POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

3D RECONSTRUCTION OF INDOOR ENVIRONMENTS

Supervisor: **Candidate:**

Prof. ANDREA SANNA **FRANCESCO TAMBURELLO**

March 2021

Contents

1 Introduction

1.1 How the World view is changing

In the modern era, most of the tasks that are assumed to be done by humans are slowly, but steadily, being replaced by computers. Most of the algorithms are capable to exceed by far the work of people, reducing the time to complete it and increasing the accuracy and the quality of the work. In addition, the continuous development allows discovering new and powerful tools to facilitate the everyday routine. It is impossible to deny that at present, the machines influence all aspects of our life, from the trivial act of checking the weather forecast, to the entire management of our life as the way we interact and establish connection with other people around us. In fact, in less than 20 years, the technological innovation has seen an exponential growth in all possible fields of applications and in all types of industries, reaching goals which were unthinkable only a couple of years before. As already mentioned, we discovered ways to make possible for comput-

ers to accomplish tasks and learn by themselves, for example using peculiar structures defined "Neural Networks" (NNs), which have the principal purpose to emulate the cognitive process of decision and recognition. Considering those as the starting point, as the central theme of this thesis we can refer to one of the most advanced field of research in the spotlight, which is changing the interaction between the machine and the world as well between the human and the world. This theme can be defined using the term *Deep Learning* (DL), which is related to the more general concept of Machine Learning (ML), that is per se self-explanatory inasmuch it searches and defines a series of methods to make computers perceive and understand the external world in the same way we perceive it. The innovation of this idea resides in the fact whereby, if the computer manages to perceive the external world, it becomes self-aware of it in a semantic sense, making it able to manipulate and extract data from what it sees. Nowadays many applications of DL are in the field of Computer Vision (CV), but other fields such as Physics Simulation and Computer Graphics are also seeing a surge in academic endeavors inspired by ML core ideas. There are a multitude of applications that include a CV system integrated with other tools to have improved the performances in spe-

cific tasks.

Figure 1.1: Examples of different Computer Vision applications. Image taken from [33].

One of the most interesting applications of the computer vision concerns the integration of the reality with the virtual environment, which led to the creation of different applications. We can refer to those as Augmented Reality (AR), where the virtual world and the real one overlap, mixing digital elements in a real environment, for example by using a smartphone device, and Virtual Reality (VR), where the user is transported entirely in a digital environment with which the user can interacted, most commonly by using specific devices defined

as Head-Mounted Displays (HMDs). Some commercial VR products like the Oculus Quest or the Sony PlayStation VR have sold millions of units as of 2021. These applications open a vast amount of possibilities in terms of entertainment in general and video games principally, but are also finding a consistent employment in different fields such as healthcare, education, military, manufacturing and so on. It is not visionary to consider that in a not so distant future, these reality augmentation devices will be implemented in all aspects of society, changing the everyday life in the same way that computers did almost fifty years ago.

Considering the AR as the main objective of this essay, it was decided to engineer and develop a system allowing users to recreate and visualize 3D models of an indoor environment by using the RGB camera of a smartphone integrated with the functionalities provided by the Google ARCore framework. This is an attempt to give to users a tool to digitalize the environment and manipulate it, eliminating the disadvantage to use pre-built models. The smartphone application allows the user to take a photo of an object and send it to a server, which has the task to build the 3D model from the photo received and return it to the application, where it will be displayed on the AR environment.

