thesis.

3.1 Scanning solutions

Figure 3.1: A living room scanned using Polycam and an iPhone with a LiDAR sensor [10].

When an object is photographed, a digital camera maps the 3 dimensions of the real world into a 2-dimensional image. In this process, depth information is lost, so it is not possible to map 2D images to the 3D geometry of the object. A well-known method of creating a digital representation of a real object is photogrammetry. Based on triangulation it is able to reverse the process of photography by taking a series of photos of the object from different angles. To get a good result, the object should be positioned so that all parts are visible. The support of the object should be stable and as little visible as possible during the process. Otherwise, post-processing steps are required to remove everything that does not belong to the target object from the images. The number of photos needed can vary from a few for simple objects to hundreds for complex objects. There are a lot of software that can process those images and by mathematically intersecting converging

lines in 3D, they are able to determine the exact position of a point. The output is the point cloud of the object. A popular smartphone application was Autodesk 123D Catch, which required 26 photos from different angles to reconstruct an object. The higher the resolution and number of images, the longer it takes to reconstruct the object. With the rise of more affordable consumer-grade LiDAR and ToF sensors new RGB-D scanners solutions were introduced. ToF cameras measure the round trip time of an artificial light reflected in a scene creating a depth map. While LiDAR sensors use the time of flight to determine distance, instead of a flash of light, they use a pulse in a certain area of the scene and then move on to the rest of the environment in terms of nano-seconds. In the field of 3D indoor reconstruction also the advance of SLAM (Simultaneous Localization and Mapping) algorithms was fundamental: tracking and estimating the camera pose in an unknown environment is essential in an RGB-D 3D reconstruction pipeline. The typical pipeline for this scanning solution is composed by 4 stages [11]: first the input depth data generated by generated by the depth sensor is pre-processed in order to remove noise and outliers. Then the 6-DoF camera pose is estimated: the corresponding points between frames or between a frame and the model generated by previous iterations are found. The input depth map transformed using the estimated pose is fused into the 3D model of the reconstruction. A widely used smartphone app for iPhones equipped with LiDAR sensor is Polycam. With Polycam and similar scanning apps the user in order to obtain a textured scan of objects or environments needs just to frame and walk around with the device.

3.2 3D Deep Learning

3.2.1 3D Data

In contrast to two-dimensional data, which can be efficiently represented as a matrix, three-dimensional data are more difficult to manage for machine learning purposes. In computer science and geometry, an object can be represented in a variety of ways, each with advantages and disadvantages. A polygon mesh is the most common representation for graphics and solid modeling; a mesh is a collection of vertices, edges, and faces that define the shape of a polyhedral object. Edges connect vertices, and multiple edges can be connected to form faces; an edge connects exactly two vertices, whereas a face can connect three or more edges. Meshes are ideal for modeling and rendering because they provide artists with a great deal of control over the final result. Furthermore, they are very efficient in terms of storage and processing, making them ideal for interactive applications. In fact, the memory required for a mesh is proportional to the complexity of its shape rather than its spatial size. Voxel grids are regular structures in three-dimensional space that are similar to two-dimensional images [12]. The main advantage of voxels is their simplicity; the first attempts to use machine learning in 3D space involved the use of voxels because most nns conceived for image tasks such as classification can be easily translated to 3D when using voxels. Their main disadvantage is that they require a large amount of memory; this is a serious concern for machine learning because it is necessary to load several samples of data during the learning phase, which leads to memory saturation. Point clouds, like meshes, are collections of points with three coordinates that correspond to the dimensions of Euclidean space. Point clouds are sets [13] meaning that they are not ordered; any of the N! permutations of N points in a set represents the same object. The lack of topological order is the main

disadvantage of point clouds. Implicit representations [12], such as distance fields or occupancy functions, are a convenient way to represent objects in memory because they encode shapes using mathematical equations rather than just data. Because they are functions, they can provide infinite reconstruction resolution, whereas meshes, voxels, and point clouds suffer from limited resolution to varying degrees due to their discretization nature.

3.2.2 3D Reconstruction

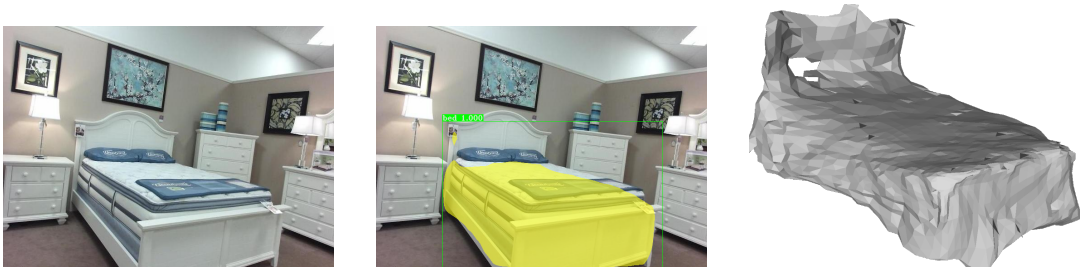
To reconstruct objects with complex structure and shape, the object representation must meet the following requirements: a) it must be as compact as possible, b) it must provide topological information, c) its memory requirement must scale well with shape complexity, and d) it must be well suited for machine learning. Regrettably, none of the currently used representations meet all of these requirements. Voxels do not scale well with complexity, point clouds lack topological relations, implicit representations are unsuitable for complex shapes, and meshes are difficult for NNs to handle. All reconstruction methods attempt to extract features from the object observations that best describe it. Then, those features, which are typically divided into global features describing the object's structure and local features describing the shape of its parts, become inputs for a reconstruction module, which outputs the 3D object in the desired representation. When using implicit representations, the reconstruction module may accept point coordinates as input; for example, occupancy networks output the probability of that point being within the volume of the object [14]. Another type of method does not create new objects; instead, it retrieves models from a database based on their similarity to the observed object. The criterion for determining whether two objects are similar is usually a function of their features, which define a

space in which all objects reside. The most common criterion is nearest neighbor, which selects an instance from the database based on its proximity to the queried object.

3.3 Single-View Scene Reconstruction

Obtaining a 3D reconstruction of a scene from a monocular RGB or single view image is an ill-posed problem and no single view scene reconstruction method exists that solves the global scale ambiguity that occurs when inferring depth from a 2D image. While the following works can create a synthetic scene that looks very similar to one depicted in a photograph, they are not suitable for AR/VR applications that require the actual position and scale of the objects.

3.3.1 Object-wise Mesh Prediction



(a) Input Photo from SUN RGB-D dataset. **(b)** 2D Recognition **(c)** 3D Reconstruction

Figure 3.2: An example of prediction using Mesh R-CNN.

As emerged by Tatarchenko et al. [15] many works that perform 3D reconstruction based on an encoder-decoder network architecture actually perform image classification. The reconstruction task instead is a complex problem and the solution needs to combine low-level

image cues, structural knowledge, and high-level object understanding. Some research has attempted to develop networks that directly output polygonal meshes. Mesh R-CNN [16] detects objects in a 2D image and returns a category label, a segmentation mask, and a mesh thus providing the full 3D shape of the objects in the scene. By using an approach consisting in voxel prediction and mesh deformation they are able to represent arbitrary topologies. In [16, 17, 18] the object is considered as a whole, while in [19, 20] the object is managed as a union of simple parts; each part is modeled separately, leading to better quality results. These approaches suffer from bad topology, and they do not really take advantage of the ability of meshes to efficiently scale with the shape complexity. Other works avoid the discretization of 3D space by using implicit representations such as occupancy functions [14, 21, 22] or signed distance functions [23]. However, these do not apply well to unknown objects (i.e., when objects are not present in the training dataset) and poses. Beside, they tend to output overly smooth shapes. Holistic approaches such as [24] try to consider the multi-later relation between objects in the whole scene (e.g., support, symmetry, etc.) to predict room layout (center, orientation and size), camera pose, object bounding boxes, and meshes from a single image.

3.3.2 3D CAD Model Retrieval

Izadinia et al. [26] reconstruct an indoor scene from a photo using 3D models in the following steps: estimating the geometry of the room, detecting the objects, retrieving the most similar synthetic objects from a large database of 3D furniture models, estimating the 3D pose of the objects, placing the objects in the scene, and finally using an optimization algorithm based on Convolutional Neural Networks (CNNs) trained on image comparison metrics to obtain the final configuration of the 3D models.

Figure 3.3: Mask2CAD prediction on two ScanNet images [25].

In Kuo et al. [27] work, given a single RGB image, each object in the image is detected, its 2D bounding box and segmentation mask are predicted, and from this the most similar 3D CAD model is retrieved from a database. The retrieval is possible because a neural network learns to map images of physical objects to 3D CAD models during training. At test time, the joint embedding space images-3D models is used to find the most similar synthetic object with respect to the detected object in the photo. The 5-DoF pose of the objects is also predicted to match the input image.

3.4 Multi-View Scene Reconstruction

Using a sequence of localized RGB frames, Multiple View Scene Reconstruction methods can reconstruct 3D indoor environments. Most of them assume that the camera pose has already been computed using SLAM [28] or Structure from Motion algorithms, and that it is known for each frame.

3.4.1 Object-wise Mesh Prediction

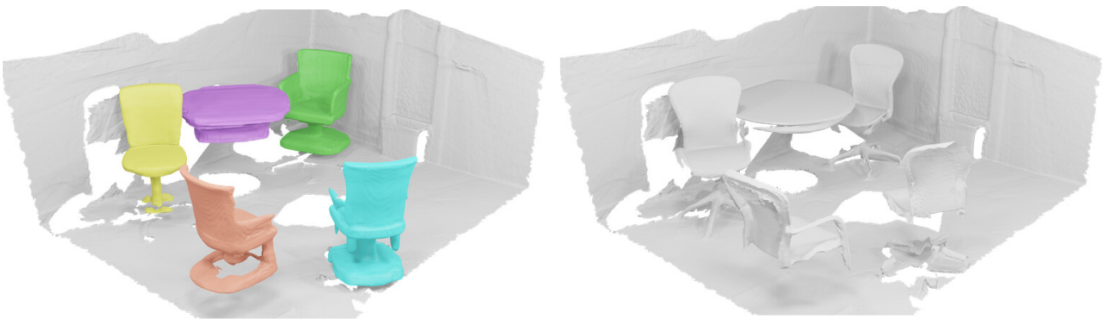


Figure 3.4: FroDO prediction on a sequence of images from ScanNet and ground truth scan [29].

FroDO [29] takes as input a sequence of RGB frames and camera poses and output 2D and 3D bounding boxes of object instances in the frames, their 7-DoF poses a sparse point cloud and a polygonal mesh. The core part of FroDO is novel deep joint shape embedding that makes possible for shape codes to be decoded to sparse point cloud and Signed Distance Functions (SDF), this allows faster shape optimization. The multi-view optimization is obtained though geometric, photometric and silhouette losses. MOLTR [30] uses a similar approach but [29] assumes that the environment is static while [30] can work also with dynamic objects. This work uses a multiple model Bayesian filter to track the motion state (both kinematics and motion status). The finale shape is reconstructed by decoding the shape code obtained by fusing the shape codes of each frame.

