POLITECNICO DI TORINO

Master's Degree in Computer Science Engineering

Master's Degree Thesis

AUTOMATIC RECONSTRUCTION OF INDOOR ENVIRONMENTS FOR SHARING AR AND VR SPACES

Supervisor: **Candidate:**

Prof. ANDREA SANNA **FABRIZIO RONZINO**

March 2021

Contents

1 Introduction

1.1 How the world view is changing

When we have to consider how much the digital technology is present in our lives, we cannot deny that we rely on it for every kind of activity. Online purchases, organizing a trip, searching for the closest restaurant, every moment of our day is influenced by technology in a way that would be unbelievable 20 years ago, and sometimes we are so used to it that we underestimate its importance.

Everyday we take a step forward building something that could help someone, in a faster and better way; the impact of the progress, which is only possible thanks to the continuous research in academia and industry, often motivated by the love for new discoveries, is the power which motivates people to go beyond their limits.

What has been tried to realize during this thesis experience is to add a little brick, a little knowledge in order to make a little step forward.

What really motivated in the choice of the topic was the direction that the world is taking these years: everything is automated and does not need human operators; some machines are even able to perform complex tasks, for instance generating a picture from a textual description, that until a few years ago only the most evolved living beings could perform. These machines exploit the so called Artificial Intelligence (AI). This could scare someone, but we have to consider two things: the first one is that this new technology allows us to solve problems that could not be solved in the past, achieving results close or sometimes even better than what humans can do; the second aspect is that this approach has been built observing how we humans learn from experience. Exploiting the way humans learn and think, we can build sophisticated systems and apply this main concept to a high number of different areas.

What it was intended to realize is to gather what is innovative and with large possibility of development, such as Machine Learning (ML), which is a subset of AI, that can be used to solve non trivial problems by means of a computer, to something really interesting such as Computer Graphics and Computer Vision; the way these disciplines can work together is really challenging.

Figure 1.1: Examples of different Computer Vision applications. Images taken from [30].

1.2 The Project Idea

Considering two main branches of Computer Graphics such as Augmented Reality (AR) and Virtual Reality (VR) as the main objective of this thesis, it has been decided to design and develop a system capable of reconstructing, as 3D models, the objects of an indoor environment; the system is composed by an Android smartphone, which can be used by users in order to take photos of the objects that they want to reconstruct, and by a server that processes the acquired pic-

tures as the input of the reconstruction pipeline that will be described in chapter 6.

This is an attempt to give to the users a tool to digitalize the environment and manipulate it, eliminating the constraints to use prebuilt models. The system is capable of producing a watertight mesh representing an object from a single colored photo, which undergoes multiple steps of segmentation, background removal, cropping and conversion to grayscale; the system is even able to estimate the pose of the object. When the server finishes the reconstruction process, it sends the mesh to the client, where the object will be displayed in the AR environment.

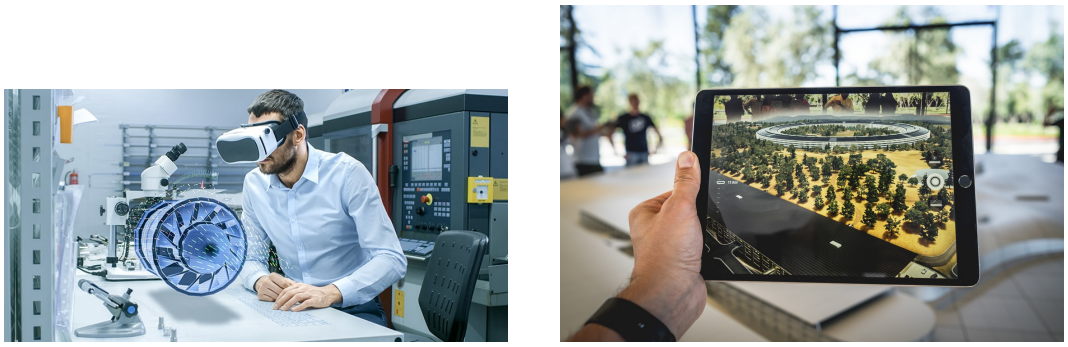

Figure 1.2: A Virtual Reality application (on the left) and an Augmented Reality application (on the right). Images taken from [17] and [12].

The first chapter consists in a brief introduction, followed by a general description of the system and its purpose in the context of the thesis. The second chapter investigates the theoretical concepts of Machine Learning, Neural Networks, Computer Graphics and Computer Vision, in order to guarantee a better comprehension of the mechanism developed in the thesis. The third chapter is a discussion of the state of the art, where different works, previously tested, are described. The fourth chapter presents the general idea of the application and its functioning, splitting the description between the application and the server side. The fifth chapter discusses in details the mechanisms of the client side and its implementation. The sixth chapter discusses the tasks performed by the server in order to achieve a successful reconstruction of a model, relying only on the image of the object. The seventh chapter discusses the results obtained in the final implementation of the server and client and the overall accuracy obtained, as well as the time employed to perform a complete cycle, from the start of the application to the final visualization of the model in a AR environment. The eighth chapter contains a discussion regarding the obtained achievements, the possible improvements and the field of application that the app can have.

2 Deep Learning and Computer Graphics

2.1 Machine Learning

If we consider Artificial Intelligence as the macro area of research in which a machine tries to emulate human behaviour according to predefined patterns in a smart way, the Machine Learning is a subset of this branch based on adaptability and learning in an autonomous way. Machine Learning is the science of teaching to a machine how to perform some specific tasks using existing knowledge that takes the form of collection of data; e.g., we can think about the problem of classifying a picture which contains a cat or a dog, or the problem of generating a picture from a text description of its subject. We usually call the process of learning from data *training*, whereas collection of data are called *datasets*. Similarly to how we process and understand reality, a machine can learn to extract meaningful features and

patterns from past observations, and store this knowledge for a later use.

One of the most interesting aspects of Machine Learning are Neural Networks (NNs), which consist in the baseline of the Deep Learning, a very popular approach which tries to imitate and reproduce the way that a human learns and thinks, reproducing a real artificial neuron-based architecture, which emulates the cognitive process of decision and recognition.

Machine Learning algorithms have different approaches depending on the kind of problems they have to solve, but they follow a general approach divided into 3 phases: training, validation and testing. The training phase is the most important one because it will define the behaviour of the software: during this phase a large dataset (e.g., a common dataset for image classification can contain thousands of images) is used to feed the machine in order to let it understand which features are very important for the task it has to solve (e.g., to distinguish between men and women, the machine can learn to recognize long hairs, as these are more common in women rather than in men), then it will store those features and (usually) the label that identifies them and will be able to recognize them in a second moment. This

phase is usually expensive in terms of time and resources. Similarly to the problem of fitting a curve given a set of noisy points, there is the risk that the machine will memorize the specific dataset and it will not be able to generalize to new observations; when this occurs it is called *overfitting*, which means that there are too many parameters in the model (i.e., the model is too complex) or too few data, leading to poor performance. The opposite problem is the so called *underfitting*, where there are too few parameters with respect to the complexity of the task that it is intended to solve.

Once the machine has learnt everything, it is able to validate its knowledge on a different dataset, usually constructed by dividing the initial dataset into two separate datasets, one big dataset for training and one smaller dataset for validation (usually, the ratio between validation and training datasets is 20% / 80%). It is important that validation is not performed using the same dataset used for training. For example, in a classification task the machine should be able to predict, for every new input, which is the correct label to assign to it (e.g., to label a dog as a dog and not as a person). One of the most important criteria to evaluate how good a machine is in performing a classification task is the accuracy, i.e., how many times the label

has been correctly assigned. The highest the accuracy, the better the classification.

Finally, the software can be tested by anyone on his own data and check whether the accuracy is good enough, which means that it can correctly detect what it sees.

In order to achieve better results in Machine Learning also the so called Transfer Learning approach can be applied, which consists in exploiting the previously acquired (through machine learning) knowledge to accelerate the learning phase of a new network. This method drastically reduces the training time (weights are already provided) and lead to better results (the final accuracy is usually higher since the weights have been already tested many times).

Another interesting technique in ML is Data Augmentation, which is used to solve the problem of finding a lot of samples to populate the dataset. Data Augmentation can generate new samples alongside the original ones by means of cropping, filtering, rotating the original samples, preserving the image-label association and leading to better final results since the dataset will be bigger and more varied.

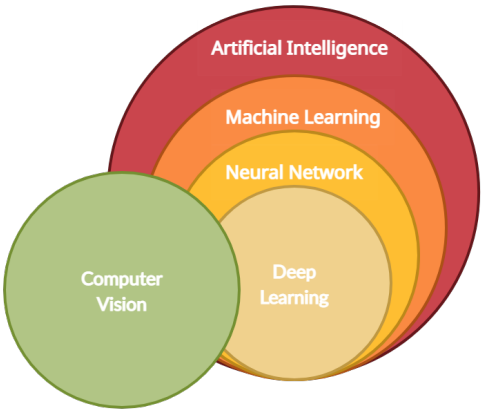

Figure 2.1: Venn Diagram of the relation between the Artificial Intelligence and the Computer Vision.

2.2 Neural Networks

An important step to take into account in order to understand the base of this application is the explanation of what a Neural Network is and how it works. A Neural Network can be defined as a complex configuration inspired by the biological network of the animals and is designed to simulate its way of learning using a non-linear configuration. The system improves its performance and continuously evolves its structure, adapting according to what we feed. Every NN is composed by a variable amount of processing units or "neurons", each one belonging to a specific layer and connected to other neurons

of other layers through an input/output chain system. Every neuron contains an internal state, called activation function, that is used to construct the signal to send to the next ones and defines how the information will be transferred; the knowledge is stored according to a specific weight associated to it. Considering a generic neuron X_0 with a weight w_0, which has as inputs other two neurons X_1 and X_2 with weights of respectively w_1 and w_2, it will have an input x_{in} equal to $y_{in} = w_1x_1 + w_2x_2$, then its output will be given by his activation function as function of his input and its weight as:

$$y_{out} = f(x_{in}, w_0) = w_0(w_1x_1 + w_2x_2) \tag{2.1}$$

Hence the neurons are arranged by layers and are connected to the other layer's neurons in a sequential mode. The NN deals with multiple layers interconnected to a single input and output, or with layers fully connected (i.e., each neuron is directly linked to each other neuron of the next layer), or we can crop some interconnections with ad hoc techniques (e.g., drop out, normalization) in order to speed up the computation and to improve the learning process. Every processing unit makes decisions based on a weighted system that is used to

produce an output for the next neuron in the successive layer or to the final output. These weights used to take decisions are not static, in fact the aspect of the NN similar to the units is the capability of adjusting the weights of the neurons that are multiplied with inputs, i.e., it is learning. The Network is structured into three main levels: the input level, where all the information from external sources are fed into the system and passed to the successive layers; the hidden layer, which contains every stack of neurons that computes the inputs and forward the elaborated output to the next layer (e.g., the Multi-Layer-Perceptron) or directly to the output (e.g., the Single-Layer-Perceptron). Finally, all the elaborated features are passed to the output layer which produces a final outcome.

The overall system can improve its performance using a system called back-propagation on the neurons. It consists in a reverse mechanism based on mathematical derivatives which adjusts the weights according to what has been learnt so far.

Having defined the general points of a Neural Network, we can discern different types of these. In fact, based on the workload associated to it, the NN can change the structure or the modus operandi which defines it.

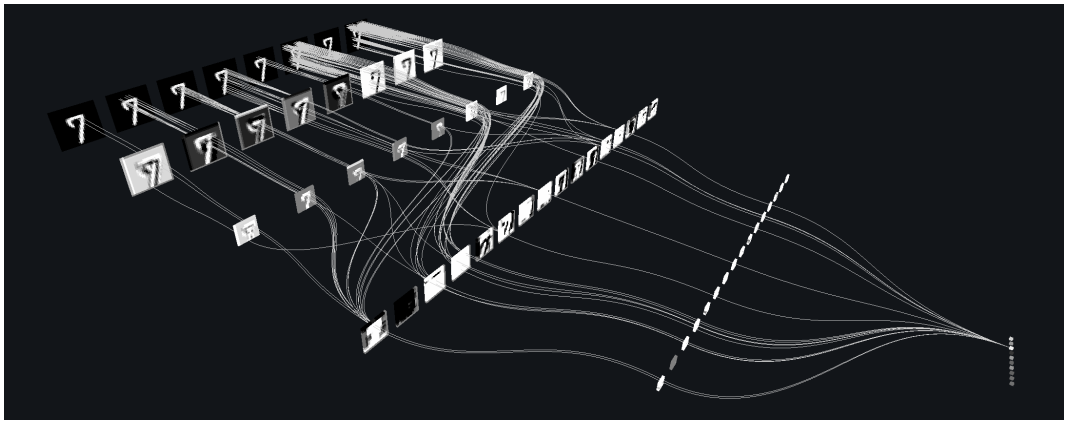

Figure 2.2: The connections in a generic structure of a Neural Network on a 3D view. The example regards the *numbers* recognition. Image taken from [8].

In the vast domain of the NN we can consider some specific categories:

1. Feedforward Neural Networks
2. Recurrent Neural Networks
3. Convolutional Neural Networks
4. Deep Neural Networks
5. Generative Adversarial Network

Feedforward Neural Networks

This the most basic type of NN where there are no cycles between neurons. The information given to the input always navigates forward with respect to the input-output logic, without storing the information. We can distinguish two variants of this structure: the Single-Layer Perceptron (SLP) and the Multi-Layer Perceptron (MLP).

The SLP consists in a single layer of output nodes. The data elaboration is provided by the single layer and does not contain any other hidden layer. This type of system is not used, since his computational capability is limited due to his non-existance capability of data combination.

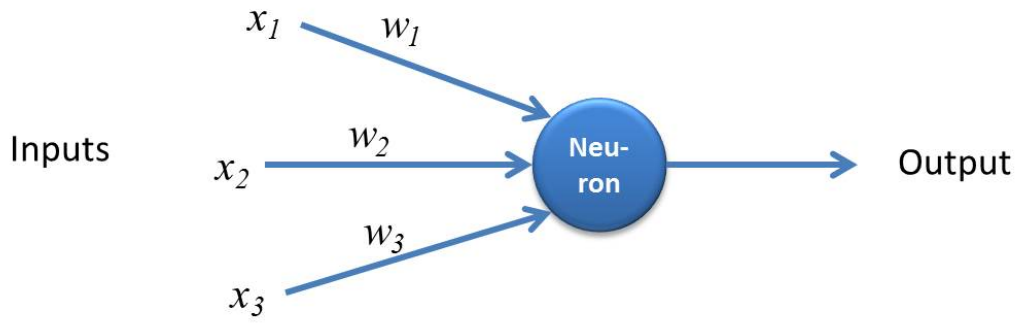

Figure 2.3: The scheme of a Single Layer Perceptron. Image taken from [3].

The MLP can be defined as the evolution of the SLP idea and it

basically means that between the input and output are defined a variable number of layers, each one with a variable number of neurons. The system increases in complexity and it is more suitable to perform more difficult tasks and predictions. The adoption of these system revolutionized the approach of all the tasks that a machine was capable of doing.

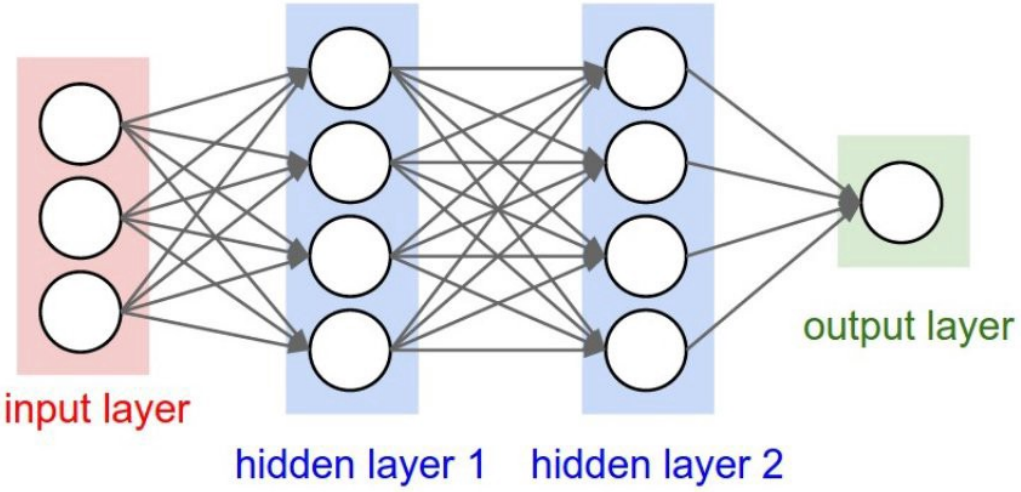

Figure 2.4: The scheme of a Multi Layer Perceptron. Image taken from [9].

Recurrent Neural Networks - RNN

Recurrent Neural Networks (RNNs) are a class of NN derived from the feedforward ones and differs for the layout of its network which

defines a graph connection between nodes, forming a sort of temporal sequence, allowing the dynamic modelling of the network. In addition, it exploits a particular type of neurons which are connected with themselves, forming a loop. This type of Networks can be inserted in a various amount of applications that led to the birth of several structures:

1. Fully recurrent RNN
2. Hopfield Network
3. Recursive RNN
4. Long short-term memory (LSTM)

We can briefly refer to the recursive neural networks, which are networks where a static set of weights is applied recursively to a structured input, with the goal of producing a structured prediction. Those has been used to learn the logical terms and the distributed representations.