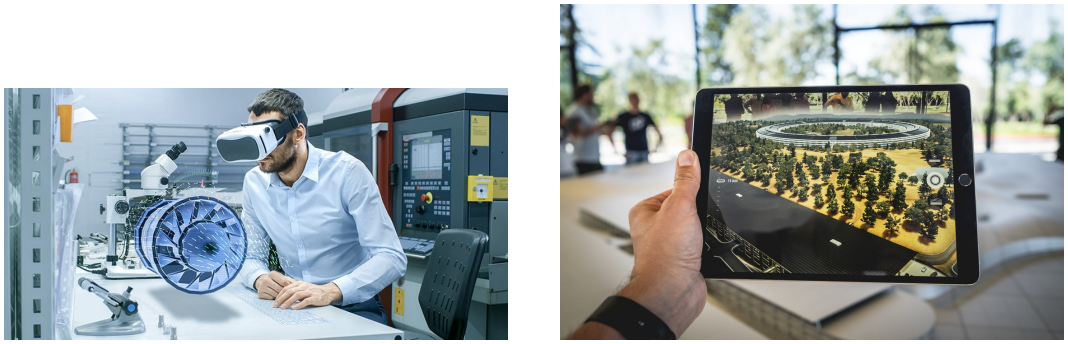

Figure 1.2: A Virtual Reality application (on the left) and an Augmented reality application (on the right). Images taken from [20] and [14].

The first chapter is a brief introduction with a general view of the technologies in today's world, followed by a discussion on the AR and VR. The second chapter investigates the basic concepts for a better comprehension of the mechanism employed in the thesis. In the third chapter a discussion of the state of the art on object reconstruction is presented, and preliminary results on the most promising existing reconstruction methods are described. The fourth chapter presents the general idea of the proposed system and its functioning, splitting the description between the application and the server side. The fifth chapter discusses in detail the mechanisms of the client side and its implementation, whereas the sixth chapter discusses the tasks per-

formed by the server in order to achieve a successful reconstruction of a model, relying only on the image of the object. In the seventh chapter the results obtained in the final implementation of the server and client and the overall accuracy obtained are presented, as well the time employed to perform a complete cycle, from the start of the application, to the final visualization of the model in an AR environment. The eighth and last chapters contain a discussion regarding the achievement obtained, the possible improvements and fields of application that the system can have.

2 State of the Art

The Computer Vision aims to recreate the human vision and its way to process and interpret images or videos, and extracts features from it using deep neural networks as support and the integration of cam and software for the acquisition and elaboration of images, producing a digital electric signal as output to be elaborated.

2.1 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 Time-Of-Flight (TOF) lasers, LiDAR (Light Detection and Ranging) as in Fig.2.1 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. These 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.

2.2 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

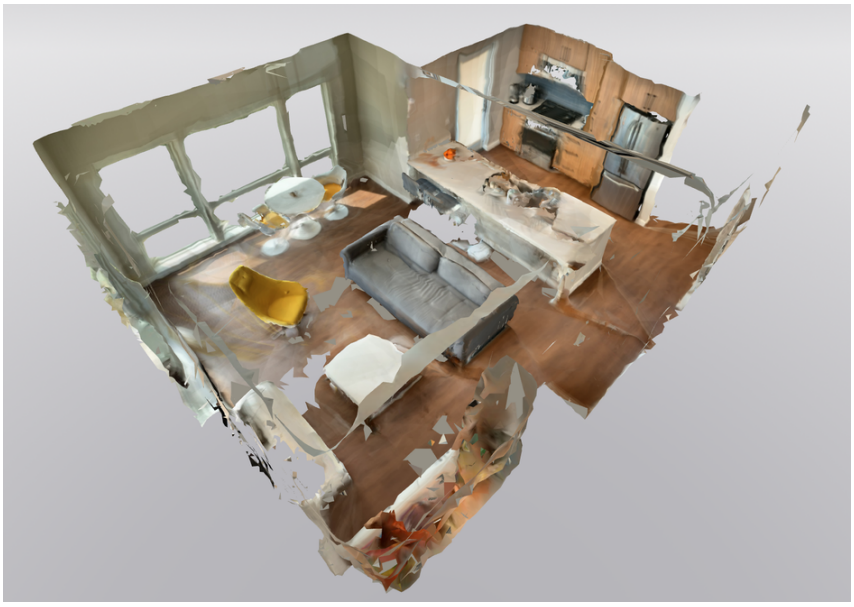

Figure 2.1: Example of Room Scanning using the Iphone 12 Pro LiDAR. Image taken from [2].