3.4.2 3D CAD Model Retrieval

Vid2CAD [31] is built on [27] and aligns 3D CAD models to a sequence of RGB frames containing different objects, it predicts the 9-DoF pose for each object present in the frames. While [27] suffers from the

scale-depth ambiguity, Vid2CAD integrates single-frame predictions by NNs across views in order to obtain globally-consistent reconstructed scene where each retrieved object has absolute scale and depth. This temporal integration improves 3D object rotation and position. Multi-view constraints also help dealing with occlusions. Vid2CAD assumes that camera pose with respect to the world is known for each frame.

3.5 Reconstruction from RGB-D scan

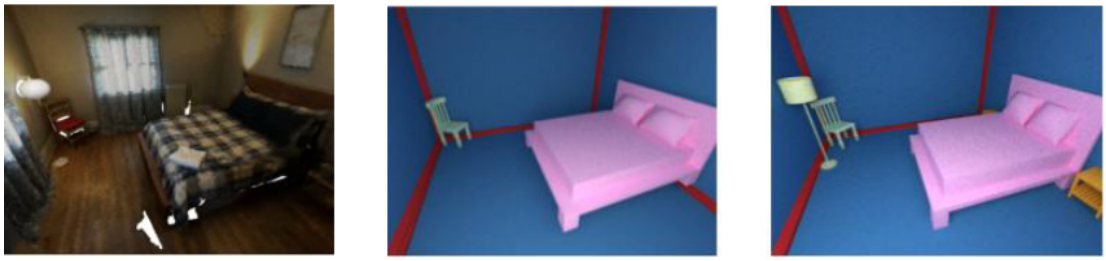


Figure 3.5: From left to right: ScanNet scan, SceneCAD [32] prediction, ground truth.

Reconstruction from RGB-D scan can be performed by predicting missing geometry or by aligning 3D CAD models. SG-NN [33] is a self-supervised method that takes as input partial RGB-D scans and predicts missing and unseen geometry. They introduced a 3D sparse generative neural network architecture that is able to output high-resolution scene geometry. RevealNet [34] is a 3D neural network architecture that jointly learn color and geometry features. It detects objects in the scene and predicts their complete geometry given an incomplete RGB-D scan. Scan2CAD [35] predicts correspondence heatmaps between regions of an RGB-D scan and 3D CAD models using a 3D CNN, then it finds the 9-DoF poses for 3D CAD model alignment to the scan. The main disadvantage of this approach is that it requires to compare each model

in the database to each scanned region to reconstruct the whole scene. SceneCAD [32] adopts an holistic approach by taking into account objects and layout component detected in an RGB-D scan and the relationship among them. A graph neural network is used to estimates object-object and object-layout relationships. The output is a scene reconstructed with 3D CAD models.

3.6 Datasets

Figure 3.6: Some 3D models from ShapeNetCore dataset. The models are in a canonical pose.

Numerous large-scale datasets of shapes have been made available to the academic community in order to facilitate the development of novel learning-based methods for 3D reconstruction and related tasks.

ShapeNet [36] is a collection of annotated 3D models organized under semantic categories in the WordNet taxonomy (a lexical database). ShapeNet contains more than 3 million 3D models, of which 220 thousand are divided into more than 3 thousand WordNet categories. ShapeNetCore (Figure 3.6) is a subset of ShapeNet that contains 12 thousand models distributed across a set of 270 categories. It is used in data-driven approaches that require 3D model data. ModelNet [37] is another database of 3D model, it contains 151,128 3D CAD models divided into 660 categories, it serves as benchmark for classification and retrieval tasks. ModelNet40 is a subset of ModelNet with only 40 categories and 100 object per category. NYU Depth Dataset [38] is used for scene understanding and consists of 1449 RGB-D images, acquired in 464 indoor scenes, annotated with support relationships. SUN RGB-D [39] containd annotated RGB-D images with: 3D bounding boxes, object orientations, room layout and scene category. ScanNet [40], which contains annotated RGB-D video of indoor scenes, is another commonly used dataset: 2.5 million views, over 1500 scene, camera poses, surface reconstructions, and semantic segmentations.

3.7 Image-Based 3D Shape Retrieval

Image-based 3D shape retrieval involves finding a relevant 3D shape from a 3D shapes database given a 2D natural image. Latest approaches use CNNs in order to overcome the domain gap between 3D synthetic shapes and real-world images. In Y. Li et al. [41] a CNN is trained to map images rendered from a database of 3D shapes to the base 3D shape. The 3D models are rendered using different viewpoints and lighting and then they are superimposed on real-world backgrounds.

Since similarities between images and 3D synthetic objects can be expressed as distances in embedding space, [42] uses Cross-Domain Triplet Neural Network (CDTNN) that incorporates an adaptation layer to match the features across synthetic and natural domains. In [43] it has been found that the texture of objects in natural images is one of the main factors that increases the gap between the 2 domains. They try to bridge this gap by synthesizing textures for the 3D shapes in order to make the neural network to focus more on geometry features.

Table 3.1: Summary of advantages and disadvantages of 3D reconstruction methods.

Method	Advantages	Disadvantages
Photogrammetry	• Cheap • High-resolution texture	• For indoor environments it could require hundreds of photos • Noisy result
3D Scanning	• Faster than photogrammetry • Chance to correct artifacts during the scanning	• Requires dedicated depth sensor • Oversmoothed shapes
Single-View Reconstruction	• Fast • It needs just a single RGB camera	• Global scale depth ambiguity
Multi-View Reconstruction	• Solves scale-depth ambiguity • Faster than a full scan • It needs just a single RGB camera	• Requires multiple RGB images for each object in the scene
Reconstruction from RGB-D scan	• Fixes most of the artifacts of conventional 3D Scanning	• Requires dedicated depth sensor • Full scan of the environment

Chapter 4

The Proposed Solution

4.1 Specifications

The proposed system recognizes the framed objects and retrieves the most similar 3D CAD models (represented as polygonal meshes) from a database. Let $\mathbb{M} = \{m_n\}$ be the model set, with $1 \leq n \leq N$ and N be the number of 3D CAD models. Each model m_n belongs to a specific category or class $c_k \in \mathbb{C} = \{c_k\}$. For each object, the goal is to predict its class c_k, find the most similar 3D CAD model m_n and estimate its 7-DoF pose: 3-DoF for translation, 3-DoF for scale, and 1-DoF for rotation around the longitudinal axis 4.1.

The system operates in indoor scenes using a handheld Android device running a Google ARCore-based app [44]. ARCore is a software development kit that can be used to create augmented reality applications. A *snapshot* containing: 1) the RGB image, 2) the depth data, 3) the device pose, and 4) the lowest horizontal plane is acquired for each physical object.

The system does not require a depth sensor since it leverages the Visual-SLAM and the depth map estimation algorithms included in the ARCore Depth API. Specifically, a Samsung Galaxy S8 fitted with a single RGB camera was used in this work.

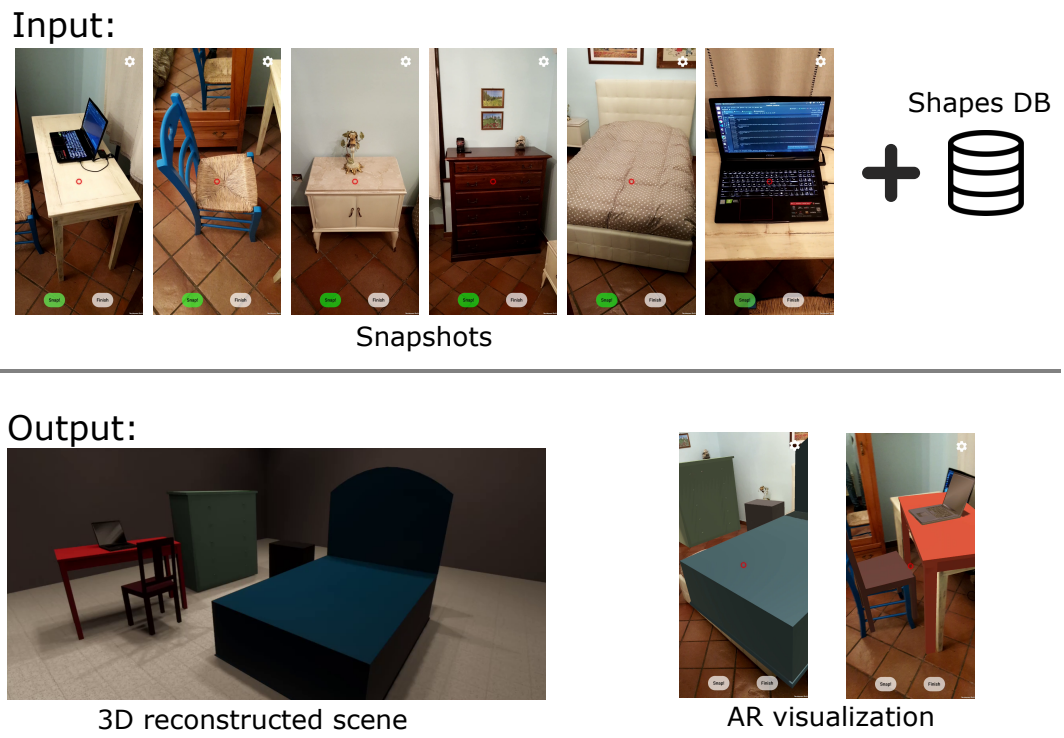


Figure 4.1: The proposed system takes as input a database of 3D models and a single RGB image for each object that the user wishes to reconstruct. The output is the reconstructed scene where each 3D model has a 7-DoF pose estimated by the system: position, scale and vertical rotation. The scene can also be visualized in AR using the smartphone, so that the user can verify that each object matches its physical counterpart

4.2 System Overview

It is possible to divide the reconstruction process into 4 stages. In the first stage, snapshot acquisition is performed using an Android smartphone supported by ARCore and equipped with a single RGB camera. In this work, a Samsung Galaxy S8 is used. After the first stage, all data is sent to a server running Ubuntu. All further stages

are executed on this server by an application developed in Python. Then in the second stage called object classification, position and scale estimation, the HR-Net-W48 CNN is used for classification and segmentation, then Open3D is used to process the 3D data. In stage 3, a VGG-19 CNN is used for model retrieval and finally in stage 4, the pose estimation of the object is performed by PoseFromShape with another CNN. In the tests conducted, reconstructing a scene with 5 models takes 21 seconds. Figure 4.2 illustrates the four stages of the proposed system with the main components.