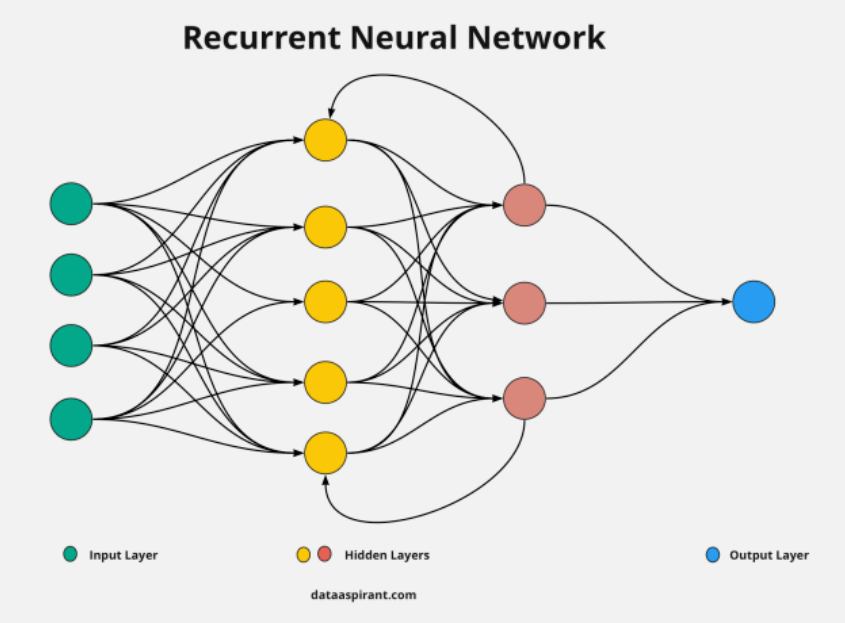

Figure 2.5: The scheme of a generic Recurrent Neural Network. Image taken from [13].

Convolutional Neural Network - CNN

The CNN is a revolutionary class of NN that is designed to learn features and spatial association in a adaptive and automatic manner using backpropagation. In this way it is possible to transfer vectors of weights and biases, which can be also shared with other neurons, and adjust them, meaning that little pre-processing is used. A typical CNN can be defined by multiple blocks such as convolutional, pooling, ReLU and fully connected layers and takes a tensor as input.

The Convolutional layers have the main scope to convolve the data

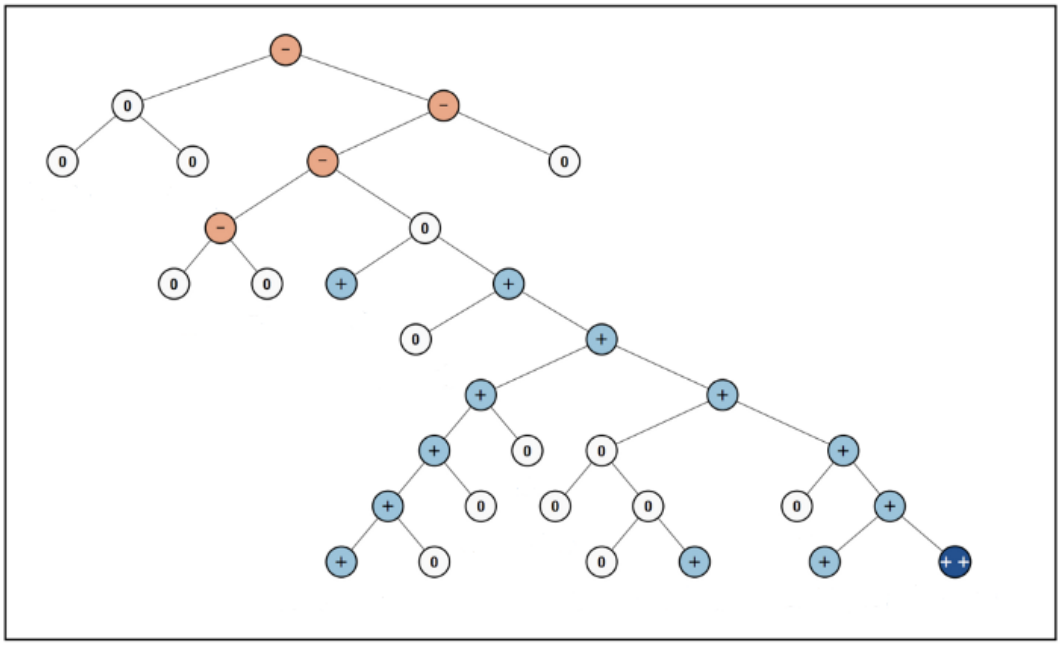

Figure 2.6: The scheme of a Recursive Neural Network. Image taken from [4].

and the final result will be a feature map, i.e., an abstraction of the inputs. The neurons of a layer process data only for its receptive field.

The Pooling Layer has the main goal to reduce the amount of data passed as input, combining the inputs of multiple neurons, and can be of two types: Max Pooling where only the max value of the outputs received by the cluster of neuron is passed, or the Average Pooling, where the passed value is the result of the average between all the inputs given to the neuron.

The fully Connected Layer is a regularized and structured version

of a MLP where every neuron in a layer is connected to all the neurons in the successive layer; this leads to slower but more accurate final outcome.

This part of the CNN is the one responsible for classification

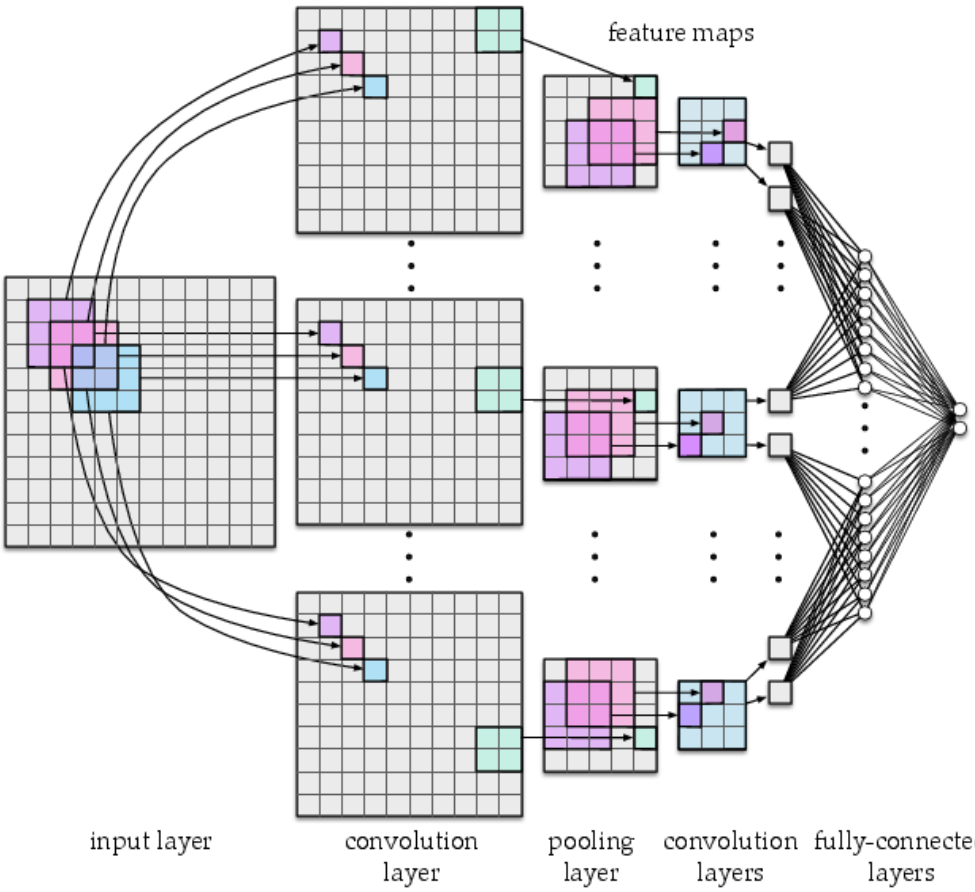

Figure 2.7: The scheme of a Convolutional Neural Network.

Generative Adversarial Networks

This framework, defined also as GAN, was designed by Ian Goodfellow and consists in two neural networks that compete each other in a contest where the gain of one of the two is a loss for the other (a form of zero-sum game). This method is developed under a single training set given to the two networks, where the goal is to generate new data with the same features with respect to the known set. The idea is that the created data should be as original as possible and appearing realistic to a human eye. To perform such action each system needs to "fool" the discriminator, which is at the same time updated dynamically in order to think that the image is real and not generated by the Network. This methods indirectly achieve a sort of "unsupervised learning" for the two networks, that constantly uses backpropagation to update its nodes and thus improve its work.

Figure 2.8: Faces reproduced by a GAN, given certain photos as input. Image taken from [11].

2.3 Computer Graphics

Computer Graphics is one of the most interesting IT branch because gathers technical aspects and creative contents. It is based on the creation and edit of images, video and videogames, the manipulation of 3D models, the emulation of a natural environment and its physics into a digitalized context. When we consider digital images, we usually refer to them in terms of pixels forming a matrix with a certain resolution, or via vectorial representation as a set of geometrical primitives. For what concerns 3D representation, we can reproduce them

as a solid model which owns all the physical aspects of a real object, such as mass and volume, or simply by a polygonal surface, e.g. a polygonal mesh, which consists of a perimetrical view of the object but empty inside.

Virtual Reality (VR) and Augmented Reality (AR) are two ways to simulate the real world into an artificial context by means of a digital device. We can think about the Oculus Rift or the Microsoft HoloLens as two devices which grant a VR or AR experience. Applications of these new technologies is not limited to games; AR and VR are very active research topics and recent advancements are creating new possibilities, for instance a surgical operation can be faithfully simulated in a safe and virtual context before the operation on the real patient, where the surgeon can train and practice in a safe and realistic environment. The benefits from VR and AR are huge and various, the quality of the simulations is constantly enriched with details and sophisticated techniques, the digital innovation is in continuous evolution. Before starting, it is important to distinguish between VR and AR.

2.3.1 Virtual Reality

Virtual Reality is based on a complete immersion into the simulation: the user is totally surrounded by 3D models and artificial surfaces and usually he can interact with most of the objects inside the scene; the floor, the walls and the ceiling are also reproduced as meshes and the user experience reaches the highest quality.

As mentioned before, The Virtual Reality give the possibility to the user to interact in a virtual three-dimensional space and interact with it using headset, which are specific devices that replace the information that the user receive from its senses with virtual ones, allow to reconstruct a complete experience, as it if were there.

The VR evolved during the years and led to the possibility to expand thanks to the massive advancements in the computer vision and the improvement of the available computational tools. The principal apply in these years were in the videogame industry, to allow the player to have a complete experience of the game.

A different purpose is in medical context, where the students can practice on interventions in a safe and controlled environment, where they can develop practice skills without any impact in case of fail-

ure(fig. 2.9).

Another interesting application is in the Psychotherapy field, where the VR is used to cure the PSTD (post-traumatic stress disorder) of the patient.

Figure 2.9: A medical student performs a treatment simulation on a virtual patient. Image taken from [25].

Despite huge development in modern times, the VR constitutes a not completely accessible technology, due to different factor.

The main one is the computational load that have on the computers, caused by the high stream of data to process in real time, which reflects an ever higher demands in terms of performance required to be utilized. These demands are not inaccessible, but require an high level hardware in order to work at its best capability and costs, as well as the cost of the headset which is quite expensive too. The stream of

data to be computed is a crucial problem caused by the fact that affects the responsivity of the environment to the user, lacking the capability of a full immersion.

2.3.2 Augmented Reality

Augmented Reality instead mixes 3D models with real environment, "augments" the reality including only certain digital objects with which the user can usually interact. The purpose is different: we want that the user recognizes where he is since the context is important in this case, and the fruibility of the application is based on the interaction between the two worlds which are mixed into one.

It is the enrichment of sensory perception using information not achievable using the human senses. It is obtained using a smartphone camera or a computer with an adequate webcam and differs from the Virtual Reality since the user does not lose the perception of the surrounding world, but instead he or she acquires additional information from it and extends the possibility to visualize object which are not present.

The Virtual Reality offer different fields of application where it

improves the user experience showing the content of a box (as in fig. 2.10), or the staff training, or still the automatic detection and visualization of imperfections in products in a factory and so on.

Figure 2.10: The Augmented Reality Lego kiosk, where customers can visualize the final content of the product by placing the box in front of a camera. Image taken from [16].

The AR, similarly to the VR, requires a computational capability that exponentially increases with the application complexity and sometimes can be prohibitive to run on smartphones.

If on one hand VR requires its specific headset to work, the AR can run on a device with camera, despite some requirements must be fulfilled in order to be able to effectively run, as far as it requires an

efficient gyroscope to preform spacial operations and tracking, as well as the capability to create depth maps to perform mapping operation and object placement in the scene.

2.4 Computer Vision

Computer Vision (CV) is the science of using computers to acquire, analyze and understand images and videos. Common tasks that CV tries to solve are image classification, image restoration, object pose estimation and object tracking. In the last decade most of the CV research has been dominated by Neural Networks, in order to recreate the human vision and its process during interpretation of images or videos in an autonomous way, extracting features from it using deep neural networks with the integration of cam and software for the acquisition and elaboration of images, and producing a digital electric signal as an output to be elaborated. The computer vision application we want to focus on is the one concerning the 3D reconstruction methods that can be categorized based on their output representation as:

1. Voxel

2. Point Clouds

3. Meshes

4. Implicit representation

The voxels (Volumetric picture element), can be described as the equivalent of pixels in three dimensional space, where each value is associated to a regular grid. Each voxel does not have an associated position, but is based on the position relative to other voxels near it. The Point Clouds are an aggregation of points placed on a 3D space characterized by associating their position and other values such as luminosity, color and depth. Those are broadly used in the representation of large three dimensional structures when scanner or sensor 3d are used in order to minimize the storage usage. A polygonal mesh is a grid that defines an object in the space and is composed by vertices, faces and edges. The geometric form used to define a mesh are usually quadrilaterals and triangles. The Implicit representation differs from the previous ones by being a continuous (not discretized) representation (number of vertices, voxels or points). This leverages on the use of a Neural Network to create an occupancy probability of an object in the space.

3 State of the Art

3.1 The Scene Reconstruction Problem

The Reconstruction problem in Computer Vision refers to the process of understanding the shape of a specific object of the real world and reproducing it as a digitalized 3D model. Reconstructing a model implies that all its coordinates in space must be known to be able to define its profile, using various range of methods that are distinguished in two categories: the active and passive methods.

The Active Methods The Active methods directly interfere with the object with sensors as rangefinders or lasers, capable of measuring the reflected part emitting radiance towards the object and thus reconstructing its depth maps, which represent the distance from the camera to each part of the object, dividing it in various range of colors depending on the detected distance. An example of Active method is given by the TOF (Time-Of-Flight) lasers, LiDAR (Light Detection

and Ranging) as in Fig.3.1 and 3D ultrasonic sensors.

The Passive Methods The Passive methods do not interfere directly with the object but measure the radiance emitted or reflected by a surface and try to infer its structure by image understanding. This methods do not require an object, but only its photos or videos, thus, they can be used in a larger range of different cases with respect to the active ones. This category includes the highest number of machine learning applications.

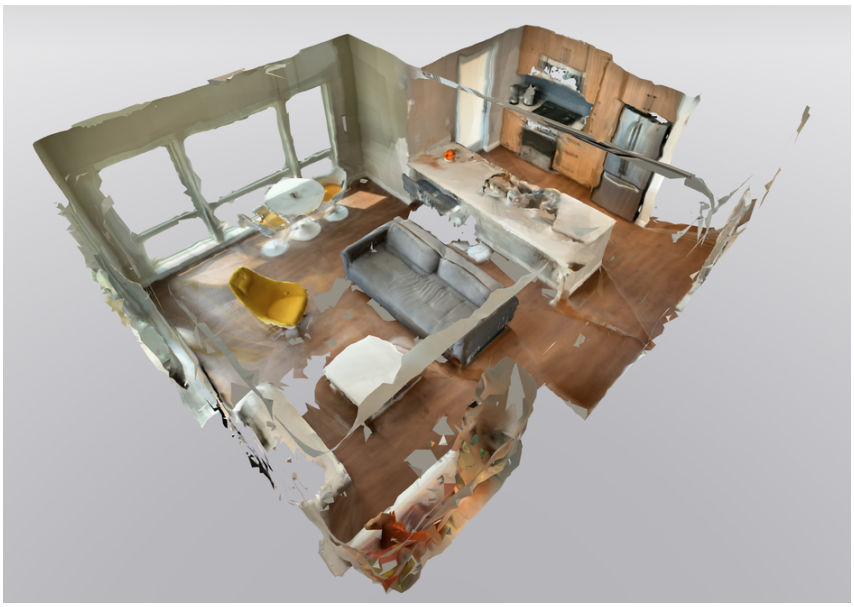

Figure 3.1: Example of Room Scanning using the Iphone 12 Pro LiDAR. Image taken from [2].

3.2 Representations of 3D Objects

In the field of Computer Graphics (CG) we have different ways to represent three-dimensional objects:

1. Voxels
2. Point clouds
3. Polygonal meshes
4. Implicit representations

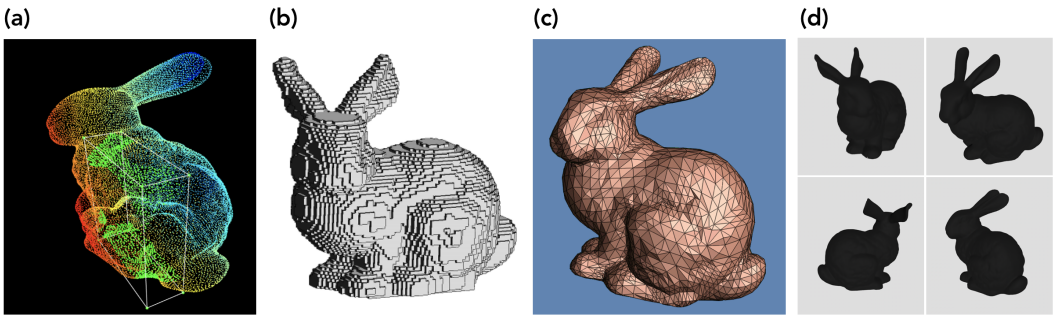

Figure 3.2: The different representation of a generic object using Point clouds (a), Voxels (b), Meshes (c) and Implicit Representations (d). Image taken from [14].

Voxels The term Voxel means Volumetric picture element. It can be described as the 3D equivalent of 2D image pixels, where each value

is associated to a regular grid in three dimensional space. A voxel does not have an associated position, but this is defined relying on the position relative to other voxels in its neighborhood. Interesting works on reconstructions based on voxels can be found in [21] and [28].

Point clouds Point clouds are an aggregation of points placed on a 3D space characterized by having associated their position and other values such as luminosity, color and depth. Those are broadly used in the representation of large three dimensional structures when scanner or 3D sensor are used in order to minimize the storage usage.

Polygonal meshes A polygonal mesh, or simply mesh, is a grid that defines an object in the space and it is composed by vertices, edges and faces. Faces are usually triangles or quads (polygons with 4 vertices), and sometimes particular faces with more than 4 vertices, called N-gons, are employed. The complexity and the smoothness of a mesh can be increased by dividing the mesh in more faces, at the expense of higher memory requirements. This representation was the subject of investigation in different articles such as [29], [10] and [18].