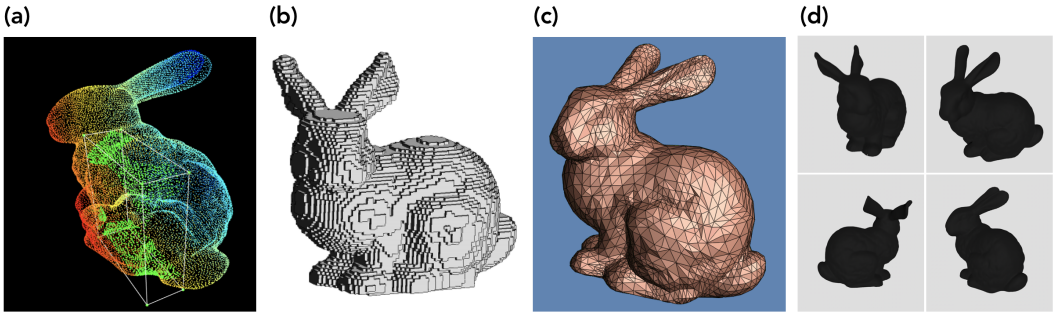

Figure 2.2: The different representation of a generic object using Point clouds (a), Voxels (b), Meshes (c) and Implicit representations (d). Image taken from [16].

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 [24, 31].

Point clouds Point clouds are an aggregation of points placed on a 3D space characterized by not having topological information. Together with position, other values such as luminosity, color and depth may be associated to points. Those are broadly used in the representation of large three-dimensional structures when scanners or 3D sensors 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 into more faces, at the expense of higher memory requirements and more processing power needed to draw those faces on screen. This representation was the subject of investigation in different articles such as [11, 21, 32].

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

2.3 Traditional Approaches

In Computer Vision the methods which do not use the Machine Learning and Neural Networks to reconstruct the object are defined as Active Methods. Those can also be defined as traditional since they 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 drawback was the necessity to be able to interact with objects, meaning that were limited to the surrounding area where they were set. We can now define some of these traditional approaches.

2.3.1 Kinect Fusion

An interesting research is presented by *KinectFusion: Real-Time Dense Surface Mapping and Tracking* [23], that shows a system which accurately maps objects and indoor scenes in real-time. The entire system is powered by a Microsoft Kinect sensor and uses an optimized Iterative Closest Point (ICP) algorithm. The Kinect was a device originally released by Microsoft in 2010 as a gaming peripheral for the Xbox

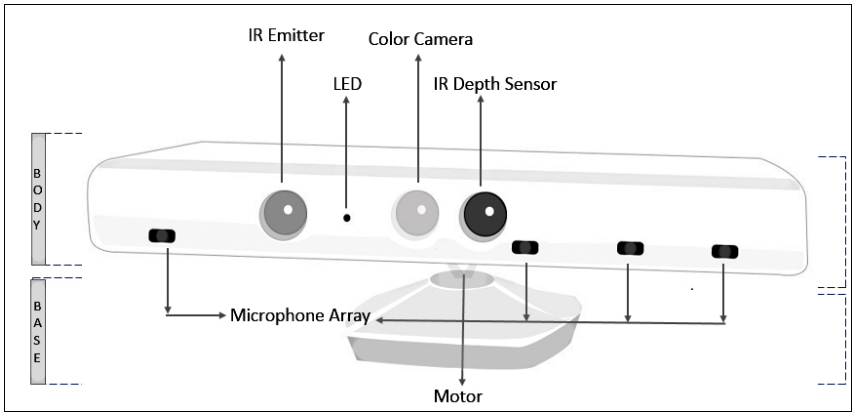

Figure 2.3: The Structure of Kinect. Image taken from [7].

360 console; although nowadays it is not used anymore for gaming, it became quite popular in the academic community. The Kinect is composed by an RGB camera, an infrared camera, a 3d scanner that maps depth using structured light or TOF calculation, a microphone array and a base with a motorized pivot.

The approach described in the paper solves the reconstruction problem similarly to how the Simultaneous Localization and Mapping (SLAM) works and is composed by four modules:

1. 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.