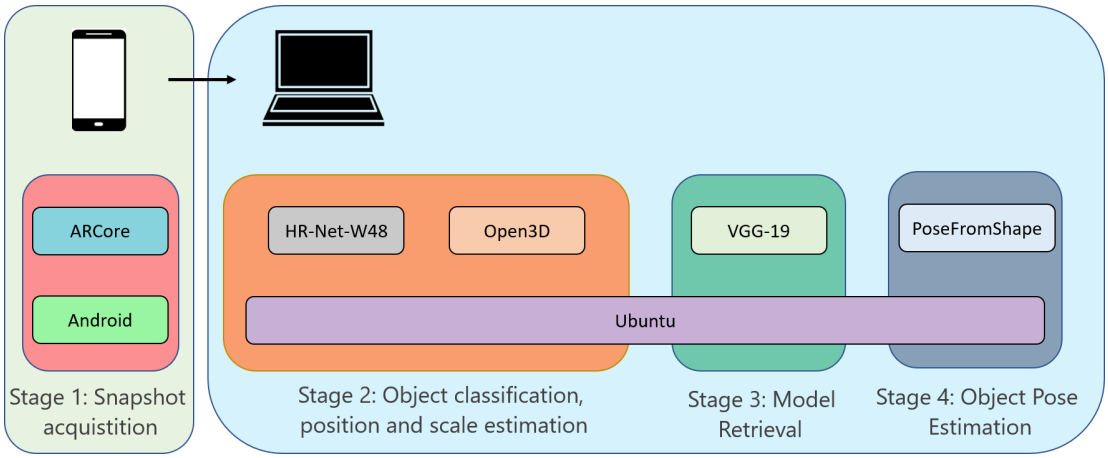


Figure 4.2: Overview of the proposed system.

4.2.1 Google ARCore

Google's ARCore is a software development kit for creating AR experiences for Android smartphones. ARCore uses motion tracking techniques to determine the location of the smartphone in the world. Motion tracking makes use of the phone's camera to identify meaningful points, which are referred to as feature points. As the phone moves, these points are tracked in real time. ARCore determines position and orientation by combining the phone's Inertial Measurement Unit (IMU)

with the tracked feature points. ARCore is also capable of detecting and tracking flat surfaces such as horizontal and vertical planes via feature points. These characteristics enable the placement of virtual objects in a physical environment, as well as their movement and interaction. With the Depth API [44], a new level of immersion and realism is achieved. By estimating depth maps in real time with a single moving camera, the Depth API allows a more immersive augmented reality experience: it is possible to take into account occlusions and enable collisions between virtual and physical objects. Using a new pipeline for depth from motion, [44] generates dense depth map: in the first step it finds a suitable keyframe from previous image frames, the relative 6-DoF pose between the keyframe and the latest frame is then used for polar rectification. A conditional random field (CRF) is used to compute correspondences that generates disparity maps. Triangulation can estimate a sparse depth map from disparities and from the sparse depth map, a novel fast bilateral solver produces a bilateral depth grid. This last one can be transformed into a dense map on demand.

4.2.2 HR-Net-W48 and Mseg

The presented solution uses for segmentation the neural network HR-Net-W48 [45], which was pre-trained with Mseg-3m-1080p model shared by [46]. Mseg is a composite dataset that brings together segmentation datasets from various domains. Due to the fact that different datasets use different names for the same semantic category and use different annotation systems, it is not possible to merge them. In [46] they relabeled over 220 thousand object masks in over 80 thousand images. They combined COCO [47], ADE20K [48], SUN RGB-D [39], Mapillary [49], IDD [50], BDD [51] and Cityscapes [52] into a single composite dataset with 194 categories. This dataset enables the development of a single semantic segmentation model that performs well in previously

unknown environments. High-Resolution Net (HRNet)-W48 [45] is a convolutional neural network designed to maintain high-resolution representations in position sensitive tasks, such as semantic segmentation. An example of the output with each object highlighted is shown in Figure 4.3.

Figure 4.3: An indoor scene segmented using HRNet-W48 pretrained with the Mseg-3m-1080p model.

4.2.3 Open3D

Open3D [53] is an open-source library available both in C++ an Python used to handle 3D data. In this thesis, it was combined with another Python library, Numpy (which deals with matrices), to process point clouds. The main functions used are: statistical outlier removal, which

removes points that are further away from their neighbors in relation to the average for the point cloud, 3D oriented bounding box, and I/O functions for 3D geometry. Section 4.4 contains more information on these functions.

4.2.4 VGG-19

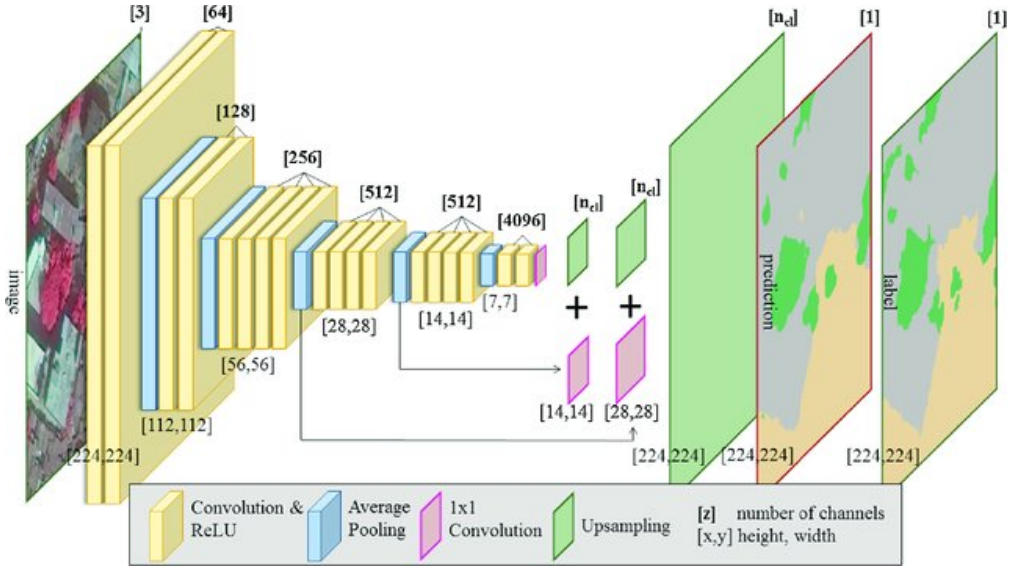


Figure 4.4: Architecture of the neural network VGG-19. Credit: Eitan Kosman

VGG-19 [54] created by the Visual Geometry Group of Oxford is a convolutional neural network with 19 layers: 16 convolution layers, 3 fully connected layer, 5 maxPool layers, 1 softmax layer. In this work it used for 3D model retrieval: it extracts features from images to establish similarity. In order to extract features from images, the output of the first fully connected layer is used, which yields a 4096-dimensional vector that is later used to compute the norm of the difference between the query image and the pre-compute features of the renderings of the 3D model DB. The system uses VGG-19 pre-trained

with the ImageNet [**image**] model. ImageNet [55] is a database of over 14 million images categorized according to the WordNet hierarchy. Figure 4.4 shows the network architecture, notice the 4096 dimensions vector at the output of the first fully connected layer.

4.2.5 PoseFromShape

PoseFromShape [56] can estimate the pose of a 3D object from a 2D image. This method enables the prediction of the viewpoint of a 3D from an RGB image even if the object was not seen at training time. In the first step they extract deep features from the input images: one is the image with the object in the wild the other is the set of rendered views of that object or a similar one. A ResNet-18 is used for this task. These features are then concatenate in a global feature vector. A multi-layer perceptron is used as estimator and since they use a classification-and-regression approach the network output is made by probabilities and offsets (2 outputs for each Euler angles of the camera: azimuth, elevation, in-plane rotation) those are combined into a single loss function. In this thesis PoseFromShape was pre-trained with a ShapeNetCore model and each 3D object was rendered in 216 views.

4.3 Snapshots Acquisition

The first stage of the proposed solution involves taking snapshots of the objects with an Android smartphone. Using Unity[1] and also the ARCore Depth API SDK, an Android app was created based on the utilities introduced in Google's DepthLab app [57]. The created app tracks the smartphone position in world coordinates, predicts depth maps in real-time, and detects horizontal planes. When the app is

[1] `https://unity.com/`

Figure 4.5: The smartphone app developed for data acquisition.

launched, ARCore creates a coordinate frame $W = \{O, xyz\}$ by tying its origin and orientation to the device's starting position $P_{device}(0) \in \mathbb{R}^3$ and orientation $R_{device}(0) \in \mathbb{R}^3$:

$$O = P_{device}(time = 0) \tag{4.1}$$

$$xyz = R_{device}(time = 0) \tag{4.2}$$

By moving the smartphone, the device pose is computed by the ARCore Visual-SLAM algorithm. When the user wants to retrieve a digital version of an object, they should target the object on the smartphone screen by using the fixed virtual red cursor shown in Figure 4.5. The user can then take a snapshot by tapping the "Snap!" button, which

activates four events.

1. The app stores the i-th single-view RGB image F_i^{rgb} of the scene in camera resolution Res_{cam}^{rgb}. This is done by using an API that provides CPU access to GPU texture and then through another function that convert the texture that is in YUV-420-888 format to RGB24.

2. The app checks the list of detected horizontal planes $\mathbb{P}_i^h = \{p_{i,j}\}, \forall j \in \{1,2,\ldots,J\}$, where J is the number of detected planes in the acquired snapshot. The lowest plane $h_{lp,i}$ is considered as a potential ground plane, and thus its height in relation to the smartphone's initial position is saved in a CSV file. *SnapInfo*:

$$h_{lp,i} = \min_{j \in \{1,2,\ldots,J\}} height(p_{i,j}) \tag{4.3}$$

   The rotation of the smartphone around its vertical axis $R_{device,i}^v$ is also saved and later used to predict the rotation of the object.

3. The app saves the mapping between screen space coordinates of the captured image F_i^{rgb} and the coordinates of the estimated depth map F_i^{depth} of that image in a CSV file called *ScreenToDepth*. This is needed because the depth map computed by ARCore has a resolution of 160x90px, which is significantly less than the color frame resolution Res_{cam}^{rgb}, equal to 2220x1080px.

4. In order to generate the point cloud in camera space Q_i^{cam} corresponding to a determined depth map, the app uses the camera intrinsic parameters, i.e. principal point (c_u, c_v) and focal length f. If $Q(u,v)$ denotes the vertex of the point cloud corresponding to the pixel (u,v) of the depth map, and $z(u,v)$ is the depth value, then its x, y and z coordinates are computed using the following

equation:

$$Q(u,v) = \begin{bmatrix} \frac{u-c_u}{f} z(u,v) \\ \frac{v-c_v}{f} z(u,v) \\ z(u,v) \end{bmatrix} \tag{4.4}$$

5. The app stores in another CSV file called *PointCloud* the mapping between the screen space coordinates of the estimated depth map F_i^{depth} and the world space 3D points. Using the [57] utilities, the world space 3D points were calculated starting from the camera space 3D points. The constructed point cloud has approximately 14000 points.