Implicit representations The Implicit representation differs from the previous type of rendering by being a continuous representation, thus not discretized in a finite quantity (number of vertices, voxels or points). This representation leverages on the use of a Neural Network to create an occupancy probability of an object in the space.

3.3 Traditional Approaches

In Computer Vision the methods which does not use the Machine Learning and Neural Networks to reconstruct the object are defined as Passive Methods. Those can also be defined as traditional since exploit the information derived directly from the object and often require the use of specific tools. Those were the first steps towards the automatic reconstruction of objects and environment whose major backlash was the necessity to be able to interact with object, meaning that were limited to the surrounding area where they were set. We can now define some of these traditional approaches.

3.3.1 Kinect Fusion

An interesting research is presented by *KinectFusion: Real-Time Dense Surface Mapping and Tracking* [20] that shows a system which accurately maps objects and indoor scenes in real-time. The entire system is performed using a Kinect sensor with and ICP (Iterative Closest Point) algorithm. A Kinect sensor is a device released by Microsoft in 2010 for playful purpose, composed by an RGB camera, a infrared camera, a 3D scanner that maps depth using structured light or time-of-flight calculation, a microphone array and a base with a motorized pivot.

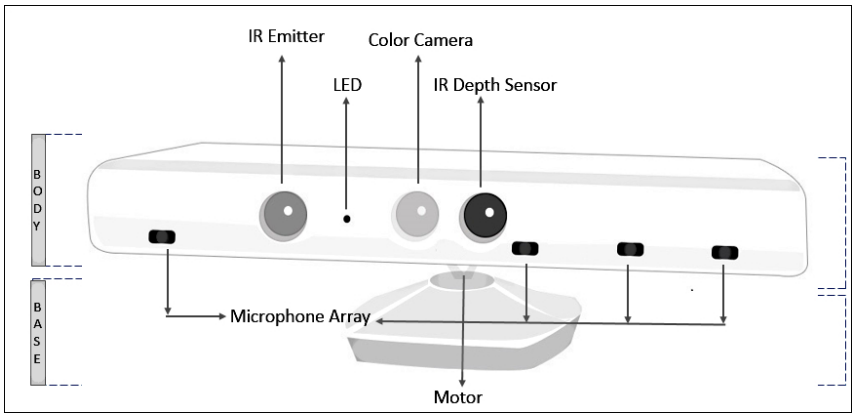

Figure 3.3: The Structure of Kinect. Image taken from [7].

The system described in the paper is structured to overcome a SLAM

(Simultaneous Localization and Mapping) problem and is composed by four modules.

The first module is the Surface Measurement, that can be defined as a pre-processing stage, where the initial transformation matrix of the camera is defined and computes the raw measurements from the Kinect device, producing a normal pyramid map and a dense vertex map.

The second module is the Sensor Pose Estimation that basically uses a multi-scale ICP to define the alignment between the current sensor measurement and the predicted surface.

The Third module is the Surface Reconstruction Update, where the depth data obtained in the first module are used in order to reconstruct a scene model that is integrated and maintained as representation of a volumetric TSDF (Truncated Signed Distance Function).

The Last module, defined as Surface Prediction, provides the information from the previous models to define a dense surface prediction and reconstruction aligned to the depth map given from the Kinect sensors. The system contains a feedback loop, where the reconstructed surface is passed again to the Sensor Pose Estimation module in order to verify the actual correspondence between the reconstruction and

the given measurement.

Figure 3.4: The results produced by the *Kinect Fusion* system.

3.3.2 AliceVision

AliceVision is a photogrammetric framework which provides a 3D reconstruction using camera tracking algorithms ([19] and [15]). It can be defined as something between the Active and Passive methods, given the fact that its reconstruction does not need the physical object to infer its reconstruction, and is not based on any of the learning-based approaches. AliceVision is based on the Photogrammetry, that is the field of science that aims to obtain reliable environment and physical object information by means of interpreting, recording and measuring images and the phenomena in it. The system is programmable as a chain of different blocks whose input is the output of the previous block and takes as input a massive quantity of

images of a single object from different points of view. The system first extracts a group of pixels from the images, which are invariant to changing camera viewpoints, using a SIFT (Scale-Invariant Feature Transform) algorithm, extracting natural features, then it uses a image matching in order to find images pointing to the same area in the scene, defining the distance from the object of interest along with the camera position. After comparing all the images, an operation of feature matching is performed with the purpose to define all features possessed by the object in order to prepare a dense scene using the information from the feature matching and a depth maps estimation, obtaining a dense pointcloud scene of the object.

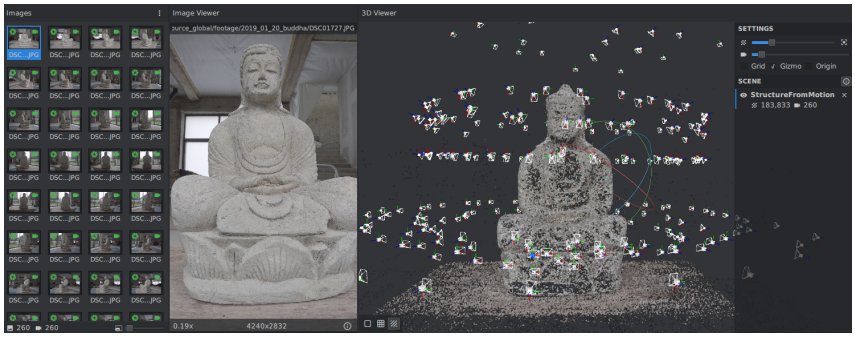

Figure 3.5: A scene reconstructed using pointclouds in AliceVision. Image taken from [1].

The system then tries to build a mesh from the pointclouds position joining all depth maps in a global octree, merging the compatible

depth values and performing a 3D Delunay tetrahedralization, producing a dense geometric surface representation. Finally, the system computes UV maps and textures of the scene and applies those results to the object. The entire process spends from thirty to forty minutes for the reconstruction of a medium dimension object and its accuracy depends from the number of photos fed to it and the gap between an orientation of a photo with respect to its previous and successive one. Normally an optimal number of photos to successfully reconstruct the system is from thirty to sixty with a gap between the photos which is not larger that ten degrees.

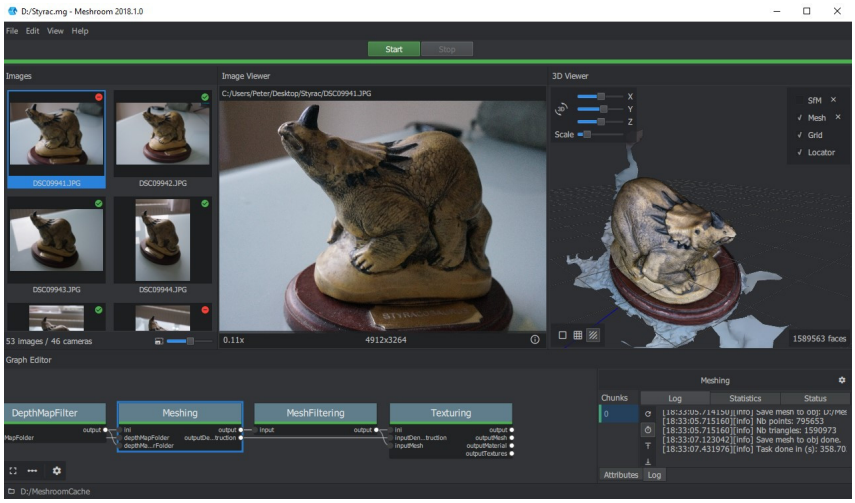

Figure 3.6: The results produced by the *AliceVision* system. Image taken from [23].

3.4 Learning-Based Methods

A method is defined learning-based when it exploits specific structures to define specific patterns from a great amount of data and is capable of predicting an outcome given specific inputs. The advantages in these methods is to be able to define autonomously a specific pattern for a given set of inputs, thus being able to have more flexibility, better accuracy and not being limited to a specific type of data. These methods have seen an enormous amount of applications.

The neural networks implemented to perform such hard tasks as classification and recognition are the result of an intensive research and advancement in the field of Machine Learning.

The Machine Learning is a branch of Artificial Intelligence which studies algorithms which allow the machine to accomplish different tasks in an autonomous manner, recognizing patterns of data. The Artificial Intelligence evolution skyrocketed when the first Artificial Neural Network (ANN or NN) began to be employed, defining a new type of learning, called deep learning, with the goal to imitate and reproduce the way a human learns and produces results based on thinking.

The process which allows the machine to learn how to perform a task is called *training*. What the training does is essentially tuning the parameters of a model in order to maximize the performance in executing the desired task. The collection of data used for training is usually called *dataset*. The training can be considered in three different types and is influenced by the type of data contained in the dataset:

1. Supervised learning
2. Unsupervised learning
3. Reinforced learning

Supervised learning Supervised learning is the most common one, and is distinguished by the use of two datasets, which differ in dimension and purpose.

The small one is a database consisting in images which were previously labeled by human and contains the right association between inputs and outputs. This aims to let the program to establish a relationship between the images. The algorithm starts to associate the label and the image through the adjustment of the weights. At the

end of the training, a refinement step is done in order to increase the accuracy of the NN by using the bigger database where it needs to associate the labels without knowing a priori the right output.

Unsupervised learning Unsupervised learning is more versatile than the supervised one. The versatility is given by the fact that the images given to the algorithm are not machine readable, that basically means that no labels are associated to them. This brings to a lack of information, forcing the algorithm to create hidden structures where the data points are perceived in abstract manners and defining connections between them.

Reinforcement learning Reinforcement learning is a peculiar algorithm whose central idea is based on the human behaviour. It consists in the implementation of a system with reward/penalty concept based on the output of the algorithm. In particular if the output is right it "rewards" the algorithm, otherwise it forces it to reiterate the procedure until a better result is provided. This method con be also defined as a trial-and-error method based on the decision of an interpreter which decides how accurate the output of the algorithm is.

In the context of machine learning, the capability of acquisition and understanding of data is performed using a singular type of algorithm with a composite structure defined as Artificial Neural Networks. Those were firstly theorized in 1943 and developed further in the years until today, where a different variation was created and implemented to perform higher grade tasks.

3.5 State-of-The-Art Systems

3.5.1 Convolutional Occupancy Network

Regarding Convolutional Occupancy Network [22], it presents an approach where the general idea consists in the reconstruction of an entire scene using the implicit representation. The system can take point clouds or voxels as input and feed them to the system, where are encoded in a feature grid, which can be 2D or 3D. This features are passed to a convolutional network which processed them and produces an occupancy probability in the space.

The system is composed by an Encoder, a Convolutional Network and a Decoder.

The Encoder The encoder is where the external inputs are first processed and varies its use depending on the type of representation fed to it. As mentioned before, the inputs can be voxels, which in this case will require the use of a 3D CNN with one layer, or Point Clouds, which will require a PointNet [24]. The encoder extracts features and maps them, constructing a planar or volumetric representation to encapsulate local information from neighborhood.

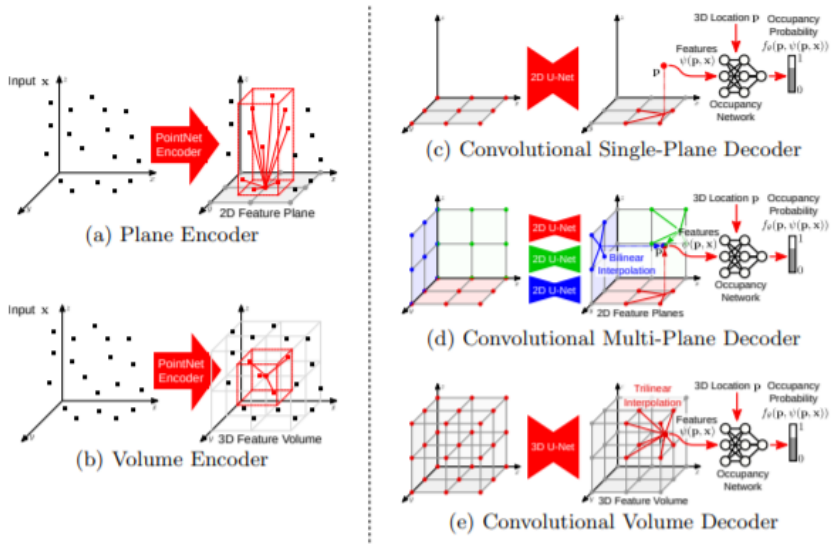

Figure 3.7: Representation of different types of encoder in the system.

The Decoder The features extracted from the Encoder are then processed through the Decoder, whose function is to process the information of feature planes and feature volumes received with the aid

of a 2D and 3D convolutional Hourglass Network (U-NET [26], [6]). The system produces feature maps in output and therefore, given the convolutional operation equivariant in translation, is possible to preserve global information to allow the reconstruction of the model from sparse inputs.

Occupancy Prediction

The feature maps give access to the estimation of the occupancy of any point p in space. The method of extraction varies accordingly to the type of decoder used that can be single-plane or multi-plane, where the first projects each point p ortographically onto the ground plane, and extracts the feature values using a bilinear interpolation. The multi-plane decoder makes a sum of all of the 3 planes features, aggregating the information from them. Then the occupancy of a point p is predicted, given a vector for input x as:

$$f_\theta(p, \phi(p, x)) \rightarrow [0, 1] \tag{3.1}$$

The final results of the reconstruction can be seen in (3.8)

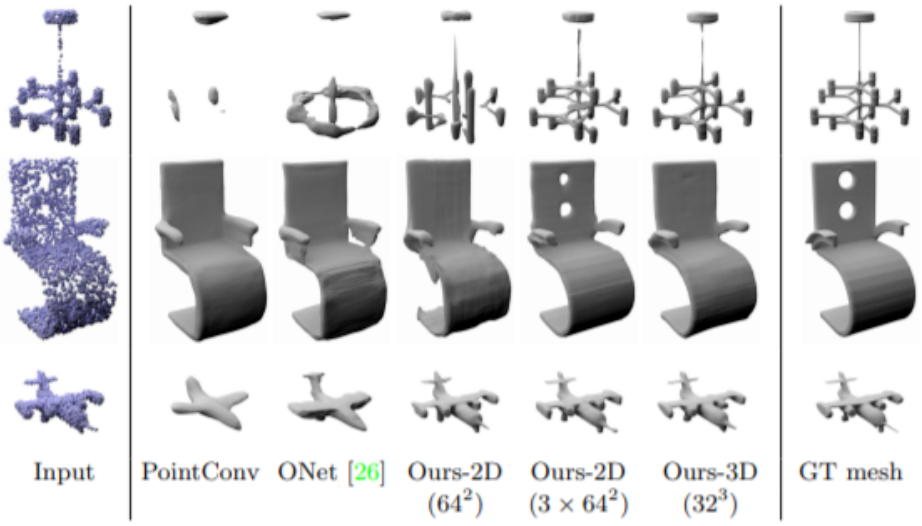

Figure 3.8: The comparison of results between different reconstruction approaches.

3.5.2 PIFuHD

PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization [27] were tested for its astonishing performance that focuses his works on Human reconstruction, producing detailed 3D models of a person, starting from a 2D image. This goal is achieved using a specific function called Pixel-Aligned Implicit Function to estimate the occupancy of a 3D dense volume. The system structure is composed by two modules, called Coarse Pixel-Aligned Implicit Function or Coarse PIFu and the Fine PIFu. An additional operation is performed at the start of the framework to predict the

front and back normal maps of the photo. In particular, the back map which is not directly observed is inferred using an MLP network that, due to the uncertainty, tends to produce smooth and featureless reconstructions. Finally, the system was abandoned due to the fact that the conversion of the system from human to object reconstruction was complex and the training time for the NN needed to acquire an acceptable result was long-lasting and required a high computational capability.

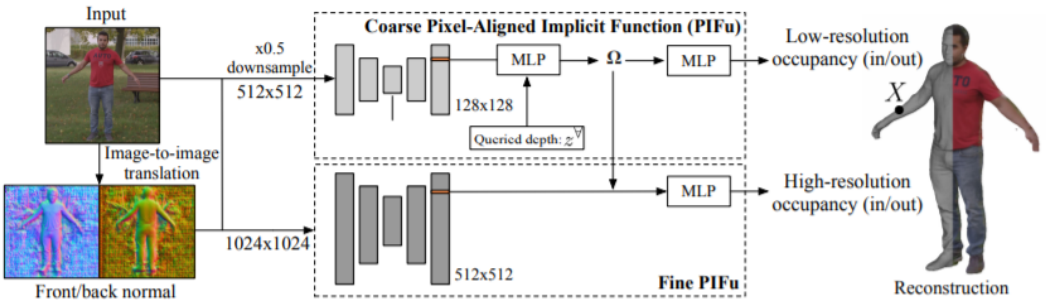

Figure 3.9: Scheme of the framework implemented in the *PIFuHD* system.

Pixel Aligned Implicit Function This function is the baseline for all the works described in the paper and its goal is to define a function capable of estimating the binary occupancy for any 3D position in a

camera space, $X = (X_x, X_y, X_z) \in \mathbb{R}^3$, given an RGB image I:

$$f(X, I) = \begin{cases} 1, & \text{if } X\text{is inside mesh surface} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

This function is modeled using a NN architecture and an end-to-end training. The final result of the function is thus expressed as:

$$f(X, I) = g(\Phi(x, I), Z), \tag{3.3}$$

where x is the orthogonal projection expressed as : $x = (X_x, X_y)$,$\Phi(x, I)$ is the image feature extracted from the 2D point location and $Z = X_z$ is defined as the depth of the ray in the 2D projection of x. As well as all the points along a ray can have different image features, a MLP is used for the 2D feature of the g function to vary the input depth Z, and a CNN is used for the 2D feature of the function Φ

Coarse PIFu The Coarse PIFu has the function to extract the global features from the image and also to produce the backbone features in a resolution of 128x128. The input of the module takes a 0.5x down-sample of the image, resulting in a 512x512 resolution. This module

differs from the standard PIFu as far as it uses also the predicted front/back normal maps. Given I_L as the low-resolution input, F_L and B_L as predicted normal maps, the function is thus defined as:

$$f^L(X) = g^L(\Phi^l(x_L, I_L, F_L, B_L,), Z) \tag{3.4}$$

Fine PIFu This module has the task of producing the details of the model, extracting them from the image. It takes as input the high resolution image (1024x1024) and also the predicted back and front normal maps. It also takes into account the 3D embedding features which are extracted from the coarse level. The function, as variation of the standard PIFu, is defined as:

$$f^H(X) = g^H(\Phi^H(x_H, I_H, F_H, B_H,), \Omega(X)), \tag{3.5}$$

where I_H, F_H, B_H represent the input image, the front normal map and the back normal map and $\Omega(X)$ represents the feature extracted from the coarse PIFu module in an intermediate level of g^L.