2. 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.

3. 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).

4. The fourth and 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.

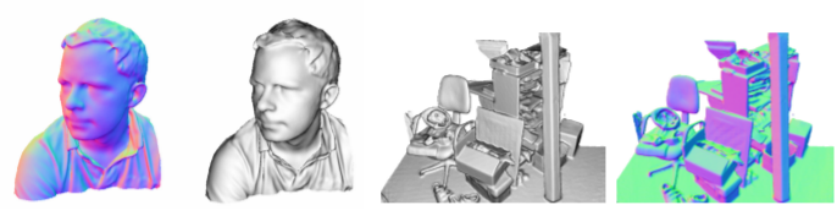

Figure 2.4: The results produced by the Kinect Fusion system.

2.3.2 AliceVision

AliceVision is a photogrammetric framework which can provide 3D reconstructions using camera tracking algorithms [17, 22]. 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 it 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 large 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 view-

points, using a SIFT (Scale-Invariant Feature Transform) algorithm, extracting natural features, then it uses an image matching algorithm 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 representation of the object.

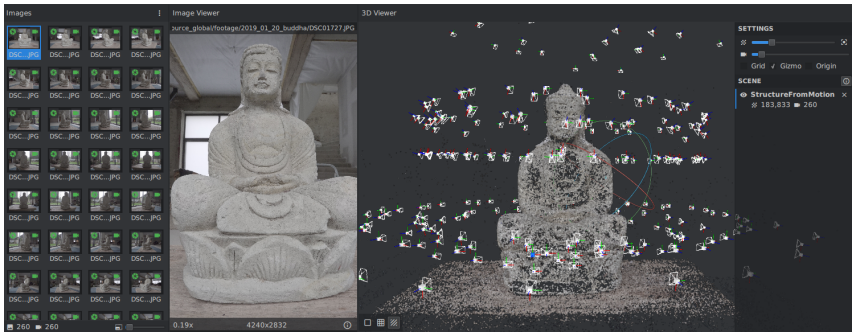

Figure 2.5: The scene reconstructed using pointclouds in AliceVision. Image taken from [1].

The system then tries to build a mesh from the points of the pointclouds 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 sys-

tem computes UV maps and textures of the scene in order to apply color textures to the reconstructed object mesh. The entire process requires from 30 to 40 minutes for the reconstruction of a medium-sized object and its accuracy depends on 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 object is from 30 to 60 with a gap between the photos which is not larger that 10°.

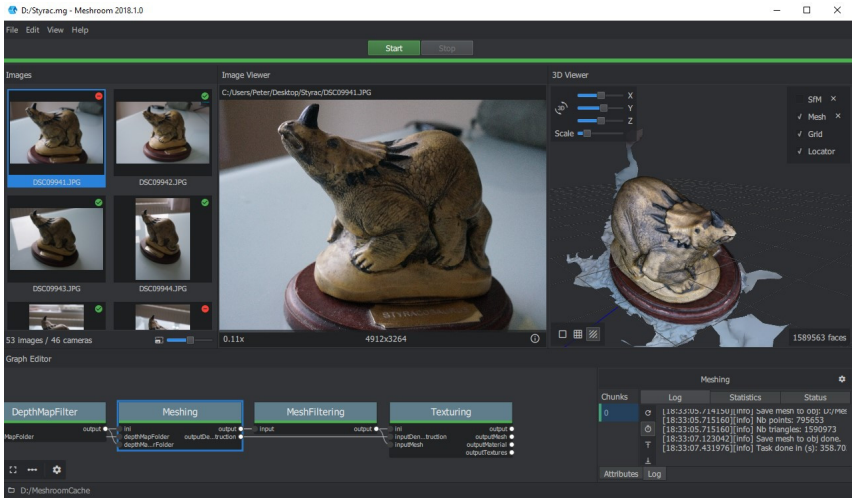

Figure 2.6: The results produced by the AliceVision system. Image taken from [26].

2.4 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 in 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.