It is more efficient to have separate files. The depth map is 160x90 pixels so in the *PointCloud* there are 14000 lines for the mapping between x, y of the depth map and the related world space coordinate of the points. The *ScreenToDepth* file is obtained through an API function that can only map the RGB image coordinates to the depth map coordinates and not vice-versa. This file would require more than 2 million lines since the RGB image resolution is 2220x1080 pixels but it is less than 30000 lines since a downsample is applied. The matching between the point cloud and the RGB image is done on the server, starting from the *PointCloud* file. On the other hand, if this process were performed on the smartphone, the app would freeze for a noticeable amount of time. Google ARCore's APIs and the smartphone processing power pointed the author towards this multiple file solution. These steps are carried out for each object that the user wishes include in the 3D reconstructed environment. This stage concludes when the user clicks the "Finish" button, instructing the smartphone to send all generated files to the server. The server script was written in Python, whereas the client is embedded in the ARCore-based app and is written in C#. The connection is established as soon as both applications start; when the client sends the image and snapshot files, it uses a coroutine so that

all ARCore operations can continue during transmission; the server receives and places them in enumerated folders. When the finish button is pressed, a message is sent to the server indicating that the acquisition phase has ended and that no more objects will be framed.

4.4 Object Classification and Estimation of Object Position and Scale

The Python program running on the server automatically processes the files generated by the smartphone application. The goal at this stage is to classify the selected object for each snapshot as well as estimate its position $P_i \in \mathbb{R}^3$ and size $S_i \in \mathbb{R}^3_{>0}$ in the world space. To extract only the object targeted by the user from an RGB image, the software uses semantic segmentation, which assigns a semantic label (i.e., an object class label) to every pixel in the image. The segmentation network produces a gray scale image F_i^{gray} with pixels labeled with a gray color code corresponding to a particular semantic class. Due to the fact that the segmentation network yields an image with a lower resolution than the source image, the image is resized to the resolution of the frame F_i^{rgb} acquired by the smartphone using nearest-neighbor interpolation to prevent unwanted artifacts such as gray tones that were not present previously. The pixel in the image's center contains the color code for the selected object. If the color code denotes a class of system-managed objects, the system continues processing the image; otherwise, it is discarded. Since the red pointer used to target the object was located in the center of the image, a flood fill applied at these coordinates, creates a binary mask that isolates the framed object. The system can now retrieve the point cloud of the object Q_i' highlighted in the mask from the raw point cloud Q_i of the image using the pixel-by-pixel mapping between the mask, the corresponding estimated depth map, and the

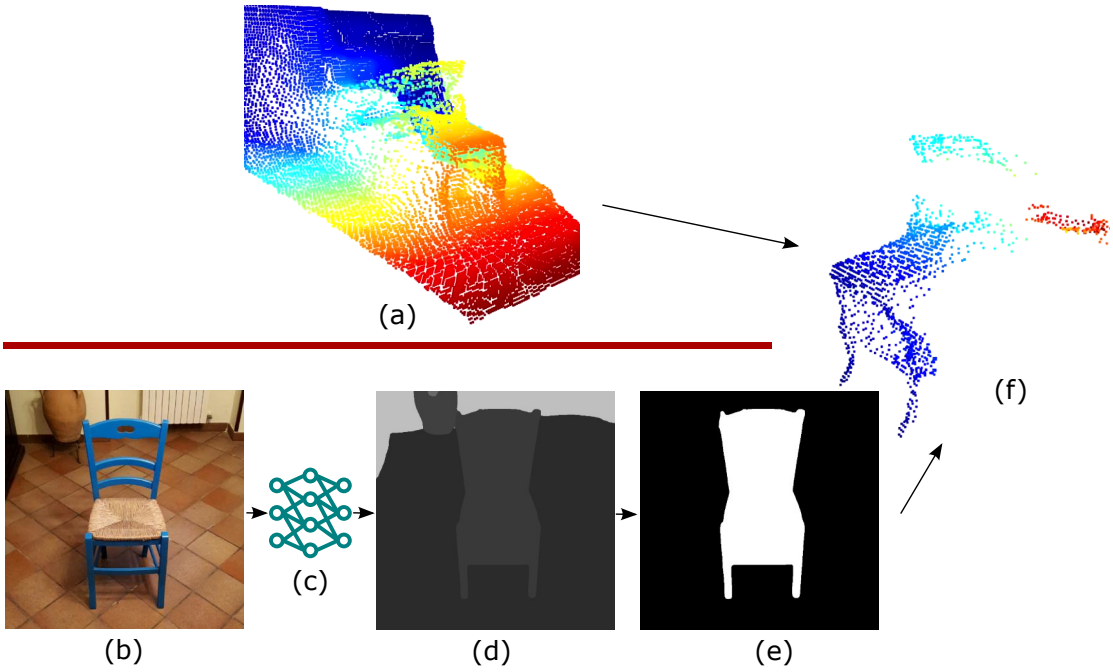


Figure 4.6: Object classification and segmentation. (a) Raw Point Cloud. (b) Photo. (c) HR-Net-W48 Segmentation Neural Network. (d) Segmentation Mask. (e) Binary Mask. (f) Segmented Point Cloud.

world space point cloud. A transformation matrix $T_{cam \to world}$ is used to transform the point cloud from camera space to world space, as seen in the following equation:

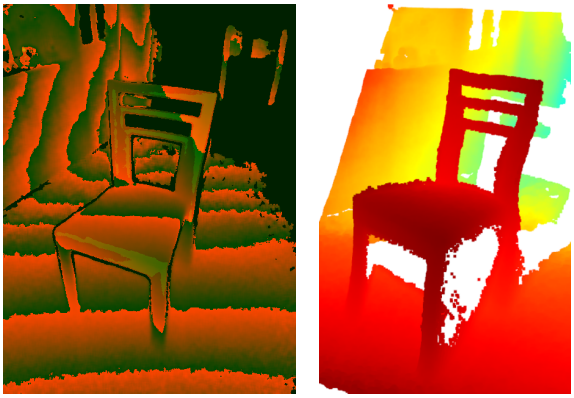
$$Q_i^{world} = T_{cam \to world} \cdot Q_i^{cam} \tag{4.5}$$

Then, the set of points belonging to the object is extracted from the original point cloud by using a binary mask that can be obtained by performing a flood fill operation on the segmentation image starting from its central pixel. The operation is depicted in Figure 4.6.

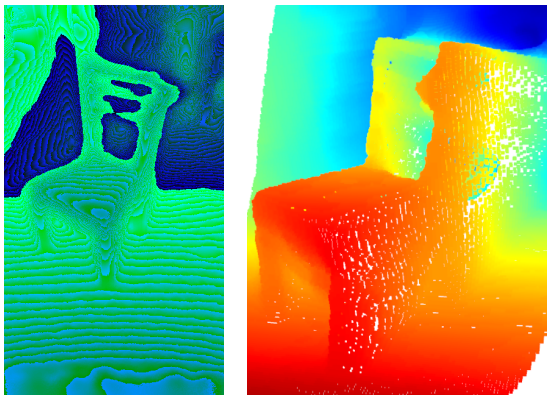
Different object types present different challenges, so once the system detects the category of the framed object it proceeds in using the right category-based approach to estimate position and scale. For

small objects (e.g., mice, remote controls, bowls), computing a 3D oriented bounding box of such segmented point cloud Q_i^{world} is sufficient to determine the centroid and 3D size of the object. The system employs an Open3D-implemented [58] algorithm that approximates the minimum volume box containing a set of points. The objective is to compute the smallest-volume box that contains a finite set of 3D points. The convex hull of the points provides support for the minimum-volume box. Because the hull is a convex polyhedron, any points inside it have no effect on the bounding box. Thus, the algorithm's first step is to compute the points' convex hull. At least two adjacent faces of the minimum-volume oriented box must be flush with the convex polyhedron's edges. The box faces must be perpendicular. The function processes all edge pairs in order to determine the appropriate box-face normals that are candidates for the pair's minimum-volume box. Due to the iterative nature of the computations, the output is not the exact minimum-volume box, but a close approximation based on the minimizer's sample size [59]. When objects are small, and especially when the point cloud is incomplete, information such as an RGB image is required to detect them. Many neural networks that detect objects solely from point clouds are unable to recognize them because the vertices do not form a discernible shape.

Large-size objects, on the other hand, present a number of difficulties: 1) they may be made up of different parts, 2) they may have outliers throughout the point cloud due to the object's single view, 3) and they may have flying points due to the smoothing that ARCore utilizes on the depth maps. Figure 4.7 depicts the ARCore smoothing effect, which cannot be disabled. For such objects, the segmented point cloud is subjected to a RANdom SAmple Consensus (RANSAC) algorithm that detects only horizontal planes. Given a set of 3D points, the RANSAC algorithm determines the equation of an infinite plane and returns inlier points from the point cloud. It iteratively selects



(a) ToF sensor depth **(b)** ToF point cloud



(c) ARCore depth **(d)** ARCore point cloud

Figure 4.7: The depth maps and point clouds. (a-b) The depth map acquired using a generic ToF sensor and the corresponding point cloud. (c-d) The same ToF depth map processed with ARCore and the resulting point cloud, respectively. The ARCore smoothing process creates flying points on the object contour.

three random points from the point cloud, finds the plane equation defined by those three points, calculates the distance between the points and the discovered plane, and selects those with a distance less than the threshold. To obtain only the horizontal plane, all plane equations

with more than 10° of inclination with respect to the ground plane are discarded. Then, a 3D oriented bounding box is computed using the points on the detected plane. This operation returns the object's extent and the position of its centroid, with the value of its vertical axis representing the object's height. When objects are made up of holes, the ARCore smoothing process yields flying points, which can affect the size of the 3D oriented bounding box. Chairs present particular challenges due to the possibility of cavities in the backrest and the fact that the 3D-oriented bounding box does not provide information on the seat's height. To address these issues, the system takes three steps. The first step is to create a mask of depth variances in the depth map in order to remove the majority of the point cloud outliers: a gray scale version of the depth map is normalized to enhance the depth variations, and an automatic Canny edge detection algorithm similar to the one presented in [60] is applied. Then, two morphological transformations are used: dilation and closing. As illustrated in Figure 4.8, the kernel of the Canny edge ranges with the distance between the object center and the smartphone. Outliers in the point cloud that are concentrated in depth discontinuities are removed using the generated binary mask. In the second step, the RANSAC algorithm detects the horizontal plane of the seat, and in the third step, a statistical outlier removal algorithm is applied to the point cloud segmented by the plane, removing any remaining outliers from the previous step. Figure 4.9 describes the entire procedure. At this point, the system computes the 3D oriented bounding box of the segmented plane identified by RANSAC, which yields the extent of the seat as well as the centroid falling on the seat, which gives the seat's height. Additionally, the chair's overall height is determined by locating the highest point in the segmented point cloud data. Cabinets, dressers, and nightstands, on the other hand, are handled differently by the system due to three factors: 1) the segmentation network does not always predict the right label; 2) objects lying on the

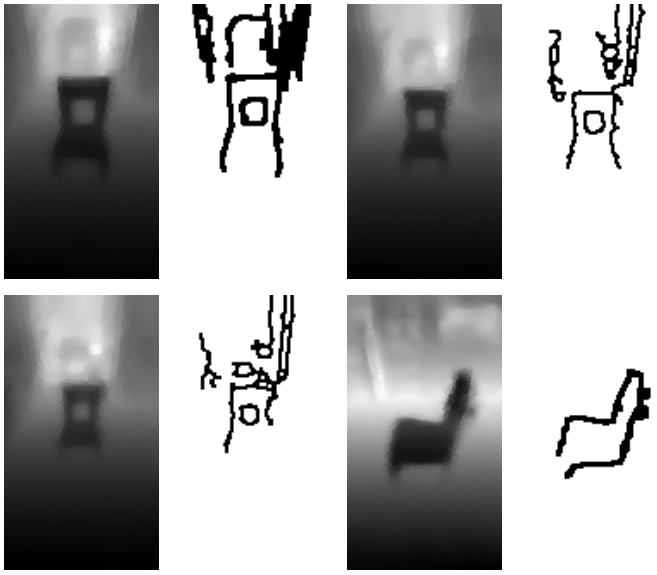


Figure 4.8: Depth maps (normalized in gray scale) and relative edge masks generated using a Canny edge detection operator.

small surface of a nightstand may cause artifacts in the point cloud data; and 3) the RANSAC algorithm may fail to detect the horizontal plane (the camera may not be able to frame the upper surface of a cabinet due to its vertical position). For these types of objects, the system locates the highest vertex of the segmented point cloud, that is presumed to be the height of the upper surface, and then moves the vertices on the vertical axis to create an artificial horizontal plane. The Open3D algorithm for estimating the 3D oriented bounding box is used to obtain the object's extent and the 2D position of the centroid, which is unaffected by the previous operations. An example of this operation is shown in Figure 4.10. Figure 4.9 describes the entire procedure.