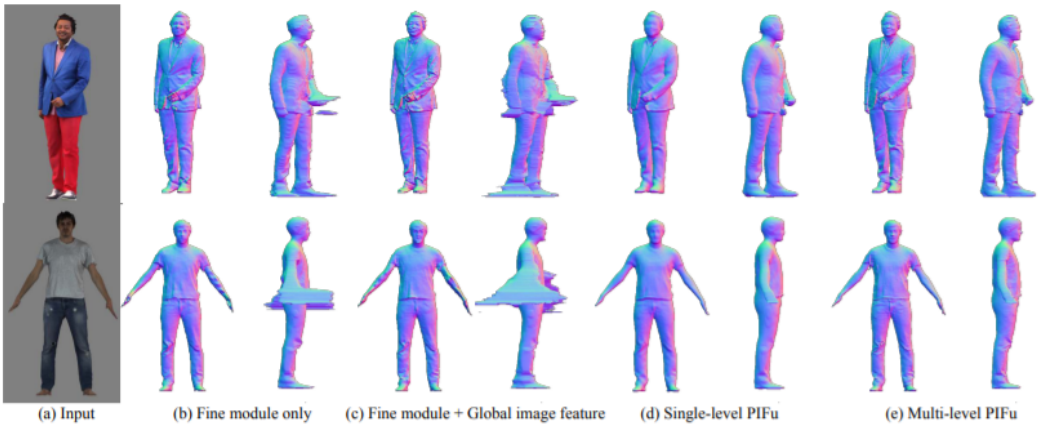

Figure 3.10: Results obtained trying different designs of the system.

3.5.3 BSP-Net

The foundation of this entire project is presented in the article *BSP-Net: Generating Compact Meshes via Binary Space Partitioning* [5] where the system defines an implicit field to indicate if a detected point is inside or outside the shape and thus reconstructing a compact polygonal mesh from a single view of a photo. The algorithm is composed by three main modules or layers that correspond to different feature vectors extracted by an encoder and reproduce the object as a model.

Hyperplane extraction module The first module has the goal to extract hyperplanes from the input data. The hyperplanes are subspace

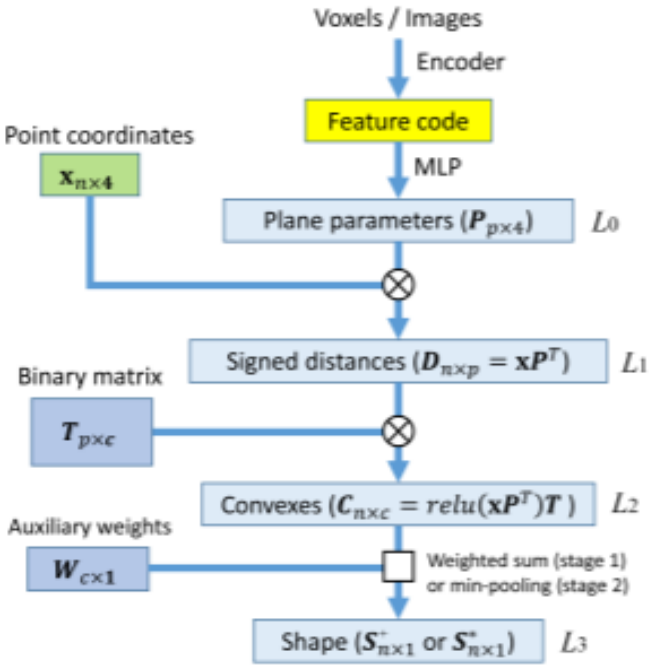

Figure 3.11: The organization of *BSP-Net*.

of an environment space, whose dimension is given by the space coordinates of that environment minus one (i.e. a Three-dimensional space will produce a hyperplane with two-dimensional planes), and are used to create learning models for classification and regression. The hyperplanes are generated by an MLP in order to obtain a vector of signed distance to each plane, exploiting the plane parameters extracted. This signed function will have negative distance if a generic point with three-dimensional coordinates is inside the plane, and pos-

itive if it is outside, considering it with respect to the normal plane.

Hyperplane grouping module This layer has the goal to create parts grouping hyperplanes in the half-spaces form. This procedure is obtained by employing a binary matrix with max pooling to form a set of convex primitives by the aggregation of input planes primitives. We can express this function as:

$$C_j^*(x) = \max_i(D_i, T_{ij}) \begin{cases} < 0, & \text{inside} \\ > 0, & \text{outside} \end{cases} \tag{3.6}$$

The Shape Assembly module

This block is designed, as the name suggests, to assemble the parts together and rebuild the object. This goal is achieved by using a min pooling in order to obtain a non-convex shape to group convexes in output, using:

$$S^*(x) = \min_i(C_i^+(x)) \begin{cases} = 0, & \text{inside} \\ > 0, & \text{outside} \end{cases} \tag{3.7}$$

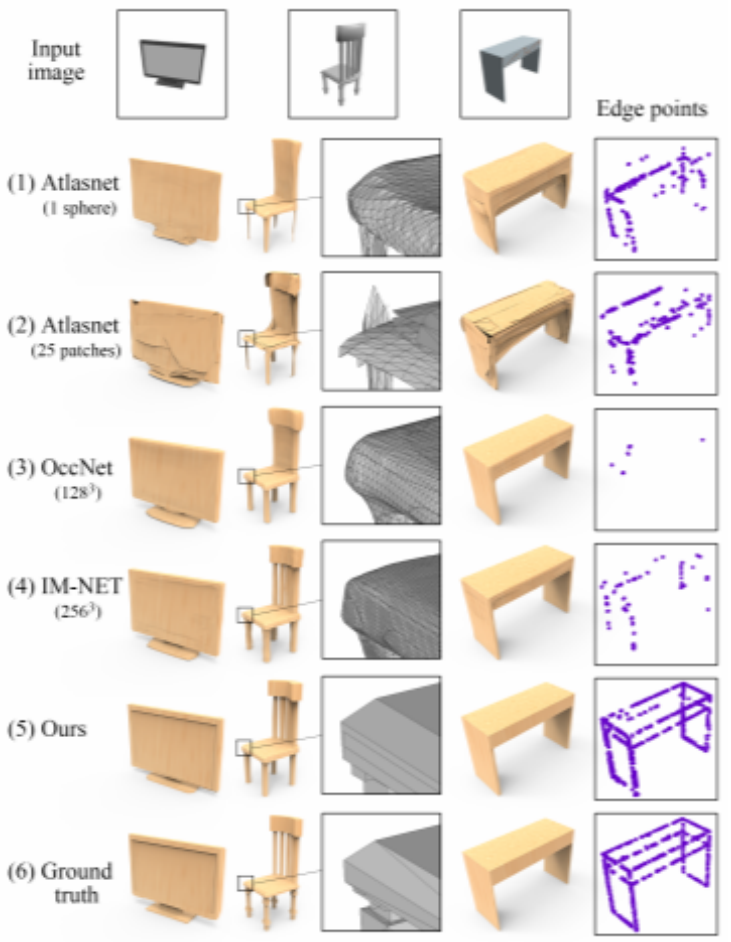

Figure 3.12: The results obtained by *BSP-Net*, compared to other object reconstruction networks and the ground truth.

4 Reconstruction System

The system was firstly designed to have specific requirements. The concept idea was to have a real-time application to perform a reconstruction of the object in an indoor room by taking a photo, and then visualize the 3D reconstruction in Augmented Reality in that same room. It was needed to split the entire project in two parts, as the computational capability of a smartphone was not enough to run the reconstruction program and the user application as well the possibility to make them work simultaneously, thus it was decided to divide the entire process in the "Acquisition and Placement Side" on the smartphone which acts as a client, and the "Reconstruction Side" on a PC which acts as a server, connecting each other through a TCP (Transfer Control Protocol) connection in order to have a reliable communication between them during the data exchange. Concerning the client side, it was important to have a reliable structure that was capable to take images and send them to the server side, as well as the possibility to reconstruct the entire object from a file or string. The real time

mesh reconstruction of an object was not to be underestimated, considering how a not complex object could store more that ten thousand lines of vertexes and faces and could be divided into different pieces, therefore there was needed a reconstruction hosting a single complete mesh of the object without material or texture to lower the complexity of reading and reconstructing. For what regards the server side, hosting the entire reconstruction program, it requires to be able to establish a connection with the application, to retrieve the image sent and to reconstruct the corresponding model. The model then should be encapsulated and returned to the application. The critical point of this system is the necessity to spend as little time as possible for image processing and object reconstruction. The best possible option is to have a system capable to perform those action in around one minute. The starting step towards the realization of the project is to agree about a suitable reconstruction system which is able to adapt to the necessity of the requirements.

4.1 Preliminary Work

Among the different works done in this field, in order to obtain the objective prefixed in the thesis, different approaches were implemented and tested, comparing the obtained results and choosing the one that was the best compromise in terms of quality and time spent on the rebuilding process. More over is also interesting to highlight and explain the procedure for deciding which process was the best to be implemented. The tested systems were:

1. Convolutional Occupancy Network
2. PIFuHd
3. BSP-Net
4. AliceVision Meshroom

The implementation and testing were implemented on each of those systems on different class of objects. Starting from Convolutional Neural Network, this system was extremely similar to BSP-Net but it has substantial differences in the used algorithm and the required input from the system. Convolutional Neural Network required as input

a pointcloud file of the object which was extremely demanding, considering the idea of an application which was only capable to take a photo of the object and thus required an additional system capable to reconstruct the 3D pointcloud from a single-view image. The implementation of a system composed by two modules, one for converting image from pointcloud and one to convert pointcloud to meshes, was deemed wasteful compared to BSP-Net where the system already contains an algorithm to convert a single view image into meshes, without the necessity of an intermediate step. In terms of quality of the reconstructed mesh, it was possible to notice how the result of Convolutional Neural Networks were not on the level of BSP-Net, so the system was abandoned. The second system taken into account was PIFuHD, whose results had a quality of meshes extremely good and it was relatively fast in the computation, even more than BSP-Net. The main problem of this algorithm is that it was designed for the reconstruction of people and not objects, therefore, to match the requirements of our system, it was needed to change the neural network from people to objects and undergo to intensive training to perform again the operation of weighting of every neurons of the Network. This operation of training and testing was extremely heavy from a compu-

tational point of view and was normally performed with a computer with high quality components. Then, the time required to re-apply the weight undergoing training and the testing of the network on a normal computer, was extremely long. For those reason, the system was discarded. The third system to be tested was the Meshroom software of Alicevision. As mentioned before, it needed only photos of the object as input to works, and the provided results were extremely fine. Different types of objects were tested in order to examine the behaviour of the system but different problems have arisen. The system worked well only when a large amount of photos were fed in and the time required to apply the reconstruction was exponentially related to the number of photos and dimension of the object to reconstruct; besides, reducing the amount of photos fed in the system, the time and complexity reduced significantly, given a results which was extremely poor, incomplete and in the most cases also wrong. In the end, having as a requirement a system in real time for an application, was prohibitive to force the user to take thirty photos and wait for more or less one hour just for a single reconstruction, so, also this system was discarded. Finally the BSP-Net was tested: its results were acceptable and the time required for the entire reconstruction was less than

a minute, which was suitable for the application. The input required by the system was a single photo, which was the best possible option as a requirement. Hence it resulted as the best between them and it was decided to implement it in the project.

4.2 System Architecture

The Project has the final goal to produce an user friendly environment on the smartphone to allow the reconstruction of 3D model of real objects that the user can perceive in the environment in which it is in. The Application has the purpose to allow the user to take photos of different objects and visualize them in the application, with the possibility of manipulating them. We can split the overall system in 2 parts: the “Acquisition and Placement Side” which is substantially the application on the smartphone and acts as a client, and the “Reconstruction Side”, where the reconstruction program resides and acts as a server.

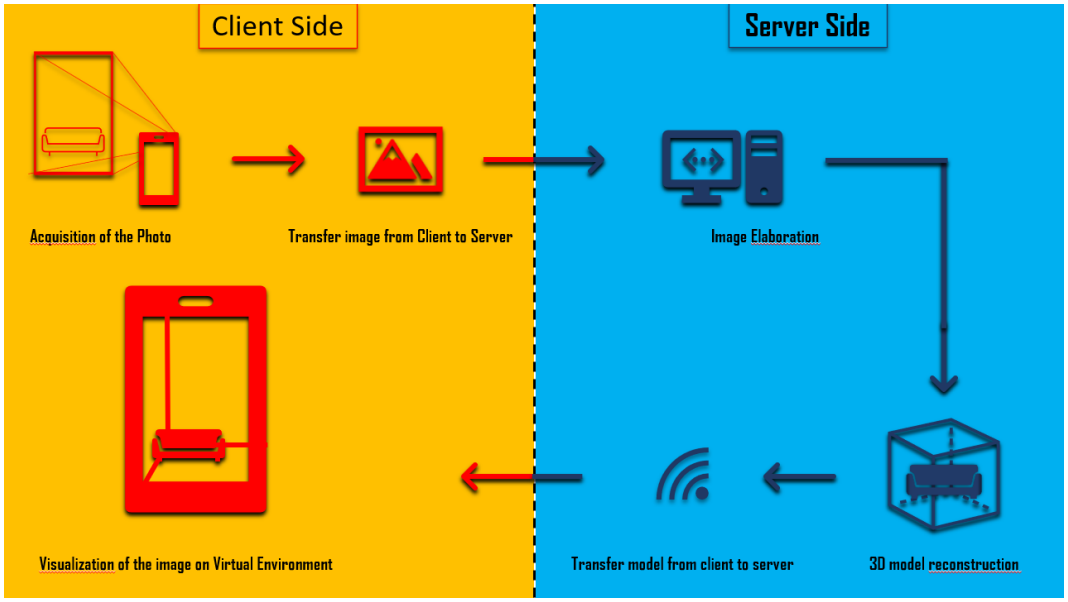

Figure 4.1: The general pipeline of the system.

4.3 Data Acquisition

Regarding the application, it is composed by three main windows, each one with a specific task. The first of them, defined as the main window or the photo window, has the intent to allow the user to take photos of the object of interest in the environment and place some indicators to designate its approximate position and also track the object already photographed, which is necessary when the number of objects to reconstruct is high. The second window, defined as the reconstruction or visualization window, is where the user can effectively

visualize and manipulate the 3D models reconstructed by the server. In this window it is possible to manipulate the virtual object in terms of scaling, rotation and translation, in order to match the real object pose as close as possible. The last window is defined as the Gallery and, as its name suggests, it allows the user to visualize and the photos they have captured.

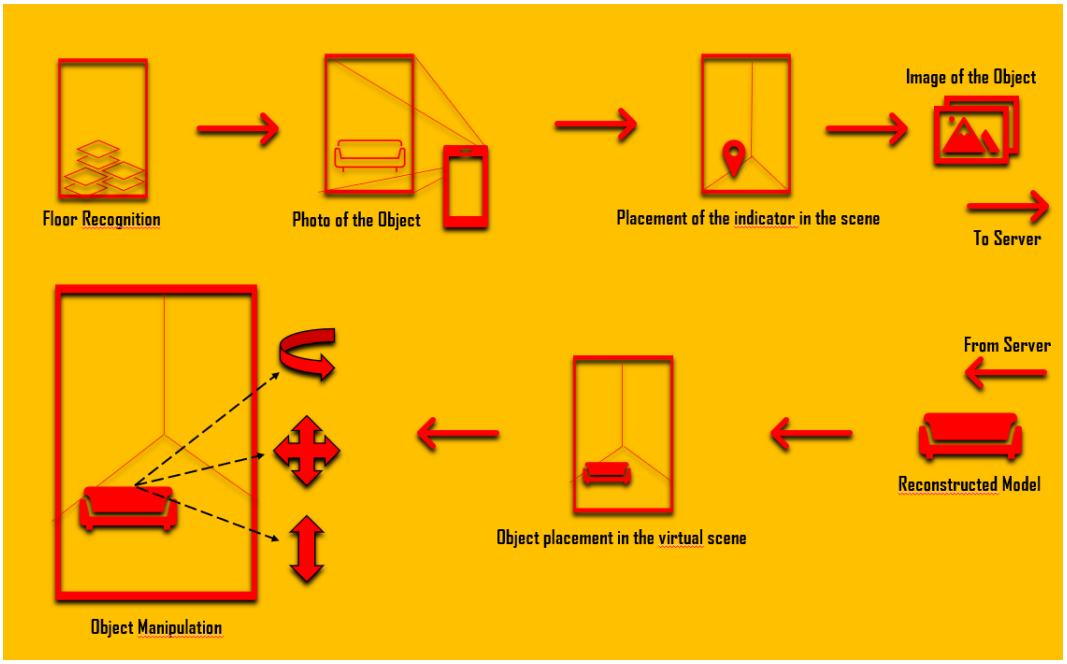

Figure 4.2: The scheme of the Client.

4.4 Reconstruction

The reconstruction of the model to be visualized in the application is built on a server to whom the client connects to. The server is composed by different parts connected in a chain, where the sequentiality is mandatory in order to accomplish its task. In fact, every block uses the input coming from the output of the block before in order to provide the proper output to the next block. The Server is constantly connected with the client and waits for information from it.

When those are received, first it decapsulates them in a legible way and then gives those to the first block of the pipeline. The blocks start by segmenting the photo in classes of objects, labelling them for better comprehension of them and painting them with a specific color. The color association is crucial for the successive part, The Mask Extraction, where only the object of interested is kept in the image and the remaining part of the scene is turned into white, resulting in a white background with the object in the foreground. After the mask extraction, the photo is then passed to the object reconstructor, where the program reconstructs the only object present in the photo in a 3D model, deducing its volume with the aid of a Binary Space

Partitioning. In this section of the pipeline, the mesh may have several imperfections, therefore the model undergo through an operation of remeshing to adjust and improve the overall quality of its shape. Finally, the model is ready to be sent to the server but, before the actual forwarding of the data, another step needs to be performed. The object, when is photographed, has a specific orientation in the space with respect to the camera, hence to instantiate the object with a correct orientation, two steps, called Rendering Generation and Rotation Estimation, are performed. The Rendering Generation consists in rendering multiple frames of the object with a slight angle of difference from each other. Those frames are stored and are used for the final step; the Rotation Estimation module searches the frame whose orientation is similar to the one in the photo and defines the angle of difference between the camera and the object. Finally, the last module called Object Rotation provides an alignment between the server coordinates and the client ones. As this step is completed, all the information can be encapsulated and sent through a socket to the client.