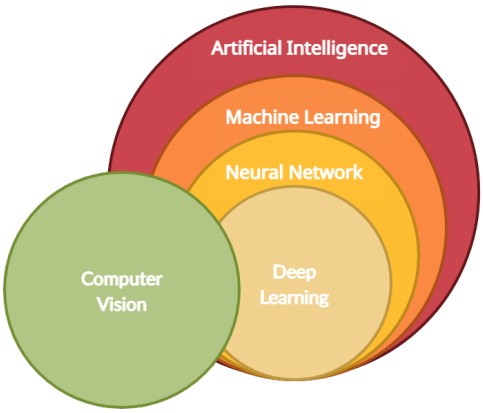

Figure 2.7: Venn Diagram of the relation between the Computer Vision and the Artificial Intelligence.

The process which allows the machine to learn how to perform a task is called *training*. Similarly to how animals learn from experience, machines can learn from large collections of data representing existing knowledge on a specific task. For example, we can identify cats in pictures because we see a large number of observations of them and we learn how to discern those features that are most commonly associated with cats; exactly in the same way a machine can learn how to recognize cats if we feed it with a large number of pictures of cats. 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.

2.4.1 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 layout. 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 weight respectively w_1 and w_2, it will have an input y_{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:

$$y_{out} = f(y_{in}, w_0) = w_0(w_1 x_1 + w_2 x_2) \tag{2.1}$$

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 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 weight used to make decision are not static, in fact the aspect of the NN similar to the units is the capability to adjust the weight of the neuron 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 elaborating

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 adjust the weights according to what has been learnt so far.

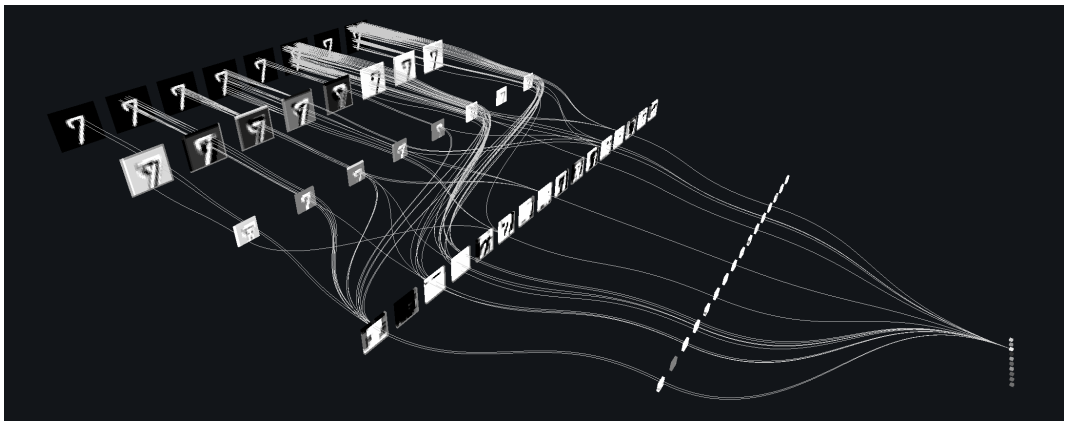

Figure 2.8: The connections in a generic structure of a Neural Network on a 3D view. The example regards the number recognition. Image taken from [8].

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.

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.

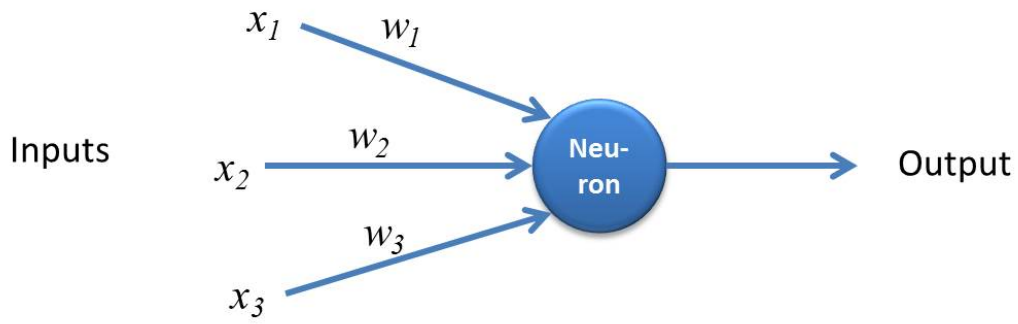

Figure 2.9: The scheme of a Single Layer Perceptron. Image taken from [3].