4.5 3D Model Retrieval

In Stage 3, the system performs image-based 3D model retrieval. This task consist in retrieving a relevant 3D synthetic object in a database

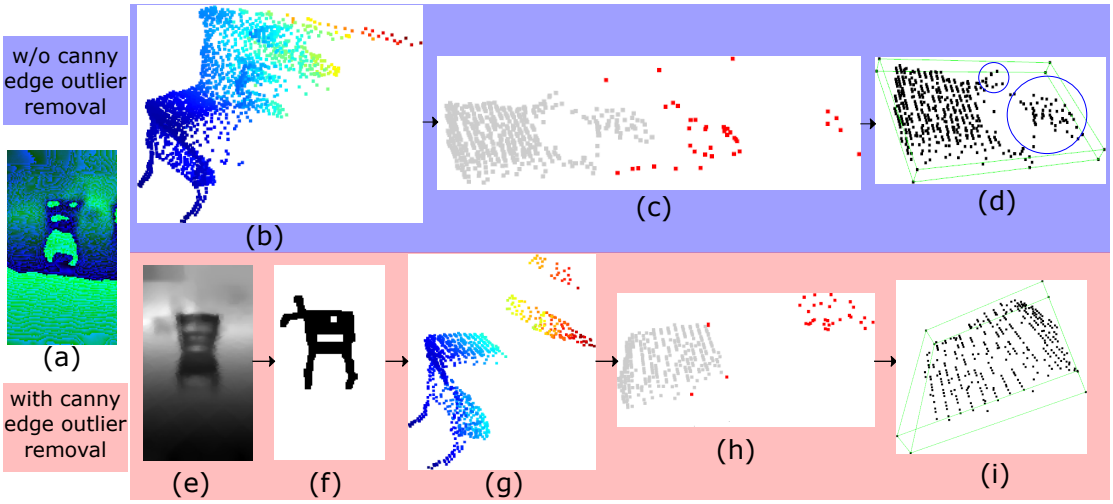


Figure 4.9: Flying points removal using Canny edge detection algorithm and object position and scale estimation. The (a) input depth map estimated by ARCore is transformed to a point cloud (b). Outliers highlighted in red (c) are detected and removed by the Statistical Outlier Removal (SOR) algorithm; in (d) it is shown how the bounding box is not correct because some outliers were ignored by the SOR algorithm. By using the Canny edge detection algorithm on the (e) normalized depth map, whose result is shown in (f), the outliers generated by the ARCore smoothing process are removed in the corresponding point cloud (g). (h) shows the remaining outliers eliminated with the SOR and the bounding box is correctly estimated (i).

which is similar to a query photo taken in the real world. A VGG-19 [54] CNN was used in this work because it provides better fine-grained 3D model retrieval than other approaches. A subset of 3D models from the ShapeNetCore dataset [36] is rendered from 12 viewpoints, and their features are extracted using the VGG-19 CNN pre-trained on ImageNet. These features are computed and saved only once. The most similar object is obtained by calculating the Euclidean distance between the segmented target object's extracted features and the features associated with 3D models in the same category. The goal is to extract the object's

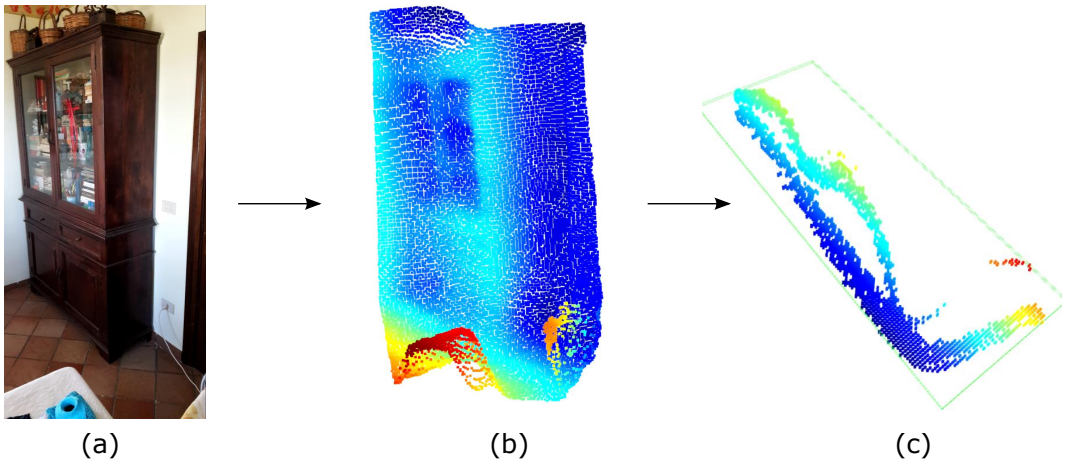


Figure 4.10: Example of cabinet position and scale estimation. (a) Photo of the cabinet; (b) Point cloud (no horizontal planes can be detected); (c) 3D oriented bounding box of the highest vertices

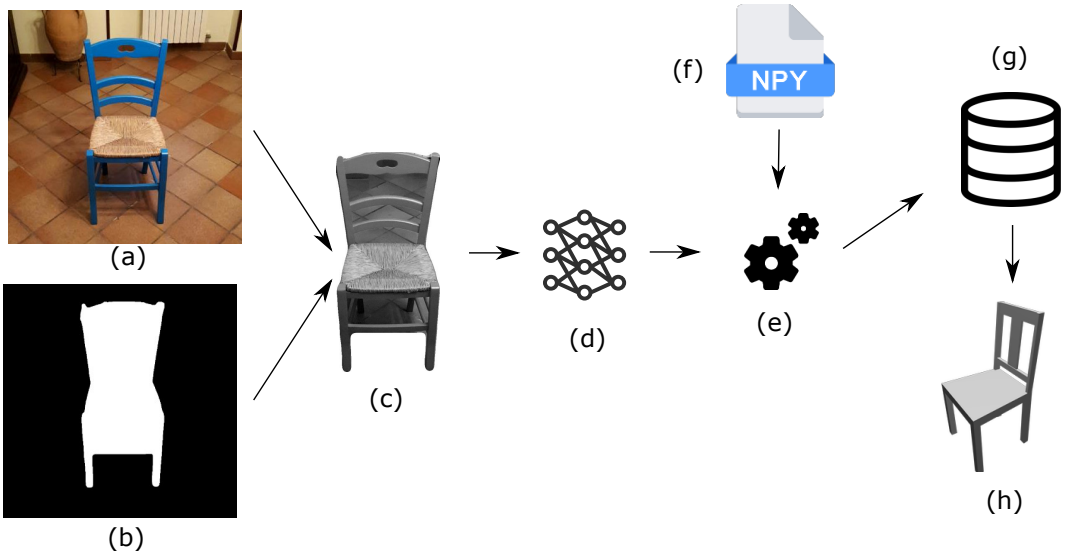


Figure 4.11: 3D Model Retrieval. (a) Photo. (b) Binary Mask. (c) Segmented Photo. (d) VGG-19 Feature Extraction Network. (e) Similarity Finder between (d) and (f) pre-computed features, the output is the id of a 3D model in the (g) 3D Model DB. (h) Model Retrieved.

features rather than those of the surroundings, and in order to reduce the noise in natural images, the system extracts the query object from the snapshot using the binary mask created in stage 2, followed by a blur operation to soften the edges and a final cropping operation to center the object. The VGG-19 CNN extracts the features of the segmented objects. The Euclidean distance between the input images and all the renderings belonging to the objects in the segmentation network's specified category is then calculated. The 3D object associated with the rendering that is closest to the natural RGB image is chosen. Figure 4.11 shows the process while an example of 3D model retrieval is shown in Figure 4.12.

Figure 4.12: The query segmented images and the top-3 models retrieved. The first column contains segmented photos of real objects that were used as queries, while the second to fourth columns contain the most similar retrieved models (best from left to right).

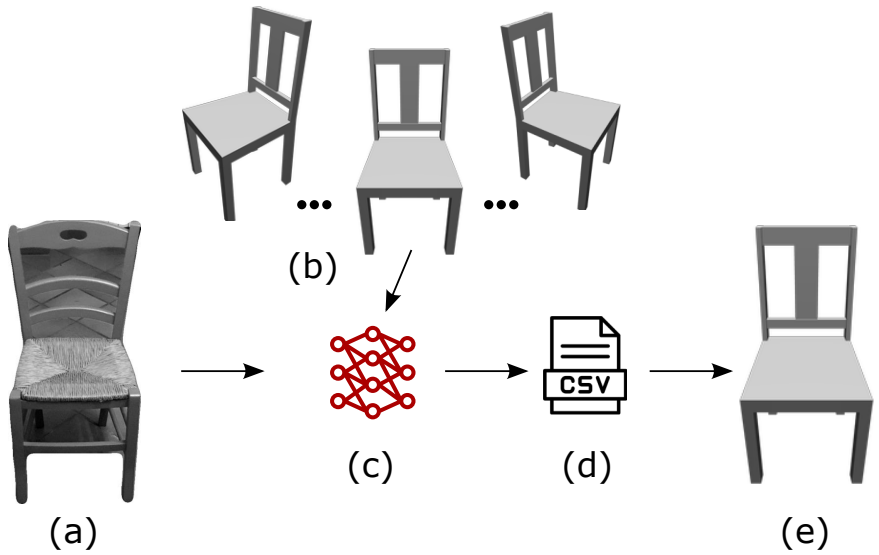


Figure 4.13: Object Pose Estimation. (a) Segmented Photo. (b) 3D model renderings of the retrieved model. (c) PoseFromShape Pose Estimation Network. (d) Estimated Rotation. (e) 3D model rendered in the predicted vertical rotation.