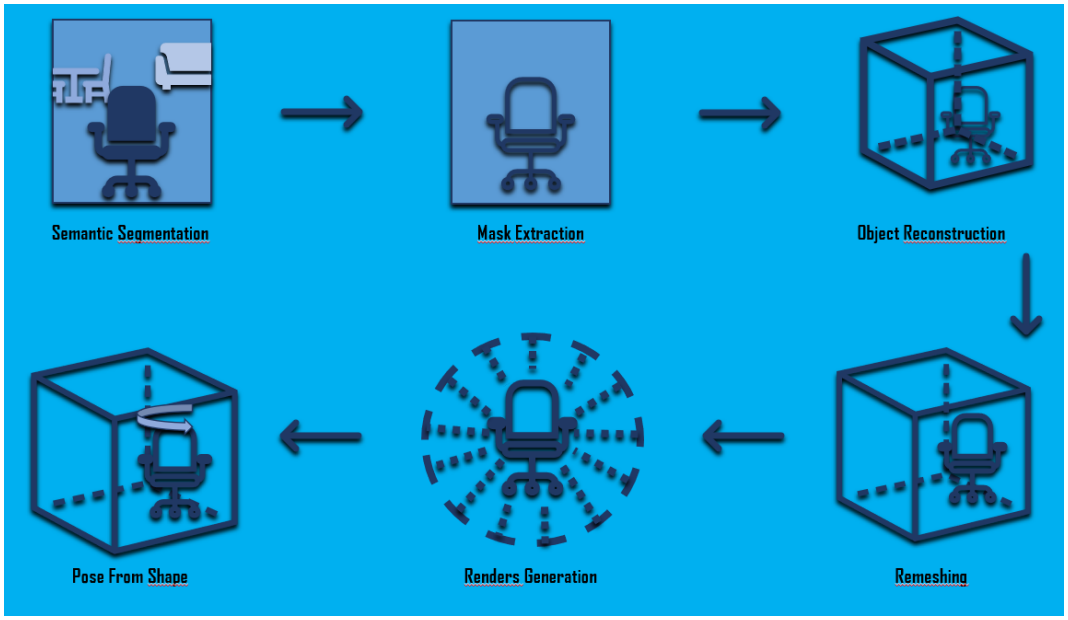

Figure 4.3: The scheme of the Server.

5 Acquisition and Placement Side

5.1 Organization and Composition

The main purpose of the client is to provide an accessible environment for the user to reconstruct and visualize a 3D environment where the 3D models in it are the result of a reconstruction from the photos that were taken during the session. The Application was developed using Google ARCore functionalites and the Unity environment. Google Arcore is a software development kit provided by Google for the creation of Agumented and Virtual reality applications. It is based on the usage of the track of the position of the telephone with respect to the world using six degrees of freedom; it is also capable to take over the dimension and position of flat objects like floor or tables and to estimate the light condition of a particular environment, adapting to it. Unity is a multi-platform graphic engine with the goal to allow the development of videogames and interactive contents such as applications or animations. The Program is based on C# language, using

an object-oriented approach and embedding some specific functions for the manipulation of objects and events inside the Unity environment. The project is mainly composed of a single script which defines the entire workflow of the application, and is supported by two other scripts: one for the connection, the TCPHandler, and one for the manipulation of the objects, the ObjectHandler.

5.2 The Lyfecycle

We can divide the lifecycle of the application in the workflow of each window.

5.2.1 The Photo Mode

The Main Window, or Photo Mode, is the window displayed when the application starts and has the main target to allow the user to take the photo of an object in the room in which it is situated, to visualize the gallery or switch to the reconstruction mode. A simple flowchart regarding the main operation of the Main window can be seen in fig.5.1

When the user starts, it is important as first step to allow the ap-

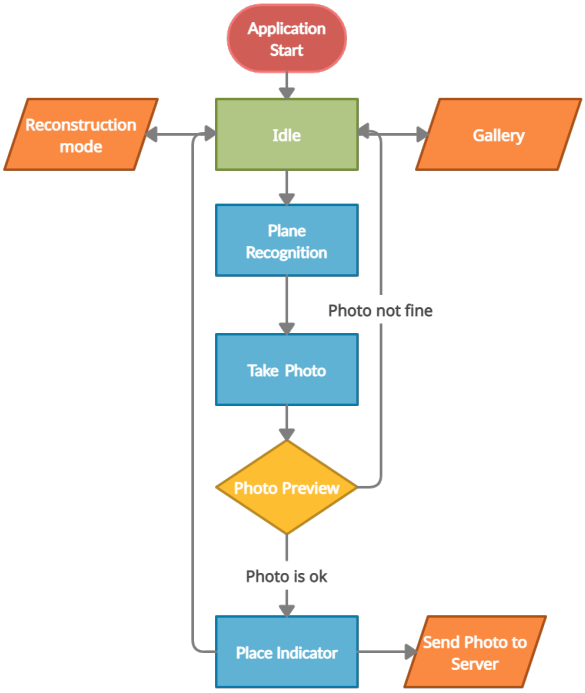

Figure 5.1: Flowchart of the Main window.

plication to recognize the floor or a flat surface, otherwise he or she would not be able to place the indicators or the object in the scene. When a photo is taken, the program shows a preview of it and if it is not in line with the standard of the user, it will remove the photo and return to the photo mode in order to take an other picture. If the photo is good enough according to the user, the program will automatically disable all the User Interface (UI) and the user will place an indicator in the scene in correspondence to the photographed object. In this way an anchor will spawn as placeholder and it will be stored to track

where the corresponding reconstructed model will be placed. After that, the program will encapsulate the index of the photo (which corresponds to the index of the anchor placed) and the photo itself and will send them to the server; finally it will return to the photo mode.

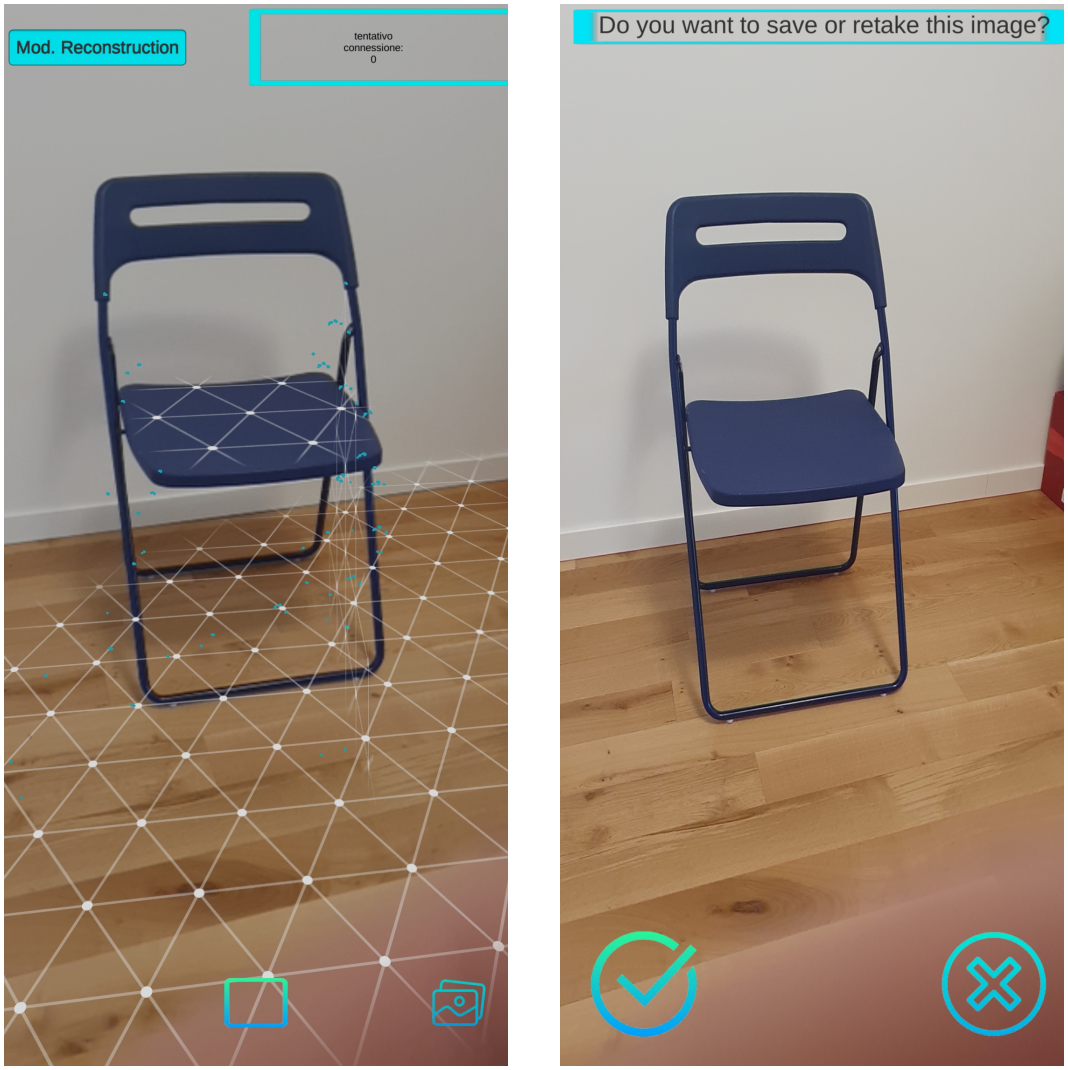

Figure 5.2: Showcase of the Main Window on the left and the preview of the taken photo on the right.

5.2.2 The Reconstruction Mode

The Reconstruction mode is where the user can visualize the 3D model reconstructed by the server and manipulate it by the switch from the

photo mode. The flowchart of the cycle of the reconstruction mode can be seen in fig.5.3

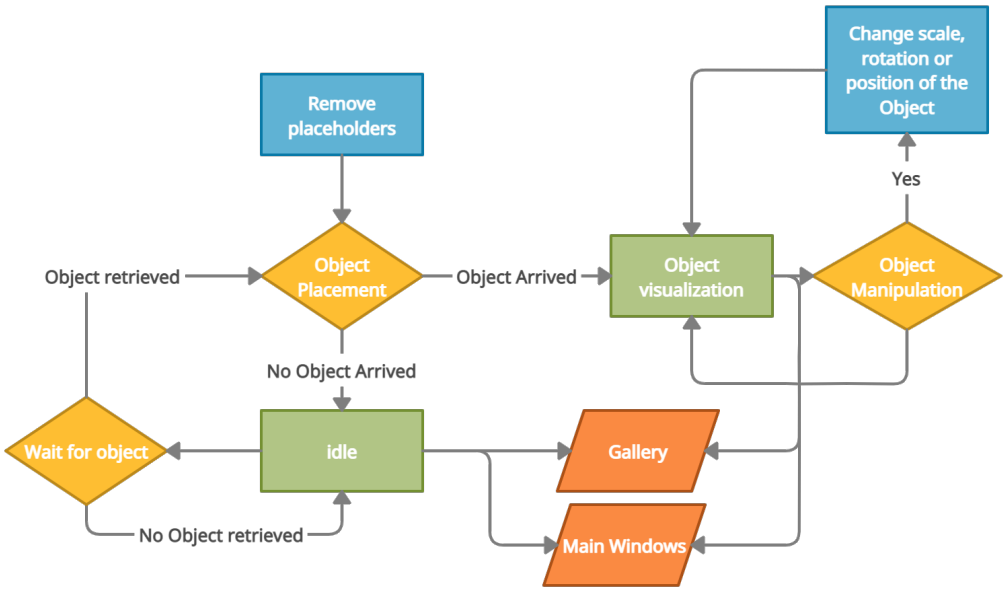

Figure 5.3: Flowchart of the reconstruction window.

When the user enters in reconstruction mode, it is assumed that he has already taken pictures of different objects in the room. If it has not, the reconstruction mode will show nothing and he can return to the photo mode or to the gallery (that, in this case, will be empty too). If the user had already taken some pictures and the server has elaborated them and returned the models, those will be shown in the scene in the exact place where the indicator was previously placed. At

this point he can tap on the object to pop-up the object handler panel.

This panel allows to rotate, scale or move the object.

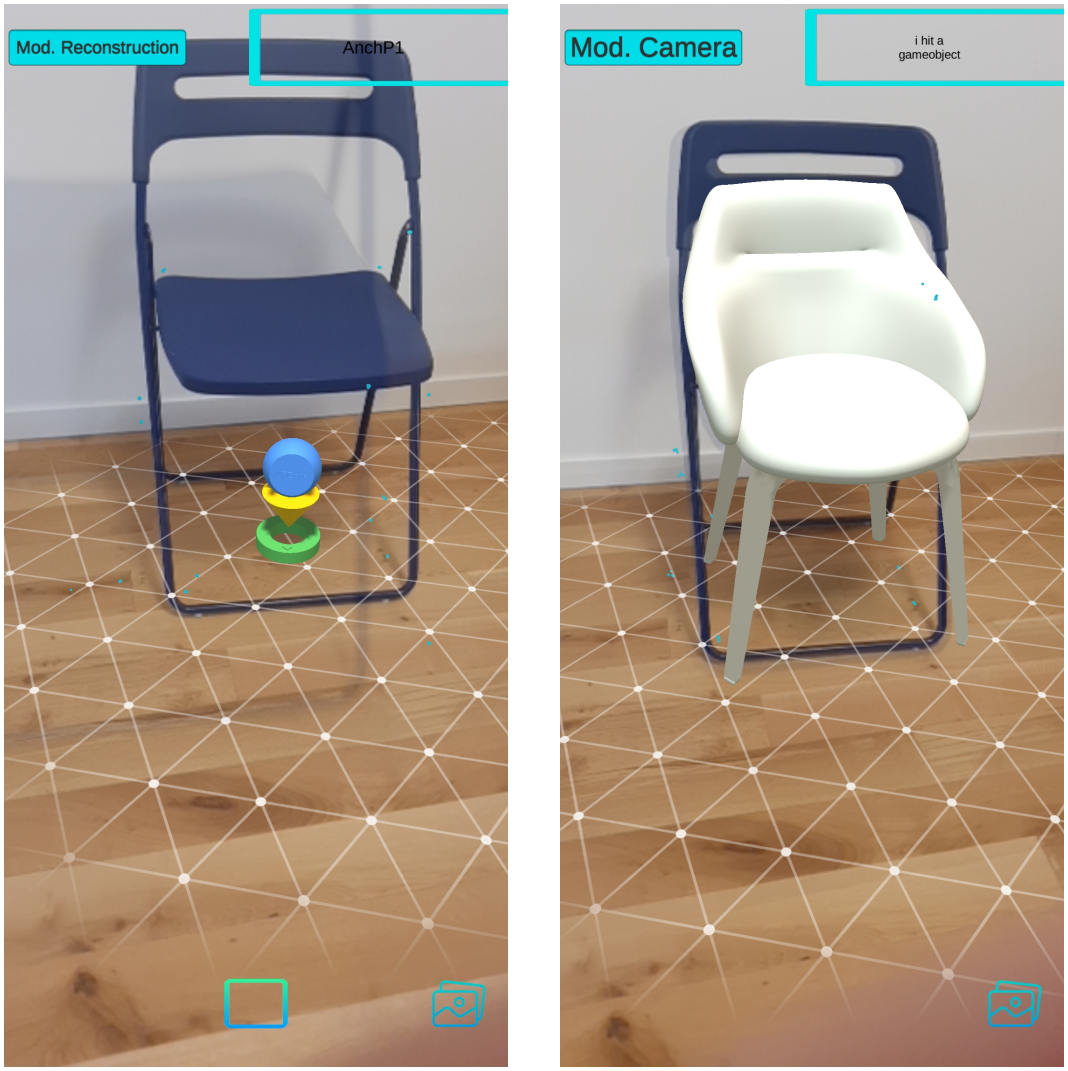

Figure 5.4: Placeholder placement on the Photo mode on the left and correspondent substitution with 3D model in the Reconstruction window on the right.

5.2.3 The Gallery

The Gallery has a simpler workflow than the other two, and its main purpose it to visualize the images taken to visualize all the objects that were already photographed. This window consists of three buttons, two to navigate the gallery, i.e., the back and forward button, and one to return to the main window.

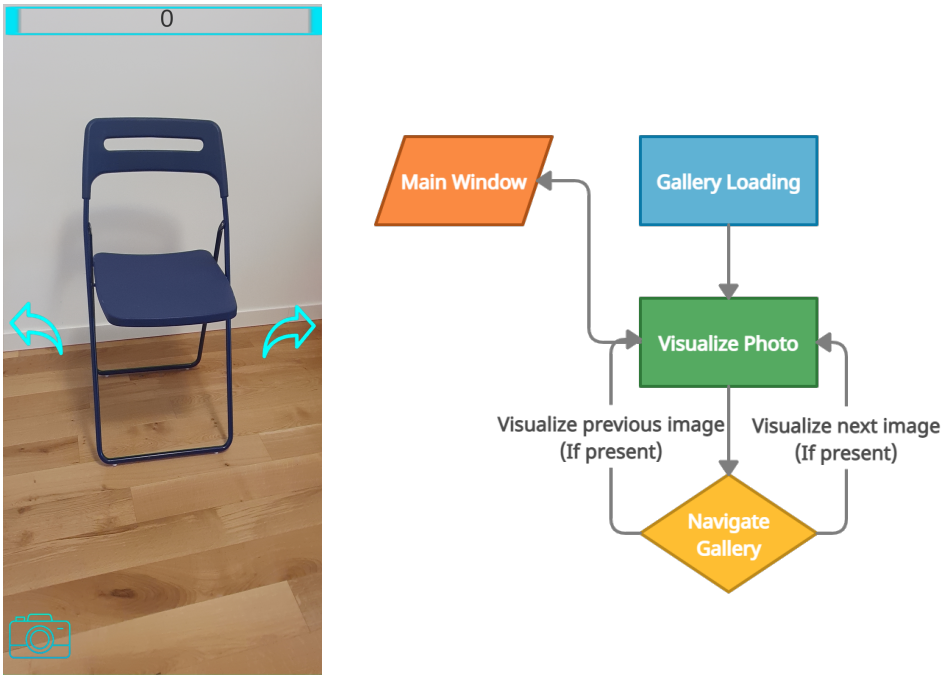

Figure 5.5: Showcase of the Gallery (on the left) and flowchart of the gallery (on the right).

5.3 Client Connection

The exchange of data between Client and Server is an important part of the application and is performed by sockets. The Client programmed in Unity is constrained to handle data in C#. The connection, the reception and all the operations performed in order to encapsulate and decapsulate data are handled by a script called TCPConnection, whose function is to establish a connection to the server, knowing a priori its IP and port number to create a socket for communication. The connection is established automatically when the application starts and remains in background until the user decides to close the program. The "send client" method has the task to provide two types of information to the server, which consists in the bytes of the photo and the index of it. Those information are stored and prepared asynchronously when the user, after taking a photo, decides that it can be used by agreeing in the "preview" panel. Then the algorithm will pass the information to the TCPHandler script, in order to encapsulate them in a JSON (Javascript Object Notation) string, which is a standard format for the data exchange between applications which need the support of a server. The data are stored in an static structure

to allow the correct read and then converted into a string; eventually a stream is opened and the data are actually sent. After the server receives the data and processes them, it returns another JSON string which contains various data type. Those data are decapsulated and decrypted by the client who knows a priori their structure and organization. The data are stored in another structure that is passed to the main script to perform the operation of object construction and instantiation of the model in the scene. The received data is composed by three types of information: the first one is the GameObject model which is basically the model destructured in its basic components, an index that defines the position in the space in which will have to be instantiated and a rotation value that represents the difference of the degree of the object with respect to the camera when the photo was taken. The purpose of this value is to instantiate the model with the same rotation as the real object in the scene.

6 Reconstruction Side

The reconstruction side is based on a server which runs on a Windows PC and manages the reconstruction of the object. It is a Python server which establishes a connection with the smartphone which acts as the client.

The server receives from the client a JSON string containing the color picture of the object that is intended to reconstruct, then it saves it in the proper folder. A second process performs the actual reconstruction task, which can last for about 50 seconds (usually less than 1 minute) when running on a Nvidia GeForce RTX 2070 GPU, and it returns a 3D model which will be placed in the physical space.

6.0.1 Server Connection

The server is a standard Python server which establishes a local connection over a specified port with the client, which is an Android application created in the Unity Engine and written in the C# program-

ming language.

The entire process starts when it accepts the connection request from the client, which starts a local loop where first it controls if the connection is still valid, waits for a reconnection if not available, or starts listening for incoming data otherwise.

As the connection is defined by a TCP socket, the messages received by the client are retrieved as a single stream of multiple packets, thus, at every packet, an operation of ASCII decoding is performed and the contents are concatenated in a string.

At every cycle a function has the task to read the string in order to control if the message is fully received. Considering known a priori the structure of the entire message and knowing that the JSON message is bounded by special characters "{ }", we can check them in order to understand if the message is completely received, without any further check since it is all based on local connection.

The information contained in the JSON string are extracted and stored separately: the index of the image is locally stored to be sent back with the model and the image is decoded from Base64 and stored in the folder "0 Input Photo", which corresponds to the starting point of the reconstruction pipeline.

The server then performs a polling operation every 100 milliseconds checking if the model exists in the last folder of the pipeline that corresponds to “7 Object Rotation” containing the final model. If the object is found, the server represents it as a string and, separately, retrieves the azimuth value from the folder “6 Rotation Estimation”.

Finally the server defines a JSON message containing the object file, the name and index of the object and the azimuth value; this message is then encoded in ASCII, sent to the application and finally the connection can be closed, ready to start a new loop iteration.

6.1 Reconstruction Pipeline

In order to solve this complex problem it has been decided to decompose this main task into many subtasks, each one of them implemented as an independent module. All modules can work and be tested independently from one another and they compose a pipeline where the output of the one module will be used as the input of another module. Each intermediate output is saved in a specific folder on the server storage memory disk, thus it is possible to easily check intermediate results for all the steps of the pipeline.

According to the complexity of each subtask, it has been decided to perform a balanced subdivision into 7 main modules:

1. Semantic Segmentation
2. Mask Extraction
3. Object Reconstruction
4. Remeshing
5. Rendering Generation (for rotation estimation)
6. Rotation Estimation
7. Object Rotation

The basic functioning of the system can be seen in the fig.6.1

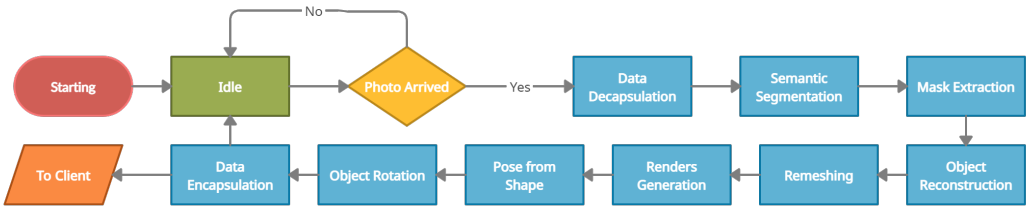

Figure 6.1: The flowchart of the pipeline of the server.

6.1.1 Semantic Segmentation

"Mseg" is one of the best software for semantic segmentation which provides great results if applied on videos or simply on images (as in our case). It is substantially a model which works on a composite dataset, which means that merges many datasets from different domains. The strength of Mseg is that it can take benefits from each of the main different semantic databases solving the incompatibility issues and providing robustness through multiple domains.

The work was based on flatting the inconsistencies and re-lable all the annotations that were incompatible with a process of large-scale annotation, leading the final outcome to really high accuracy. The unified taxonomy has been reached through the alternation of splitting and merging of the classes according to the possible clashes.

Moreover the training has been made really robust thanks to the adoption of "zero-shot cross-dataset transfer": in this way the performance that we can expect for a model in the real world is very high even in a new environment and without having target domain's data during the training phase. The condensed taxonomy resulted in 194 classes organized in a flat manner in order to maximize the compati-

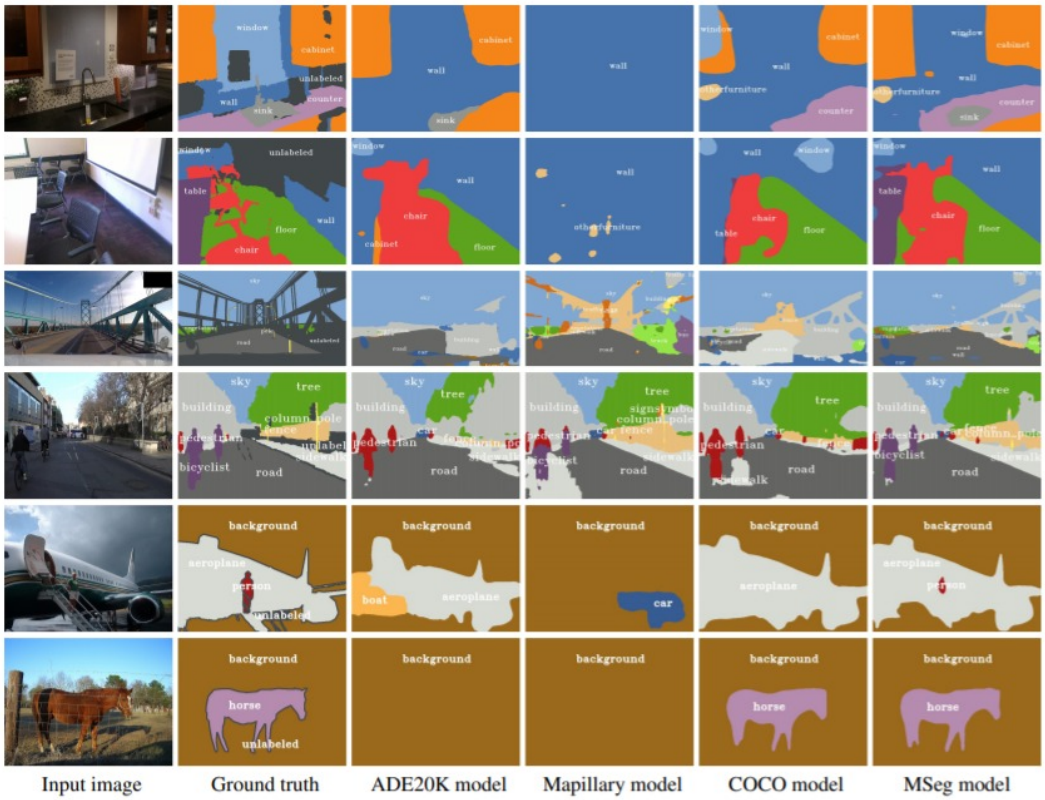

Figure 6.2: Visual representation of the segmentation results of *MSeg* with respect to other segmentation models.

bility with all the trainings which represent the standard method.

For what concerns its application inside the server side, the semantic segmentation is the first module of the pipeline; it is based on Mseg code from GitHub. It has been adapted in order to work on Windows Systems, since original repository was intended to work with Linux-based systems only.

It receives a picture as input, in our case the photograph of the

object that we intend to reconstruct, such as a chair or a table, in PNG format. The features of the picture are extrapolated and computed through a series of convolutional layers of the neural network and with the exploitation of trained weights the network is able to understand which objects appear in the frame and assign a label to each one of them.

In this way the neural network detects chairs, tables, monitors, floor, walls and so on, and for each class of object it assigns a different color, building up a colored mask which fits with the pixels of the specific object.

According to the original version of Mseg, each different object was identified with a different rgb-scale color with a transparent level that permits to recognize the real background object; also a label was assigned to each one of the object, floor and ceiling of the room, since it tries to semantically recognize each character inside the scene.

As far as it has to be adapted to what was needed for this specific application, the output has been modified in order to remove those labels, since the mask extraction (i.e., the next module of the pipeline) is based only on the centered object of the scene and ignores all the other characters; according to this principle there is no need to classify

Figure 6.3: The original photo (left) and its grayscale segmentation produced by the modified *MSeg* (right).

the objects of the scene and therefore the classification is redundant and has been removed.

Also it has been assigned a gray-scale color to each one of the object and the transparent level has been removed since all the crops operations will be demanded to next pipeline modules.

So, in order to ease the following steps of the pipeline, the Mseg

output has been modified to produce a gray-scale image where each different object is colored with a different shade of gray.

6.1.2 Mask Extraction

Now that the objects inside the picture have been detected and that a corresponding color has been assigned to each of them, the one which is centered in the frame can be extracted. With a sampling mechanism the centered point and its neighborhood can be easily found.

The first step is detecting the half-width half-height point, from which the gray-scale value of the pixel can be extracted; the color range will be in a [0-255] scale where 0 stands for black and 255 stands for white. This will be considered as a subset of pixels belonging to the current object that we want to reconstruct. Then a flood-fill algorithm is applied: starting from the selected centered point the algorithm stores it in a new file called "Binary Mask" and marks it as "read"; then it visits recursively the pixel's neighborhood and checks which ones have the same gray value of the first one.

For each pixel which shares the same color of the first one, the algorithm saves it on the "Binary Mask" file and marks them as read;

then the algorithm is applied recursively the the marked pixels which repeat the process without taking into account the already marked pixels.

At the end of the process we obtain a brand new file which contains a set of gray pixels on white background and which consists in the binary mask of the object that we want to extract.

At this point the object can be easily retrieved by multiplying the original image with the binary mask through a "bitwise-and" operation which preserves only the main objects and deletes the background. What is obtained is an inverted-color image that has to be "bitwise-not" processed in order to retrieve a clear view of the extracted object.

Now another pre-process operation has to be performed in order to feed the Object Reconstruction module with the proper input in an optimal shape: in fact it has been tested that the object reconstruction leads to better results if the raw input is served as a squared image.

To do so the image containing the extracted object has been cropped in order to delete white useless borders and keeping only what is really useful, i.e. the gray pixels of the object; then according the the largest size among width and height the picture has been enlarged in

order to obtain a squared shape. At the end of this module what is obtained is the extracted object placed in a white background picture.

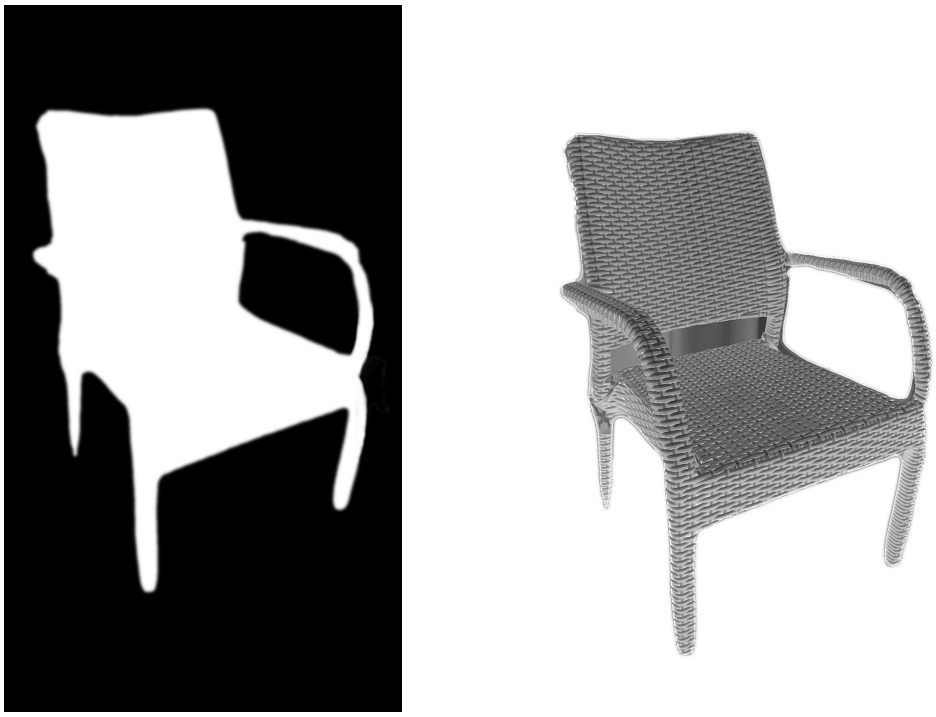

Figure 6.4: The binary mask of a chair (left) and the result obtained by applying the mask to the original photo (right).

6.1.3 Object Reconstruction

The current step is the most important for the goal of the project. In fact the state-of-the-art papers have been studied in order to select the best approach for 3D object reconstruction starting from a snapshot.

After many tests the BSP Network (Binary Space Partitioning)

resulted as the best one in terms of computational time and quality of the mesh produced.

One interesting aspect is that despite the model learns how to represent 3D shape decomposition, still BSP-Net is unsupervised since it does not need any convex shape decompositions during training phase.

Despite object reconstruction is based on convex shape decomposition, this operation is not needed during the training phase, therefore BSP-Net can be considered as unsupervised regarding during decomposition step.

In facts it can autonomously retrieve convexes from a structure of planes on which it builds a BSP-tree.

The strength of BSP-Net is that it generates compact meshes which are "watertight" and are easy to parameterize without having to perform iso-surfacing which is usually quite expensive; this is a perfect way to represent all kinds of geometries which are composed by a sharp configuration. When we talk about watertight meshes we are referring to the fact that the model consists in a closed surface without holes and it is well defined inside of it.

Moreover thanks to the recursive approach based on the space

subdivision it uses very few primitives and this allows to BSP-Net to achieve competitive results. This approach leads to a better manipulation and control over meshes achieving a compact and really high visual quality. The generative neural network has been developed in order to predict multiple planes which could replicate the surfaces of the 3D model associating each facet to a binary space partitioning, which gives the name to the whole module; the shape is finally obtained by combining all the partitions.

The baseline of the algorithm is, giving as input a shape feature vector and n point coordinates, to exploit an implicit function to decide if these points are outside or inside the shape.

This is achieved through 3 steps: the first one is based on the collection of a certain amount of planes represented by an equation which produces respectively a binary division of the space; then these partitions are gathered in order to build many convex shape primitives; as final step the output object is obtained by merging all these collections.

This leads to a speed up per mesh during inference and without having to build iso-surfacing which is typically expensive. Moreover the generated meshes are compact and watertight also with sharp fea-

tures.

For what concerns the training of the network, no convex shape decomposition is needed since it is self-supervised and exploits the set of convexes constructed in the second layer of the network. As said, the network is divided in 3 main modules which work on an encoder which extracts feature vectors according to the input data. The first module extracts hyperplanes from the row data; the second layer groups these hyperplanes in order to re-create convexes parts, and the last one builds these parts together in order to reconstruct the final shape of the object.

The training is based on classical optimization problems where integer problems are handled by relaxations; in fact it is divided into 2 steps in order to better approximate the loss: the first one works on weights represented as continuous units, then in the second phase the networks works with discretization and quantizes the previous weights creating a union which generates accurate results through fine-tuning. In this way the model is able to reconstruct much finer objects.

In order to understand how to manage input data and properly feed the network, it was important to look at the dataset that has been used

to train the model. It is a synthetic 2D dataset of images with a lot of different categories (e.g., chairs, tables, plains, cars, ...) where each image is a 128x128 pixels picture with a small object in the middle in a white background. The images of the dataset are in low resolution and probably this choice has been taken due to the expensive costs in terms of computational time needed to properly train the network.

Hence these features has been taken into account when managing raw data to feed the network: in fact in the previous step of the pipeline the image extracted from the "mask extraction" module has been cropped and converted into a square shape picture. With this simple trick the quality of the output 3D model has been drastically increased.

The input picture used to feed the BSP-Net model has certain constraints that have to be considered in order to retrieve an acceptable result.

What it needs is a picture with high quality and taken close to the object; it is also preferable if the object is set in the middle of the photograph, since in this way the algorithm can compute a higher number of features achieving better results.

The model used in this work has been trained by the authors of

Figure 6.5: The reconstructed object using BSP-net.

the original paper. The training of BSP-Net has been performed by setting 1,000 epochs with batch size 64 and using an Nvidia GeForce RTX 2080 Ti GPU, for about 5 days of training. The dataset used for training was a synthetic dataset where the images fed to the neural network were generated by rendering the CAD models from a subset of ShapeNet.