4.6 Object Pose Estimation

In the fourth stage, the system estimates the rotation of the object around the vertical axis. Several deep learning solutions attempt to solve the problem of estimating object pose from a single RGB image, but many require the neural network to be trained on the specific object whose pose is to be estimated. This approach is unsuitable with the system's intended operating conditions: an indoor environment where previously unseen objects must be considered. As a result, the proposed system employs PoseFromShape [56]. With respect to known viewpoints, this DL solution can estimate the viewpoint of an object in an RGB image. PoseFromShape employs the model provided by [56], which was trained on ShapeNetCore, a subset of ShapeNet (Figure 3.6

illustrates some models of the ShapeNet dataset). The 3D models of the recognized category were taken from this dataset and rendered from various viewpoints using Blender.

Since the background of the RGB snapshot could affect prediction accuracy, the system utilizes the image of the segmented object obtained in the previous stage. As the system has already retrieved the 3D object from the previous step, it passes the neural network the set of renderings associated with that object as well as the object's segmented photo. The PoseFromShape network's output is the camera's viewpoint, which is made up of the Euler angles for azimuth, elevation, and in-plane rotation. The system only takes into account the azimuth that will be applied to the 3D model used to represent the physical object. Figure 4.13 illustrates the process.

4.7 Scene Reconstruction and AR Visualization

The system creates three CSV files for each snapshot: 1) values of the object's position and size, 2) vertical rotation of the object, and 3) longitudinal rotation of the smartphone. Moreover, the height of the floor h_{floor} that is valid for the entire reconstructed scene is saved in a separate CSV file called *Floor*. The floor's height is calculated as the average of all values that differ by no more than 4 cm from the lowest value on the list $\{h_{lp,i}\}$:

$$h_{lp} = \min_i h_{lp,i} \tag{4.6}$$

$$h_{floor} = \text{avg}(h_{lp,i}) \quad \text{if} \quad \text{abs}(h_{lp,i} - h_{lp} < 4cm) \tag{4.7}$$

It has been noted that when a plane's tracking is lost and then rediscovered, there is a discrepancy of less than 4 cm from the previous detection. To visualize the reconstructed scene, a desktop was created

Figure 4.14: The AR view. Two chairs visualized by the AR app: the occlusions are taken into account.

using Unity. The 3D models used for reconstruction has been taken from the ShapeNetCore. The 3D models were pre-processed to present real-world dimensions and centroid positions that are consistent with those estimated by the system. The Unity desktop program parses all CSV files, selecting 3D models and calculating rotation, scale, and position values. Then it instances them in a scene where also the floor has been instantiated at the pre-calculated height. All 3D models are also visualized by the mobile AR app to qualitatively assess the accuracy of the reconstruction. This is possible because both applications include

an online multiplayer engine, Photon Engine[2]. When the desktop application instantiates the 3D objects in the scene, it also sends a remote procedure call (RPC) to the Photon server that forwards it to the smartphone. The RPC contains the ids of the instantiated 3D models, allowing them to be instantiated in AR mode using the smartphone. These virtual 3D models overlap with the physical object because the position, scale, and rotation are also sent via RPC. The desktop user (represented by a red capsule) and the AR user (represented by a 3D smartphone model) are tracked and transmitted via internet in real time, making it possible for both users to see each other's movement in the environment. The AR user can freely move around the real-world scene to ensure that the 3D models overlap with the physical objects. ARCore's geometry-aware occlusion feature aids in the evaluation. As such, since a dense depth map is predicted almost each frame, it is possible to determine whether a 3D model is correctly overlaid on a physical object (Figure 4.14). Figure 4.15 shows example of a shared reconstructed space between AR user and desktop user, Figure 4.16 between AR user, desktop user and VR user. Additionally, another application has been developed for the Oculus Quest 2 HMD in order to extend the sharing of the environment to a VR user. The Oculus Quest 2 is equipped with 4 integrated cameras and sensors that enable inside-out 6 DoF tracking, it also has 2 controllers tracked in 6 DoF. A low poly avatar has been created so the desktop user and the AR user can visualize how the VR user is moving in the scene, the movements of the real-hands are tracked through the Quest controllers, their position and rotation are also transmitted via internet using Photon engine. Figure 4.16 shows an example of this multi-user XR experience. At the following link a video showing the proposed system[3].

[2] `https://www.photonengine.com/`

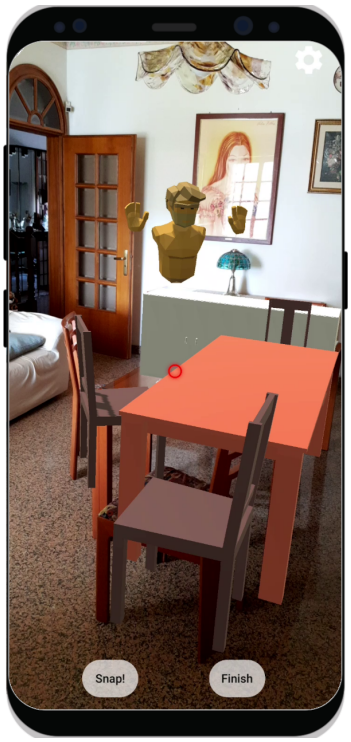
[3] `https://www.youtube.com/watch?v=uRkGR54dyt4`

(a) Camera view **(b)** AR view

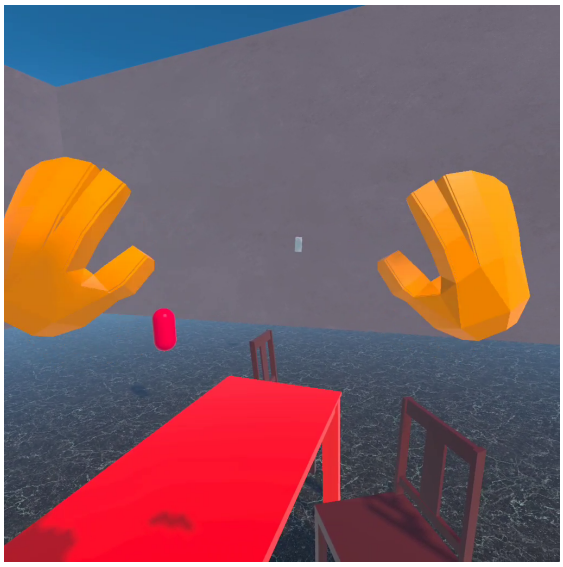
(c) Desktop view

Figure 4.15: Multi-user (AR/desktop) visualization of a reconstructed scene. Desktop player is depicted as a red capsule in (b).

(a) Camera view

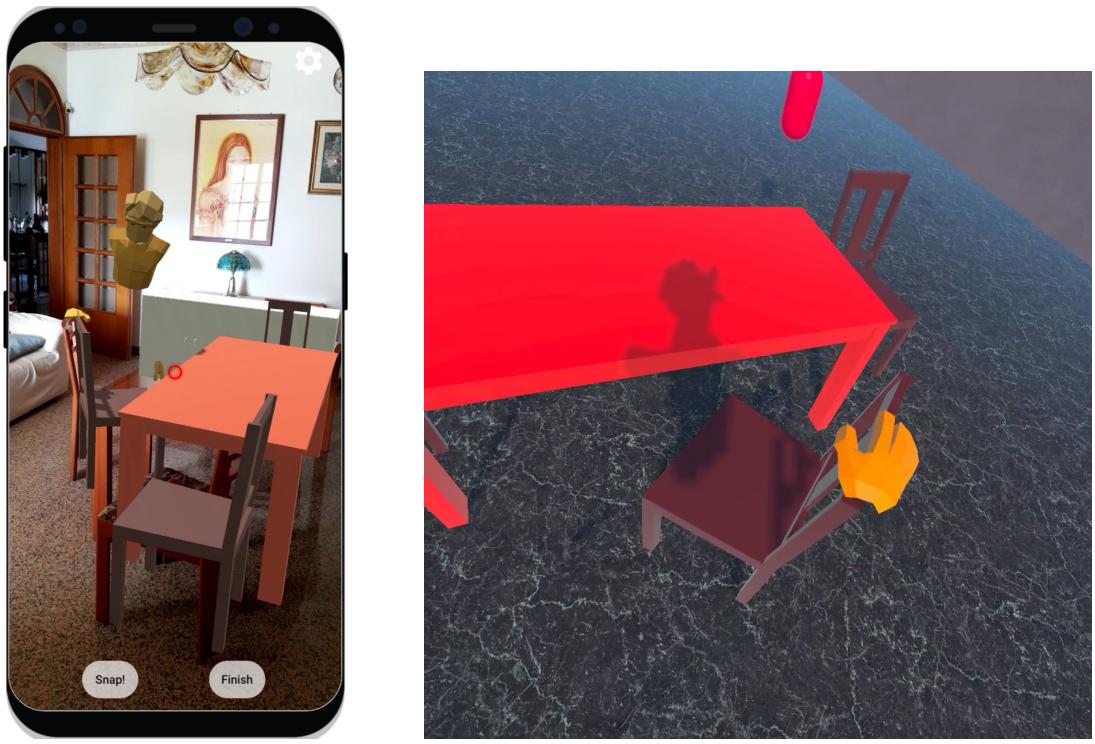


(b) AR view



(c) VR view

Figure 4.16: Multi-user (AR/VR/desktop) visualization of a reconstructed scene. VR user is the yellow avatar in (b), desktop player is the red capsule in (c), AR player is the gray smartphone in (c).



(a) AR view **(b)** VR view

Figure 4.17: VR user is touching the reconstructed chair in VR while the AR user see the VR avatar touching the real one in AR.

Chapter 5

Experimental Evaluation

In this chapter the evaluation process is described including the steps to get the ground truth (GT) values. To assess the system's accuracy, a new dataset is presented. The dataset is composed of 500 snapshots of various objects of various shapes acquired with a Samsung Galaxy S8 in indoor environments. The objects are divided into categories, and the snapshots were taken from various angles and distances. A table showing the main outcome of the experiments is presented.

5.1 Markers

While it is possible to obtain the GT values (width, length, height) for the size of an object by using a measuring tape it is not possible to use any tool to get GT values for position or rotation with accuracy. As to an object's GT position, it should be noted that this is based on a reference system created by ARCore when the app is launched and the smartphone begins to move, so it changes every time, limiting the use of a fixed coordinate reference system. The GT rotation of an object could be obtained by progressively moving the object, for example, on a platform and measuring the rotation with a protractor; however, because large and heavy objects, such as cabinets, beds, and tables, must be evaluated, this is not feasible. The presented solution

employs markers for position and pose estimation. In the case of GT position the markers is positioned at the center of the object and the portion of the point cloud that is delimited by the marker is used to obtain the coordinates of the center in the ARcore reference system. Instead for the GT rotation the marker is used to find the camera pose. The pattern of marker chosen is the ChArUco diamond because it can tolerate partial occlusions when framed, unlike the classic square black and white chessboard.