What it has to be considered during BSP-Net reconstruction is that it is based on a single view reconstruction approach. This means that it needs only one snapshot of the object that we are interested in

reconstructing and without further information the algorithm itself is able to extracts all the features and details that it needs in order to provide a fine reconstruction. The single view approach permits to lighten the computational cost of the algorithm since only one picture is processed by the neural network, therefore BSP-Net returns an appreciable output in reasonable time.

Despite the actual reconstruction step represents the bottleneck in the server pipeline, it has to be considered that time needed to reconstruct the object is always way less than 1 minute and still can be improved with more powerful hardware resources.

BSP-Net provides one of the best reconstruction quality with a high-level segmentation accuracy; it is also the first deep generative network that outputs watertight and compact polygonal meshes in one shot without be based on predefined structure and topology. Both shape correspondence and object segmentation can be inferred by the built BSP-tree which guarantees flexibility. Moreover, it is the first algorithm which reconstructs a segmented 3D shape starting from a single unstructured image achieving fine results. It can also recover and reconstruct the features which are presented as a sharp geometric shape.

Despite in some cases the output mesh is messy due to the fact that the input image is too complex, in most cases the output mesh is acceptable and the object is successfully reconstructed.

The output mesh is entirely represented by an array containing faces and an array containing vertices and is saved as a PLY format, then properly converted in OBJ format for simpler management in the next module.

6.1.4 Remeshing

Now that the 3D object is retrieved it is clear that it is still a raw model with some imperfections. Hence it is important to fix some voids or intersections among some planes of the mesh.

In order to deal with a finer mesh the best solution is to apply some remeshing algorithm over the 3D model inside a Blender 2.83 environment, which is a simple and efficient way to obtain better results.

This task has been reached by working with Python Blender command line in order to build BLEND format scripts that can refine the meshes through automatic operations.

The approach is basically based on selecting all the MESH objects

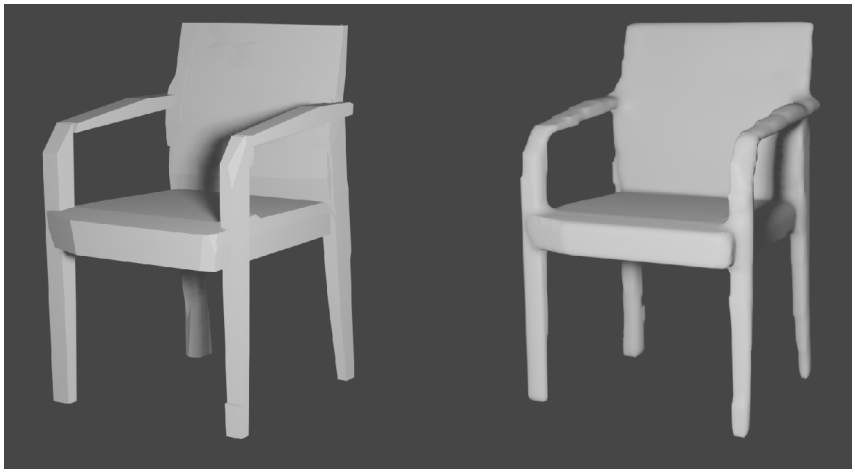

Figure 6.6: The difference between an object before (on the left) and after (on the right) the remeshing operation.

inside the scene (just one in this case) and applying a modifier property from the modifier tools provided by Blender environment to the current object.

This is a simple way to access to all the edit properties that permit to achieve good results through voxel manipulation.

In details, a “smooth” operation is applied over vertices and a remesh algorithm is performed over the faces of the object, directly in Blender environment.

As far as these parameters depend on the computational power of the resource on which the server is running on, they could be increased in order to lead to better defined meshes with a better quality

in terms of surfaces and edges, with a little drawback regarding computational cost.

6.1.5 Rendering Generation

The final step is to reflect the original position and rotation of the object in the physical space.

To do this it has been decided to use a Pose From Shape algorithm, which has been opportunely splitted into 2 different modules in order to guarantee a simple management of intermediate results and an appropriate flexibility during the testing for the search of the best parameters to set in this part.

As baseline the approach tries to understand how the model is oriented with respect to the camera taking into account a set of samples which consists in a collection of images provided by this "Rendering Generation" module, and performing a spatial and features compare once per time; this is carried on through deep learning algorithm.

The first block consists in the generation of an appropriate number of renders of the 3D model. The model is uploaded in Blender and set in a space surrounded by a fixed source of light and a camera

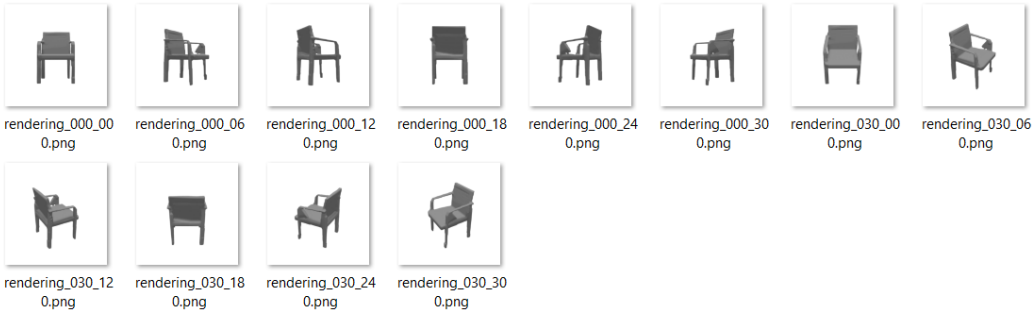

Figure 6.7: Rendering generation obtained with Python Blender script executed by command line.

which rotates around the object in the middle and takes a snapshot at every step of the iteration. At the end of this first phase the camera fully rotated around the vertical axes and parallel to the ground of the Blender world. Then a fully rotation around the model has been repeated but the snap is taken with a slight slope of the camera, in order to render images from a different point of view.

According to many performed tests it has been decided how to set the iteration step and the degree of the slope of the camera in order to optimize the render generation, since this procedure is expensive in terms of computational time and could lead to a bottleneck.

After the rendering of 12 images from different points of view they are stored into a folder and then passed to the next module which compare them with the current 3D object in order to estimate the rota-

tion of the model with respect to the camera, through a deep learning approach.

6.1.6 Rotation Estimation

The Rotation Estimation module is based on “Pose From Shape” code that can be found on Github; this version has been adapted according to the goal of the project to work with the server pipeline by changing the number of pictures that were needed as input in order to estimate the correct rotation of the 3D object with respect to the 2D image.

It takes the folder with the renders of the previous step and the original photograph as input.

The idea behind Pose From Shape algorithm is to guess which was the original rotation of the 3D model according to what can be detected by the picture.

It tries with combinations of the obtained renders and select the one which best fits; it also returns 3 parameters: azimuth, elevation and rol which are useful to build the analytic formula that will be applied over the 3D model in order to relocate it in the desired position with the appropriate rotation and orientation.

The advantage coming from PoseFromShape is that the network does not need to be trained over specific categories with a predefined pose as most of the pose estimation method do, but it is based on a "generic deep pose estimation approach". In this way the module can effectively deal with new objects and always provide accurate results.

The module is based on a Convolutional Neural Network which is fed with a 3D object and an image, and returns the relative pose of the model, achieving great results for supervised categories with respect to the state of the art.

Moreover, the network which is trained over "everyday man-made objects from ShapeNet" is able to generalize with new kind of 3D models without needing further training. In fact, a great feature of PoseFromShape is its great results for novel objects which can be totally different from the ones used to train the network, overcoming category-specific models.

Hence, it is approximable to a category-agnostic approach, but with respect to instance-specific methods since it considers as input the 3D object of interest, stressing out the attention for a single 3D instance which provides better defined details rather than the entire object categories.

The use of a 3D model also provides information to the network that also increase the performances over known categories, even if it is only approximate.

With respect to previous models, PoseFromShape introduces a brand new "category-free viewpoint estimation" which can predict object pose only taking into account its 3D model, even if completely different from the ones used for training phase; the pose supervision is simplified; the performance are boosted if the pose is applied to object of known categories.

At the base of PoseFromShape there are two encoders which extract features from the RGB input picture and the 3D object, then exploit a "classification-and-regression approach" in order to understand the orientation, according to a certain probability, and producing 3 values as output: azimuth, elevation and rotation.

In this case, it has been combined information from both shape and image, achieving great results in terms of accuracy independently from the training data and object category, and without any issues with respect to differences between real and synthetic images used for the testing phase, confirming a completely generic approach.

6.1.7 Object Rotation

As final steps the remeshed object is rotated by 90 degrees in order to map the axis coordinates from the Blender environment to the Unity one. This is in important step since it will provide the appropriate presentation of the model to the user.

Now that the object world has been aligned and the rotation has been perform the 3D model it can be sent from the "Reconstruction Side" to the "Acquisition and Placement Side" which finally accomplish to the task to place it in the physical environment and move towards the next object that the users intends to reconstruct, starting the pipeline in an iterative way until the whole scene is reconstructed.

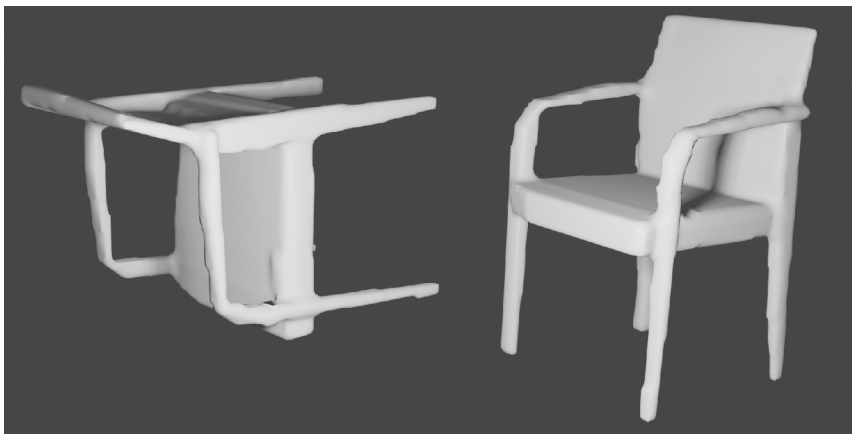

Figure 6.8: The model before (left) and after (right) the rotation of 90 degrees on its x axis.

7 Testing and Results

Having completed and implemented all the necessary steps of the project, it was necessary to run several tests. As stated in the BSP-Net description, the network was mainly trained on five object categories, which are Chairs, Tables, Lamps, Cars and Planes. Since the system was designed to work with indoor environments, the last two categories were left out from the tests. The entire reconstruction process is completed in less than a minute, and the overall procedure from the reception of the image, to the acquisition of the complete structure containing the object takes around one minute and twenty seconds. This period is an average value between the different measurements and depends on the complexity of the object to reconstruct, varying from a simple square table, to a modern chair with multiple parts.

Figure 7.1: The reconstruction of different models belonging to the three categories mentioned above.

The entire reconstruction system was investigated in each passage in order to define the characteristics that could alter the structure of an object during its modelization.

Having the pipeline composed by seven well-defined steps, it was possible to circumscribe the three steps that could be responsible for the quality of the outcomes: the semantic segmentation, the mask extraction and the BSP-Net. However we can avoid to consider the mask extraction problem as a point of failure of the pipeline since the only possible error related to it occurs when the object to be reconstructed is not centered inside the picture, i.e., leading to bad behaviour of the algorithm (e.g., it could consider a wall or the floor as centered point) as in Fig.7.3.

These problems lead to the creation of a wrong 3D model; therefore it is interesting to point out the results produced by the different errors that can affect the pipeline in order to understand how to fix and avoid them.

Figure 7.2: Other reconstructions of different models. The objects are organized in two columns, each one containing the original photo, the mask of the object and its model after the remeshing operation.

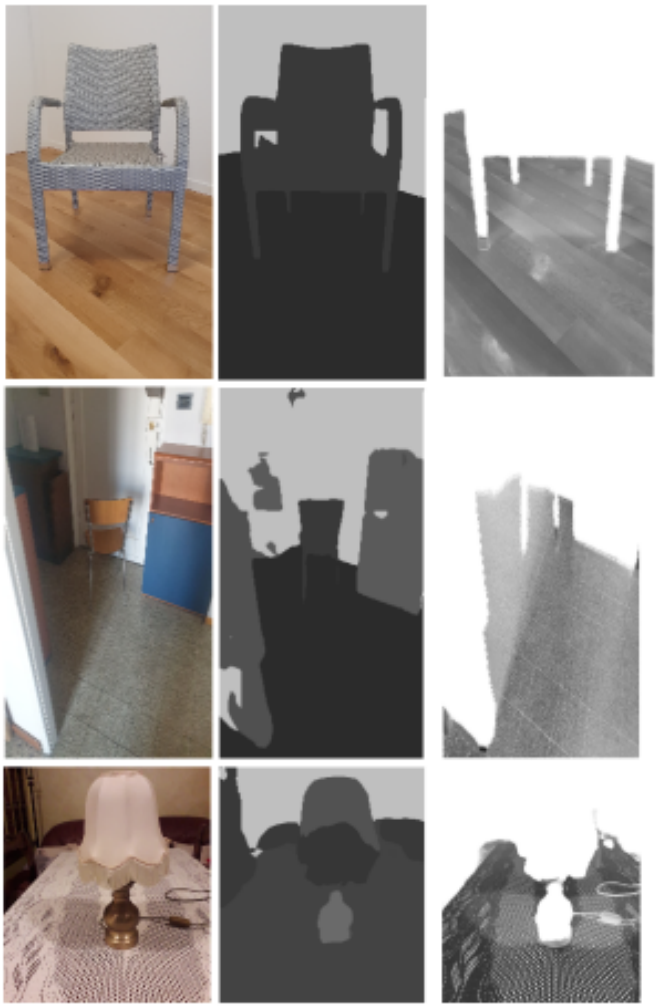

Figure 7.3: Extraction errors due to the fact that the object was not perfectly centered inside the picture.

The first step to be taken into account is the semantic segmentation, which is the starting point of the process and partitions the entire photo, defining its components with a specific level of gray. This pas-

sage is crucial since the reconstruction relies entirely on the object shape defined in this step. The initial limit of this network is determined by the dimensions of the input image, since it is constrained by the GPU installed on the PC. The test showed how, using a GPU NVIDIA RTX 2070, MSeg was capable of handling photos with a maximum size of 1080x1920, and any larger size resulted in the saturation of the memory. There are several possible errors caused by the segmentation step which are carried through the pipeline and can cause a wrong reconstruction, e.g., the missed detection of a hole between the different parts of the object, or when the segmented photo presents holes which are not present in the real object. Another possible mistake is the aggregation of different objects within the main one, entirely transforming the structure of the model. The example of those errors are shown in Fig.7.4

Figure 7.4: Different types of Mseg errors related to erroneous recognition of the holes or caused by wrong definition of the outlines of the object.

The other errors that can possibly arise during the reconstruction phase are caused by the core of the pipeline, which is BSP-Net. This part was tested using a great amount of different objects and orientations, taking also into account the perfect distance that allows an

acceptable reconstruction. The network was trained using pictures of synthetic models (i.e., rendering of digitalized 3D models, not real objects) with low resolution (128x128), probably due to the fact that the training phase was quite expensive in terms of computational time and small images can lead to comparable results saving a huge amount of time. It was decided to resize the image to have a square photo adding a little bit of padding (white space around the element) around the object in order to match the condition in which the network has been trained.

Figure 7.5: Differences in reconstruction quality when the image has raw dimensions (left) and when it is resized, properly padded and cropped to 128x128 (right).

As a matter of fact, BSP-NET builds the model starting from a two-axis point of view and tries to extract the third dimension from it, thus the capability of infering the missing dimension is greatly affected by the orientation of the object in the photo. After a couple of tests was clear that the best reconstructions occurred when the photographed object was rotated of an angle varying from 30°to 45°respect its y-axis, as shown in Fig.7.6.

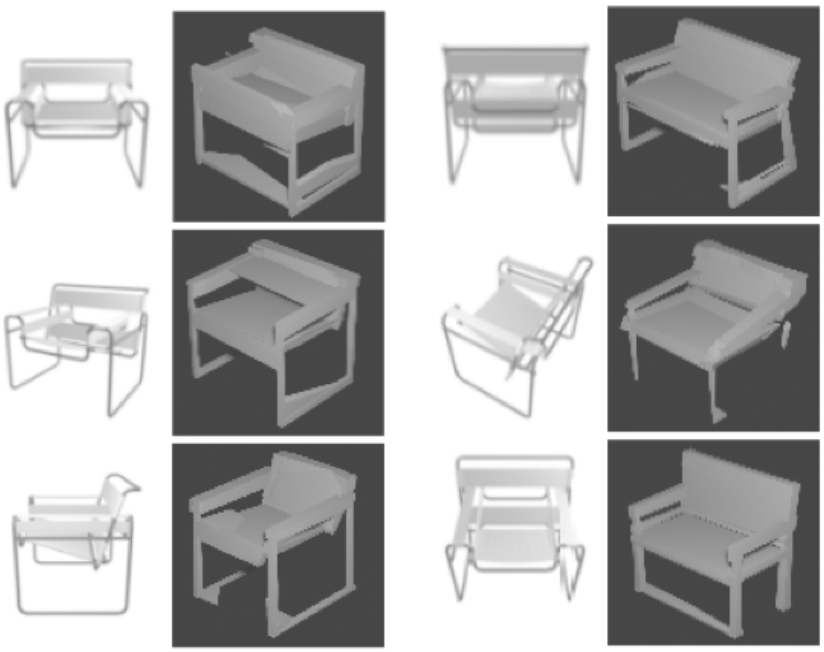

Figure 7.6: Reconstructions of the same object starting from different photo angulations.

Another aspect of the network is the fact that, having been trained us-

ing “perfect models”, the differences in luminosity given by the lights and shadows of the real environment can alter the object and its reconstruction, producing models which are less accurate with respect to the synthetic one. This problem causes the disruption of the mesh, sometimes adding pieces which are not present in the photo or with a slightly different structure.

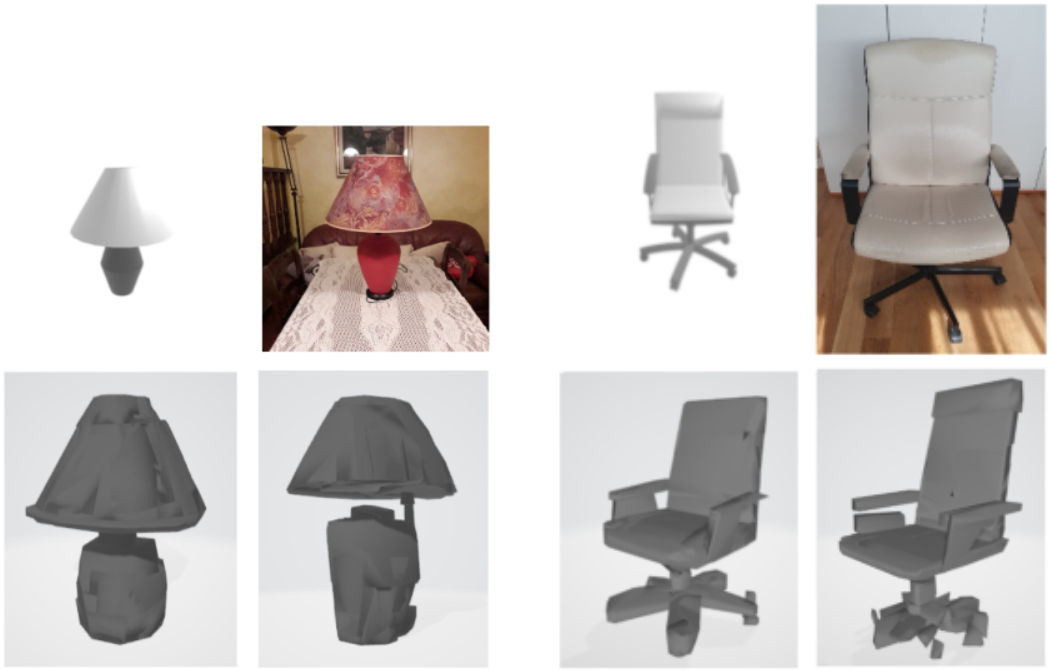

Figure 7.7: Comparison between two similar objects reconstructed using photos of synthetic and real objects. We can notice that some details are missing, incomplete or defined in a wrong manner.

The next issue detected concerning this network is caused by the wrong perception of the object dimension, since the actual dimen-

sion of the object is not fully deductible in the photo and sometimes not respected. This phenomenon misleads the network which modifies the object reconstruction by lowering or increasing its dimension or even changing the actual type of the object, relying on the synthetic shapes used for its training.

Figure 7.8: On the top: reconstruction error caused by wrong perception that changes the type of the object perceived by the server system (from chair to armchair). On the bottom: reconstruction error caused by erroneous perception of the object dimension (the back of the chair is perceived as stretched and scaled by the server system).

As completion of the overview we also present cases in which the BSP-Net is not able to reconstruct the model from the snapshot due to the lack of information or because of the higher complexity of the object shape. This can be considered as a drawback for the network which is based on the information stored during the training phase, showing that the system is slightly biased and sometimes constrained

to the models on which it has been trained on, without being able to generalize.

Figure 7.9: Cases when the object is wrongly reconstructed.

Another interesting step of the pipeline that has been taken into account is the remeshing, which consists in an operation of smoothing applied to the original mesh in order to produce clearer results. Although this procedure is not capable of altering considerably the original mesh or clean it from superfluous part, it can happen that the mesh

is slightly different, neglecting inconsistent parts. The operation could also lead to the formation of holes not present in the original one, resulting in a degradation of the overall outcome, although in almost all of the cases the final outcome is an appreciable improvement.

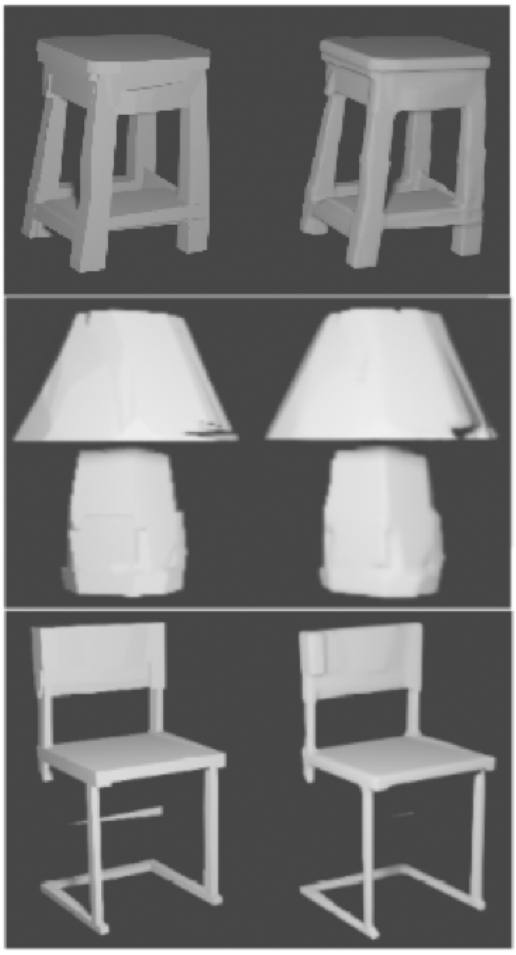

Figure 7.10: Comparison between the original mesh of the model (on the left) and the object mesh after the remeshing (on the right).

Finally, it was taken into account how the application works and responds considering the possible causes of failure. The overall flow of the application is fluid, at the start the connection is established almost immediately and it does not affect the camera frame rate that is fixed to 60 fps. Google ARCore has the task to manage the camera and at the same time performs the detection of the floor without any fail. The first problem arises when the application is active for several minutes, considering that, during this time, the floor detection is constantly active and continues to search for a plane; this can lead to an overpopulation of vertical and horizontal planes that sometimes does not exist. The large number of floors constitutes an impediment to the utilization of the app because, considering that the indicator of the object must be placed on these AR floors, it is possible to place it in a wrong spot that is directly above or below the point of interest, or also preventing the model to be moved due to the collision of the planes around him.

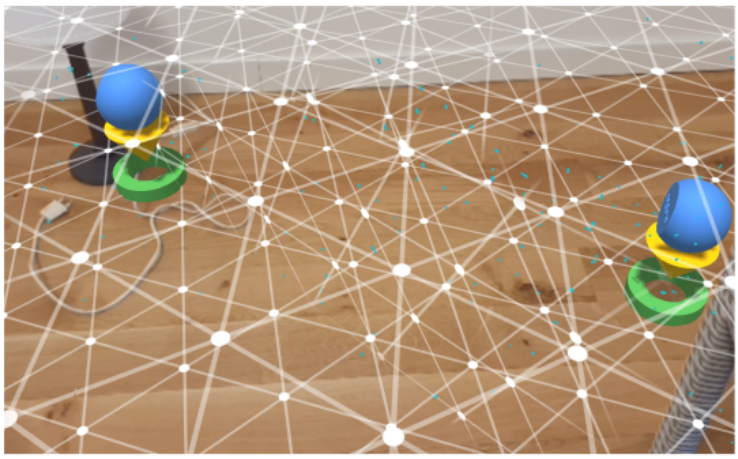

Figure 7.11: Multiple planes intersections created by the AR Core session.

For what regards the model reconstruction in the scene, it freezes the application for less that a second only the first time that an object is retrieved, meaning that the other times the building and the instantiation does not block the application flow and the entire process from the reception to the placement of the object lasts around two or fewer seconds. For what regards the placement it can happen that the model has its scale a bit different with respect to the size of the real object, due to the fact that the 3D model size is defined using an average value between the different dimensions of the tested object.

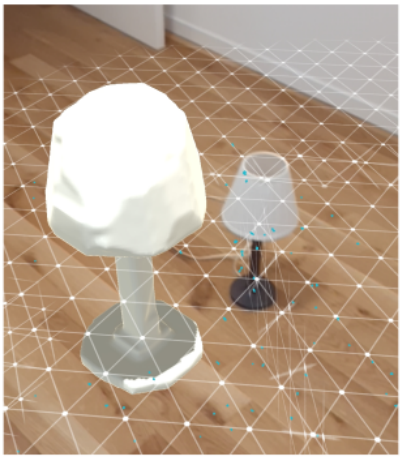

Figure 7.12: The real object compared to the model with an erroneous scaling applied.

As the application has been tested under continuous condition, it manages to successfully reconstruct more than ten objects of a single room, stressing its flow rate in a time span from ten to thirty minutes without any major issue. In the following photos are shown examples of scene reconstruction using different objects, positions and configurations.

Figure 7.13: Scene reconstructed using two different objects (on the left) and five different objects (on the right). The top left and the bottom right photos show the position of the placeholders in the environment.

Figure 7.14: Scene reconstructed using multiple instances of the same object.

8 Conclusion and Future Works

As conclusion of the project, it is possible to make several considerations regarding the application, the BSP-Net system and the overall reconstruction pipeline in order to sum up the possible implementations that could further improve the entire system described in this thesis as a future reference. Starting with BSP-net, the overall reconstruction was fairly acceptable on determinate situations, since the original dataset used for the training is balanced towards categories such as chairs, tables, lamps, planes and cars, highlighting that the application performs quite well when it faces one of these objects.

According to the goal task of the project it is suggested, in order to achieve better results, to re-train the network with all the indoor categories which the user is interested in and creating a new dataset based on real images and not on synthetic ones, in order to align the training phase to what will be tested later on. These steps were not possible to perform, considering that during the realization of this project it was not possible to access to required resources in terms of hardware and

computational capability. In fact to successfully achieve an improved training, the requested components should be quite powerful, since it affects the computational cost in terms of time which exponentially grows using older hardware; for instance, using an NVIDIA geForce RTX 2080 Ti GPU to train the model it will require more than 5 days to complete the entire training task.

Another improvement that is possible to perform is on the input images fed to the system, which are small-scale and with low quality (128x128), probably to reduce the training time and to compare the results with previous standard benchmarks. Therefore, with a high quality hardware and a considerable amount of time for training, it should be interesting trying to augment the dimension of the tensors of the network in order to enhance the resolution of the input images at least to 1080x1920 to further improve the quality of the results.

As mentioned before, BSP-Net is based on a Single View Reconstruction approach, meaning that there could be a margin of improvement for multi-shots images, taking multiple views of the same object in order to collect more information as possible and merge them together in order to obtain a high level of accuracy during the reconstruction of the details of the single model.

Another interesting aspect regarding BSP-Net is the production of compact mesh, i.e., low-poly meshes, which would lead to the possibility to represent the output shapes through a "difference" operations rather than a "union" of different parts, in order to achieve a wider generalization approach that can express complex and concave or convex details. For what concerns the reconstruction side, several improvements should be done to retrieve a high quality result. The first step that could be improved is the semantic segmentation, as it has been evinced by the results that the principal cause of a low quality reconstruction is caused by a wrong segmentation of the object. Sometimes the network is not able to recognize empty spaces between object or even the form that they have. In order to overcome this gap, with the same assumptions of BSP-Net it should be performed an intense session of training with a larger dataset to let to the network to segment more complex objects.

Another improvement can be performed on the remeshing module, whose algorithm could be upgraded to detect and fill the empty gaps that sometimes are present in some models due to an error of reconstruction (e.g., a chair with a hole in the back) and eliminating unnecessary parts which are not connected to the main mesh.

It could be also interesting to upgrade the system switching from a normal RGB camera to an RGBD one, in order to acquire the depth map along the standard image. This could allow the system to access more information such as the actual difference in depth between the object and the other parts of the environment, achieving a better result in terms of segmentation.

Finally, it is possible to debate about the Android Application, which is the interaction device between the user and the reconstruction pipeline. As mentioned, it is possible to take photos with the smartphone camera and its resolution is set to 1080x1920, then the frame will be successfully resized and cropped by the server. If the BSP-Net enhancement quality mentioned before could be applied, it could be also possible to increase the photo quality of the actual camera of most of the phones, which can reach 4K resolution, providing a larger number of details that can help for a realistic reconstruction of the model.

Another problem of the actual application is given by the scaling size of the instantiated model, which is defined as an average value between the different sizes of the object which is tested and deployed in the scene, therefore could be helpful to develop a function to automatically create a bounding box around the object whose in order to

be able to adjust the scale of the model which will overlap to it.

For what regards the model instantiated in the scene, there are little improvements that can be applied in order to increase the experience of the user in the application. One of these could consists in the introduction of a menu to change the appearance of a specific model from a set of available textures, since now the only texture automatically applied is a default white material, which is a Unity standard for models without any specific material. Concerning the manipulation of the object which allows a user to move, rotate and scale a selected object, occurs that in the circumstance in which different number of reconstructed objects are present, could be difficult to recognize the actual object subjected to the manipulation. Hence, could be interesting to introduce an indicator or marker, which univocally points or highlights the manipulated object.

Another improvement concerns the Gallery, which does not provide anything other than the image visualization of the photo taken by the app; it would be interesting providing additional functionalities by adding the possibility to delete a specific photo, also eliminating the indicator and the respective model along with it, exploiting its index; also a function to save the images in the internal storage of the smart-

phone could be useful.

Finally, what could be interesting is a complete redefinition of the archetype of the application, which now is based on taking snapshots of the single object and reconstructing it, moving towards a new system where the user records a video of the indoor scene and sends it to the Reconstruction Side, which autonomously identifies all the objects contained in the video of the scene, decomposes each one segmenting them from the background, reconstructs each one of them independently and sends them back to the client.

It could also be possible to implement a system which exports the entire visualized scene in Augmented Reality, transferring the position, dimension and rotation of all the objects contained into it in a different environment as a PC for instance, in order to visualize and modify it under a completely new level of freedom.

List of Figures

Bibliography

[1] *ALICEVISION: Photogrammetric Computer Vision Framework*. URL: https://alicevision.org/.

[2] *Apple wants to make Lidar a great deal on iPhone 12 Pro and above. What is it and why is it important*. URL: https://www.haveeru.com.mv/apple-wants-to-make-lidar-a-great-deal-on-iphone-12-pro-and-above-what-is-it-and-why-is-it-important/.

[3] *Basics of Multilayer Perceptron – A Simple Explanation of Multilayer Perceptron*. URL: https://kindsonthegenius.com/blog/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron/.

[4] *Chainer Tutorial: Sentiment Analysis with Recursive Neural Network*. URL: https://medium.com/@keisukeumezawa/chainer-tutorial-sentiment-analysis-with-recursive-neural-network-180ddde892a2.

[5] 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.

[6] Özgün Çiçek et al. “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation”. In: *CoRR* abs/1606.06650 (2016). arXiv: 1606.06650. URL: http://arxiv.org/abs/1606.06650.

[7] *Components of Kinect for Windows*. URL: https://subscription.packtpub.com/book/game_development/9781849692380/1/ch01lvl1sec08/components-of-kinect-for-windows.

[8] *ConvNet Architectures for beginners Part I*. URL: https://medium.com/srm-mic/convnet-architectures-for-beginners-part-i-233aa9d1761b.

[9] *Depth of the neural network for computer vision processing it*. URL: https://programmersought.com/article/93883332619/.

[10] Wei Dong et al. “An Efficient Volumetric Mesh Representation for Real-time Scene Reconstruction using Spatial Hashing”. In: *CoRR* abs/1803.03949 (2018). arXiv: 1803.03949. URL: http://arxiv.org/abs/1803.03949.

[11] *Generative adversarial networks: What GANs are and how they've evolved*. URL: https://venturebeat.com/2019/12/26/gan-generative-adversarial-network-explainer-ai-machine-learning/.

[12] *How did technology transform the retail industry?* URL: https://www.quora.com/How-did-technology-transform-the-retail-industry.

[13] *How Recurrent Neural Network (RNN) Works*. URL: https://openbootcamps.com/how-recurrent-neural-network-rnn-works/.

[14] *Introduction To 3D Deep Learning*. URL: https://medium.com/@nabil.madali/introduction-to-3d-deep-learning-740c199b100c.

[15] Michal Jancosek and Tomas Pajdla. “Multi-view reconstruction preserving weakly-supported surfaces”. In: *CVPR 2011*.

IEEE, June 2011. DOI: 10.1109/cvpr.2011.5995693. URL: https://doi.org/10.1109/cvpr.2011.5995693.

[16] *LEGO Digital Box brings Augmented Reality to LEGO Stores Worldwide*. URL: https://www.mobilevenue.com/lego-digital-box-brings-augmented-reality-lego-stores-worldwide-04190305/.

[17] *Manufacturing with VR Becoming a (Virtual) Reality*. URL: https://www.qad.com/blog/2018/09/manufacturing-vr-becoming-virtual-reality.

[18] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV].

[19] Pierre Moulon, Pascal Monasse, and Renaud Marlet. "Adaptive Structure from Motion with a Contrario Model Estimation". In: *Proceedings of the Asian Computer Vision Conference (ACCV 2012)*. Springer Berlin Heidelberg, 2012, pp. 257–270. DOI: 10.1007/978-3-642-37447-0_20.

[20] R. A. Newcombe et al. “KinectFusion: Real-time dense surface mapping and tracking”. In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[21] Matthias Nießner et al. “Real-Time 3D Reconstruction at Scale Using Voxel Hashing”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013). ISSN: 0730-0301. DOI: 10.1145/2508363.2508374. URL: https://doi.org/10.1145/2508363.2508374.

[22] Songyou Peng et al. *Convolutional Occupancy Networks*. 2020. arXiv: 2003.04618 [cs.CV].

[23] *Photogrammery testing 14: AliceVision Meshroom*. URL: https://peterfalkingham.com/2018/08/11/photogrammery-testing-14-alicevision-meshroom/.

[24] Charles Ruizhongtai Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[25] *Residenti aperti per la struttura di formazione VR*. URL: https://www.excite.co.jp/news/article/MoguraVR_vr-medical-training/.

[26] O. Ronneberger, P.Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, pp. 234–241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[27] Shunsuke Saito et al. *PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization*. 2020. arXiv: 2004.00452 [cs.CV].

[28] Steven M. Seitz and Charles R. Dyer. "Photorealistic Scene Reconstruction by Voxel Coloring". In: *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*. CVPR '97. USA: IEEE Computer Society, 1997, p. 1067. ISBN: 0818678224.

[29] Daeyun Shin et al. "Multi-layer Depth and Epipolar Feature Transformers for 3D Scene Reconstruction". In: *CoRR* abs/1902.06729

(2019). arXiv: 1902.06729. URL: http://arxiv.org/abs/1902.06729.

[30] *Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications*. URL: https://www.semanticscholar.org/paper/Sim4CV.