5.1.1 Camera Calibration

In order to perform detection and camera pose estimation with the diamond marker a camera calibration is needed. Camera calibration consists in finding the camera matrix (made by intrinsic and extrinsic parameters) and the distortion coefficients. A chessboard marker has been printed and photographated from multiple view using the smartphone's camera. Those images are used as input for a Python script using OpenCV, a Computer Vision library, that through the *findChessboardCorners* function detects the chessboard and with *calibrateCamera* function return the camera matrix and the distortion coefficients. *CalibrateCamera* calculates the starting parameters and use the Levenberg-Marquardt optimization algorithm to minimize the reprojection error.

5.2 Scaling Error

The physical object's size is used as the ground truth (GT). To calculate the scaling error for each category, multiple snapshots from various perspectives (distances and angles) were taken. Following that, for each snapshot, the root mean squared error (RMSE) between the GT and the estimated size is calculated. Finally, the category scaling error

is computed by taking the average of all RMSE values. Because the number of snapshots is large, a Python script has been written to read the GT value for each object from a CSV file and then use the RMSE to calculate the average for the category to which the object belongs. For each dimension (x, y, and z) the error is calculated and expressed as a percentage of the absolute difference from the ground truth value.

5.3 Position Error

The GT position of an object is determined by a marker placed in the physical object's center. In the case of a chair, it would be on the seat during the first snapshot and then removed. After that, multiple snapshots from various viewpoints are taken. Because the coordinate reference system remains constant while the app is running, the object center coordinates remain fixed for each snapshot. For small objects like mice, the marker is placed on a table, a snapshot is taken, and the mouse is then centered on it. A Python script is used to detect the marker with OpenCV, then it creates a mask that separates the object's center vertices from the point cloud of the snapshot. Object's centroid coordinates are obtained from the 3D bounding box of those vertices. Then, using the same procedure as described in Section 5.2, the category position error, expressed in meters, is determined.

5.4 Vertical Rotation Error

The GT value of the object vertical rotation is determined using the diamond marker, in this case it used to obtain the camera pose. The marker is positioned and aligned with the object (as shown in Figure 5.1), a first snapshot is taken, then the marker is removed and a second snapshot is taken, this last one will be used for the rotation estimation performed by the proposed system. This is repeated for multiple

(a) Object with marker **(b)** The estimated pose

Figure 5.1: (a) Captured photo showing an object with the marker used to determine the ground truth pose; (b) the estimated pose of the same object obtained with the proposed system.

rotations for every object. The smartphone is held to a fixed support during the evaluation process to ensure consistency. The GT rotation is obtained by calculating the camera pose rotation this is done using a Python script and OpenCV. The *estimatePoseSingleMarkers()* function takes as input the corners of the detected marker, the camera matrix and the distortion coefficients obtained during the calibration. A rotation and translation vector is returned. So, it is now possible to transform points from the marker coordinate system to the camera coordinate system. The origin of the marker coordinate system is at the center of it, with the vertical axis perpendicular to the marker plane. In order to get the Euler angles of the pose, the rotation vector is converted to a rotation matrix using Rodrigues' rotation formula, then its transpose is flipped and a formula similar to [61] is applied. Finally, the rotational error is calculated by averaging the RMSE values in degrees.

	Avg. RMSE Position (m)			**Avg. RMSE Scaling (%)**			**Avg. RMSE Scaling Factor Error (%)**	**Avg. RMSE Rotation (°)**
Object (PS/R)	x	y	z	x	y	z	(x, y, z)	y
Chair (61/11)	0.03	0.02	0.02	10	2	13	10	9
Swivel chair (24/6)	0.02	0.02	0.03	6	6	12	8	15
Armchair (33/8)	0.05	0.02	0.07	13	4	12	10	9
Table (62/9)	0.05	0.02	0.04	7	3	10	7	12
Nightstand (31/6)	0.04	0.06	0.05	13	7	23	14	18*
Chest of drawers (12/7)	0.03	0.07	0.08	6	5	8	6	8
Cabinet (57/5)	0.06	0.03	0.05	15	6	14	12	14
Bed (24/7)	0.09	0.04	0.1	7	8	17	10	14
Laptop (22/12)	0.02	0.02	0.02	7	8	21	12	10
Bowl (17/-)	0.05	0.03	0.04	14	22	7	14	-
Mouse (25/6)	0.02	0.01	0.02	11	26	11	16	2*
Remote (22/5)	0.03	0.02	0.01	6	20	25	17	6*
Book (33/8)	0.02	0.01	0.02	17	23	15	18	6*

Table 5.1: The main outcomes. The average of the root mean squared errors of position, scaling, vertical rotation and the average size error in 3D. Objects marked with * may have 180° rotation error. PS and R stand for the number of position-scaling and rotation snapshots, respectively.

Chapter 6

Discussion

The position, rotational and scaling errors are shown in Table 5.1. The position and scaling errors appear to increase along the Y axis when the object's surface is not flat or when obstacles partially hide the object's surface, disturbing the depth map estimation algorithm. While minor occlusions, such as a laptop covering a portion of a table, are tolerated, a heavily occluded table may result in a segmentation failure. Regarding the rotation error, some inconsistencies in objects with symmetrical shapes have been detected, resulting in incorrect of 180° rotations around the vertical axis. These incorrect predictions appear to occur independently of the dataset used to train the network. Generally, errors increase when the object is not completely captured in the snapshot and thus the 3D oriented bounding box does not perfectly align to the object's segmented point cloud. Matte black surfaces present a challenge because the stereo matching algorithm [44] is unable to establish consistent correspondences between frames composed of a large number of dark pixels, resulting in reconstruction errors and depth map artifacts. Since the proposed system reconstructs a scene using a single snapshot for each object, it is critical to capture the object from perspectives that encompass all of the object's essential components (e.g., to estimate the position and size of a chair, the seat should be visible). It has been observed that the segmentation network may incorrectly classify chests of drawers and nightstands as cabinets. This is most likely due to the fact that the HRNet-evaluated image

regions lack sufficient information to distinguish between certain classes of objects with similar characteristics [45].

Lastly, the performance of the system was tested using a laptop equipped with an Intel Core i7-8750h CPU and an NVIDIA RTX 2060 GPU. The proposed solution reconstructs a scene with five objects in 21 seconds, including 15 seconds for segmentation, nearly 2 seconds for 3D model retrieval, less than 2 seconds for object processing, and 2.25 seconds for pose inference.

6.1 Conclusion

This thesis describes a system for reconstructing a scene captured using an Android smartphone equipped with ARCore. In comparison to the state of the art, the proposed solution makes use of only the RGB camera, eschewing the use of traditional depth sensors. The system is comprised of an ARCore-based application that captures snapshots of objects in an indoor environment from a single view. The snapshots are then processed on a server: the target object in the frame is classified, and the most similar 3D model from a database is retrieved; the object's scale, position, and vertical rotation are then estimated. All DL modules are replaceable as newer, more effective solutions become available.

Additionally, three Unity applications have been shown: 1) a desktop application that populates the virtual scene with instances of 3D models based on the estimated pose without deforming the object meshes, and 2) an augmented reality application that overlays the virtual objects over their real-world counterparts 3) a virtual reality application for Oculus Quest 2 that allows a VR user to move in the digital scene using an avatar. These apps can be used in online multiplayer mode, visualizing the desktop/VR user's AR instance from the smartphone and vice versa the tracked AR user's desktop/VR instance in the reconstructed scene,

thereby providing a shared sense of space, presence, and time [62].

Multiple techniques have been described to circumvent the limitations imposed by single view snapshots and low resolution depth maps. To evaluate the system's accuracy, a dataset of over 500 snapshots was introduced.

6.2 Future works

Future work will incorporate a larger model dataset, which, when combined with non-trivial mesh deformation, may enable more precise shape retrieval. Additionally, the scene layout will be determined, allowing for the detection and reconstruction of walls, floors, and ceilings. Finally, other methods for detecting rotations around any axis that overcome the vertical rotation limit could be investigated.

Bibliography

[1] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. «A Survey on Deep Learning: Algorithms, Techniques, and Applications». In: *ACM Comput. Surv.* 51.5 (Sept. 2018). ISSN: 0360-0300. DOI: 10.1145/3234150. URL: https://doi.org/10.1145/3234150 (cit. on p. 4).

[2] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Naveed Asghar, and Brian A. Lee. «A Survey of Modern Deep Learning based Object Detection Models». In: *CoRR* abs/2104.11892 (2021). arXiv: 2104.11892. URL: https://arxiv.org/abs/2104.11892 (cit. on p. 4).

[3] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. «Image Segmentation Using Deep Learning: A Survey». In: *CoRR* abs/2001.05566 (2020). arXiv: 2001.05566. URL: https://arxiv.org/abs/2001.05566 (cit. on p. 4).

[4] George Fahim, Khalid Amin, and Sameh Zarif. «Single-View 3D reconstruction: A Survey of deep learning methods». In: *Computers Graphics* 94 (2021), pp. 164–190. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2020.12.004 (cit. on p. 4).

[5] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. «Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era». In: *CoRR* abs/1906.06543 (2019). arXiv:

1906.06543. URL: http://arxiv.org/abs/1906.06543 (cit. on p. 4).

[6] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. «Augmented reality: a class of displays on the reality-virtuality continuum». In: *Telemanipulator and Telepresence Technologies*. Ed. by Hari Das. Vol. 2351. International Society for Optics and Photonics. SPIE, 1995, pp. 282–292. DOI: 10.1117/12.197321. URL: https://doi.org/10.1117/12.197321 (cit. on p. 9).

[7] Florence Aïm, Guillaume Lonjon, Didier Hannouche, and Rémy Nizard. «Effectiveness of Virtual Reality Training in Orthopaedic Surgery». In: *Arthroscopy: The Journal of Arthroscopic Related Surgery* 32.1 (2016), pp. 224–232. ISSN: 0749-8063. DOI: https://doi.org/10.1016/j.arthro.2015.07.023. URL: https://www.sciencedirect.com/science/article/pii/S0749806315006489 (cit. on p. 10).

[8] R.T. Azuma. «'A Survey of Augmented Reality', (6, 4).» In: *Presence: Teleoperators and Virtual Environments* 6.4 (1997) (cit. on p. 10).

[9] Philip Pratt, Matthew Ives, Graham Lawton, Jonathan Simmons, Nasko Radev, Liana Spyropoulou, and Dimitri Amiras. «Through the HoloLens™ looking glass: augmented reality for extremity reconstruction surgery using 3D vascular models with perforating vessels». In: *European radiology experimental* 2.1 (2018), pp. 1–7 (cit. on p. 11).

[10] *LiDAR scan using Polycam.* https://sketchfab.com/3d-models/living-room-d9c6ea856ef64db79d426d88101ad5ed. Accessed: 2021-12-1 (cit. on p. 13).

[11] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. «State of the Art on 3D Reconstruction with RGB-D Cameras». In: *Computer Graphics Forum (Eurographics State of the Art Reports 2018)* 37.2 (2018) (cit. on p. 14).

[12] Yunpeng Xiao, Yu-Kun Lai, Fang-Lue Zhang, Chunpeng Li, and Lin Gao. «A survey on deep geometry learning: From a representation perspective». In: *Computational Visual Media* 6 (2020), pp. 113–133 (cit. on pp. 15, 16).

[13] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. «PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space». In: *CoRR* abs/1706.02413 (2017). arXiv: 1706.02413. URL: http://arxiv.org/abs/1706.02413 (cit. on p. 15).

[14] Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. «Occupancy Networks: Learning 3D Reconstruction in Function Space». In: *CoRR* abs/1812.03828 (2018). arXiv: 1812.03828. URL: http://arxiv.org/abs/1812.03828 (cit. on pp. 16, 18).

[15] Maxim Tatarchenko, Stephan R. Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. *What Do Single-view 3D Reconstruction Networks Learn?* 2019. arXiv: 1905.03678 [cs.CV] (cit. on p. 17).

[16] Justin Johnson Georgia Gkioxari Jitendra Malik. «Mesh R-CNN». In: *ICCV 2019* (2019) (cit. on p. 18).

[17] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. «AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation». In: *CoRR* abs/1802.05384 (2018). arXiv: 1802.05384. URL: http://arxiv.org/abs/1802.05384 (cit. on p. 18).

[18] Ayan Sinha, Asim Unmesh, Qixing Huang, and Karthik Ramani. «SurfNet: Generating 3D shape surfaces using deep residual networks». In: *CoRR* abs/1703.04079 (2017). arXiv: 1703.04079. URL: http://arxiv.org/abs/1703.04079 (cit. on p. 18).

[19] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. «BSP-Net: Generating Compact Meshes via Binary Space Partitioning». In: *CoRR* abs/1911.06971 (2019). arXiv: 1911.06971. URL: http://arxiv.org/abs/1911.06971 (cit. on p. 18).

[20] Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. «SDM-NET: Deep Generative Network for Structured Deformable Mesh». In: *CoRR* abs/1908.04520 (2019). arXiv: 1908.04520. URL: http://arxiv.org/abs/1908.04520 (cit. on p. 18).

[21] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. «PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. June 2020 (cit. on p. 18).

[22] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomır Mech, and Ulrich Neumann. «DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction». In: *CoRR* abs/1905.10711 (2019). arXiv: 1905.10711. URL: http://arxiv.org/abs/1905.10711 (cit. on p. 18).

[23] Chen-Hsuan Lin, Chaoyang Wang, and Simon Lucey. «SDF-SRN: Learning Signed Distance 3D Object Reconstruction from Static Images». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020 (cit. on p. 18).

[24] Yinyu Nie, Xiaoguang Han, Shihui Guo, Yujian Zheng, Jian Chang, and Jian Jun Zhang. «Total3DUnderstanding: Joint Layout, Object Pose and Mesh Reconstruction for Indoor Scenes From a Single Image». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on p. 18).

[25] *Mask2CAD ECCV 2020 Full Video.* `https://www.youtube.com/watch?v=S7fvrQiBAcg`. Accessed: 2021-12-1 (cit. on p. 19).

[26] Hamid Izadinia, Qi Shan, and Steven M. Seitz. «IM2CAD». In: *CoRR* abs/1608.05137 (2016). arXiv: `1608.05137`. URL: `http://arxiv.org/abs/1608.05137` (cit. on p. 18).

[27] Wei-Cheng Kuo, A. Angelova, Tsung-Yi Lin, and Angela Dai. «Mask2CAD: 3D Shape Prediction by Learning to Segment and Retrieve». In: *ECCV*. 2020 (cit. on pp. 19, 20).

[28] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. «ORB-SLAM: A Versatile and Accurate Monocular SLAM System». In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: `10.1109/TRO.2015.2463671` (cit. on p. 19).

[29] Kejie Li et al. «FroDO: From Detections to 3D Objects». In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 14708–14717 (cit. on p. 20).

[30] Kejie Li, Hamid Rezatofighi, and Ian Reid. *MOLTR: Multiple Object Localisation, Tracking, and Reconstruction from Monocular RGB Videos*. 2021. arXiv: `2012.05360 [cs.CV]` (cit. on p. 20).

[31] K. Maninis, S. Popov, M. Nießner, and V. Ferrari. «Vid2CAD: CAD Model Alignment using Multi-View Constraints from Videos». In: *ArXiv* abs/2012.04641 (2020) (cit. on p. 20).

[32] A. Avetisyan, Tatiana Khanova, C. Choy, D. Dash, Angela Dai, and M. Nießner. «SceneCAD: Predicting Object Alignments and Layouts in RGB-D Scans». In: *ArXiv* abs/2003.12622 (2020) (cit. on pp. 21, 22).

[33] Angela Dai, Christian Diller, and Matthias Niessner. «SG-NN: Sparse Generative Neural Networks for Self-Supervised Scene Completion of RGB-D Scans». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on p. 21).

[34] Ji Hou, Angela Dai, and Matthias Nießner. *RevealNet: Seeing Behind Objects in RGB-D Scans*. 2020. arXiv: 1904.12012 [cs.CV] (cit. on p. 21).

[35] Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang, and Matthias Nießner. «Scan2CAD: Learning CAD Model Alignment in RGB-D Scans». In: *CoRR* abs/1811.11187 (2018). arXiv: 1811.11187. URL: http://arxiv.org/abs/1811.11187 (cit. on p. 21).

[36] Angel X. Chang et al. «ShapeNet: An Information-Rich 3D Model Repository». In: *CoRR* abs/1512.03012 (2015). arXiv: 1512.03012. URL: http://arxiv.org/abs/1512.03012 (cit. on pp. 23, 42).

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. «3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction». In: *CoRR* abs/1406.5670 (2014). arXiv: 1406.5670. URL: http://arxiv.org/abs/1406.5670 (cit. on p. 23).

[38] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. «Indoor Segmentation and Support Inference from RGBD Images». In: *ECCV*. 2012 (cit. on p. 23).

[39] S. Song, S. P. Lichtenberg, and J. Xiao. «SUN RGB-D: A RGB-D scene understanding benchmark suite». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 567–576. DOI: 10.1109/CVPR.2015.7298655 (cit. on pp. 23, 29).

[40] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. «ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes». In: *CoRR* abs/1702.04405 (2017). arXiv: 1702.04405. URL: http://arxiv.org/abs/1702.04405 (cit. on p. 23).

[41] Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J. Guibas. «Joint Embeddings of Shapes and Images via CNN Image Purification». In: *ACM Trans. Graph.* 34.6 (Oct. 2015). ISSN: 0730-0301. DOI: 10.1145/2816795.2818071. URL: https://doi.org/10.1145/2816795.2818071 (cit. on p. 23).

[42] Tang Lee, Yen-Liang Lin, Hungyueh Chiang, Ming-Wei Chiu, Winston Hsu, and Polly Huang. «Cross-Domain Image-Based 3D Shape Retrieval by View Sequence Learning». In: *2018 International Conference on 3D Vision (3DV)*. 2018, pp. 258–266. DOI: 10.1109/3DV.2018.00038 (cit. on p. 24).

[43] Huan Fu, Shunming Li, Rongfei Jia, Mingming Gong, Binqiang Zhao, and Dacheng Tao. «Hard Example Generation by Texture Synthesis for Cross-domain Shape Similarity Learning». In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 14675–14687. URL: https://proceedings.neurips.cc/paper/2020/file/a87d27f712df362cd22c7a8ef823e987-Paper.pdf (cit. on p. 24).

[44] Julien Valentin et al. «Depth from motion for smartphone AR». In: *ACM Transactions on Graphics* (2018). URL: https://dl.acm.org/citation.cfm?id=3275041 (cit. on pp. 26, 29, 57).

[45] Ke Sun et al. «High-Resolution Representations for Labeling Pixels and Regions». In: *CoRR* abs/1904.04514 (2019). arXiv: 1904.04514. URL: http://arxiv.org/abs/1904.04514 (cit. on pp. 29, 30, 58).

[46] John Lambert, Zhuang Liu, Ozan Sener, James Hays, and Vladlen Koltun. «MSeg: A Composite Dataset for Multi-domain Semantic Segmentation». In: *Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 29).

[47] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (cit. on p. 29).

[48] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. «Semantic Understanding of Scenes through the ADE20K Dataset». In: *CoRR* abs/1608.05442 (2016). arXiv: 1608.05442. URL: http://arxiv.org/abs/1608.05442 (cit. on p. 29).

[49] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. «The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes». In: *International Conference on Computer Vision (ICCV)*. 2017. URL: https://www.mapillary.com/dataset/vistas (cit. on p. 29).

[50] Girish Varma, Anbumani Subramanian, Anoop M. Namboodiri, Manmohan Chandraker, and C. V. Jawahar. «IDD: A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments». In: *CoRR* abs/1811.10200 (2018). arXiv: 1811.10200. URL: http://arxiv.org/abs/1811.10200 (cit. on p. 29).

[51] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. «BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling». In: *CoRR* abs/1805.04687 (2018). arXiv: 1805.04687. URL: http://arxiv.org/abs/1805.04687 (cit. on p. 29).

[52] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. «The Cityscapes Dataset for Semantic Urban Scene Understanding». In: *CoRR* abs/1604.01685 (2016). arXiv: 1604.01685. URL: http://arxiv.org/abs/1604.01685 (cit. on p. 29).

[53] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. «Open3D: A Modern Library for 3D Data Processing». In: *arXiv:1801.09847* (2018) (cit. on p. 30).

[54] Karen Simonyan and Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: *International Conference on Learning Representations*. 2015 (cit. on pp. 31, 42).

[55] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575 (cit. on p. 32).

[56] Yang Xiao, Xuchong Qiu, Pierre-Alain Langlois, Mathieu Aubry, and Renaud Marlet. «Pose from Shape: Deep Pose Estimation for Arbitrary 3D Objects». In: *British Machine Vision Conference (BMVC)*. 2019 (cit. on pp. 32, 45).

[57] Ruofei Du et al. «DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality». In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and*

Technology. UIST. ACM, 2020. DOI: 10.1145/3379337.3415881 (cit. on pp. 32, 35).

[58] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. «Open3D: A Modern Library for 3D Data Processing». In: *CoRR* abs/1801.09847 (2018). arXiv: 1801.09847. URL: http://arxiv.org/abs/1801.09847 (cit. on p. 38).

[59] *OBB: Minimum-Volume Box Containing a Set of Points*. https://www.geometrictools.com/Documentation/MinimumVolumeBox.pdf. Accessed: 2020-11-30 (cit. on p. 38).

[60] J. Canny. «A Computational Approach to Edge Detection». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986), pp. 679–698. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1986.4767851 (cit. on p. 40).

[61] *Matlab Rotm2eul: converts a rotation matrix to the corresponding Euler angles*. https://it.mathworks.com/help/robotics/ref/rotm2eul.html. Accessed: 2021-09-30 (cit. on p. 55).

[62] Sandeep K. Singhal and Michael Zyda. *Networked virtual environments - desgin and implementation*. Addison-Wesley-Longman; ACM Press, 1999. ISBN: 978-0-201-32557-7 (cit. on p. 59).