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.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of NN derived from the feedforward and differs from them for its network layout that de-

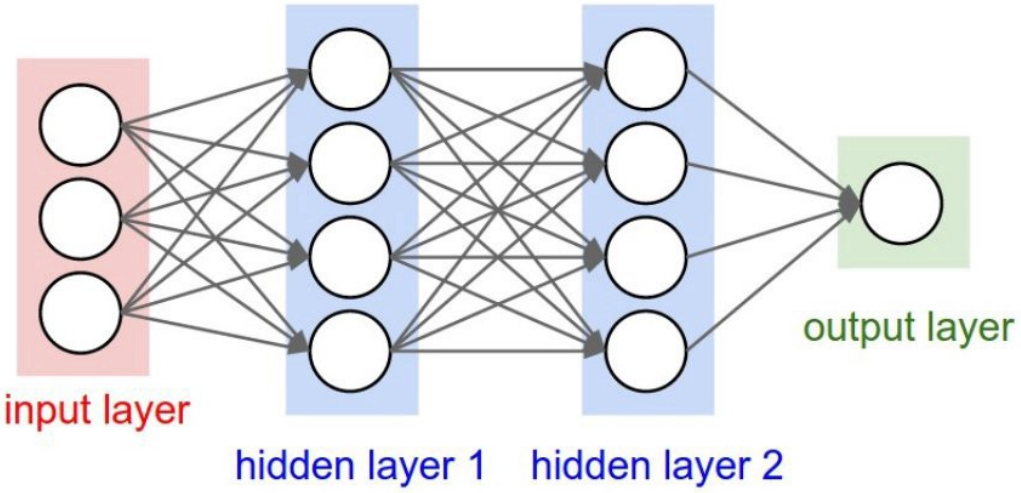

Figure 2.10: The scheme of a Multi Layer Perceptron. Image taken from [9].

fine a graph connection between nodes, forming a sort of temporal sequence, allowing the dynamic modelling of the network.In addition, it possess 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)

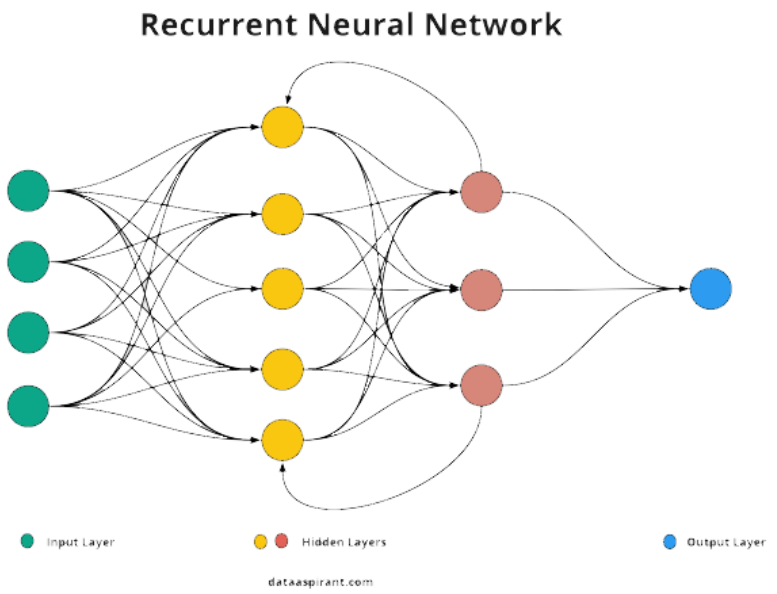

Figure 2.11: The scheme of a generic Recurrent Neural Network. Image taken from [15].

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.

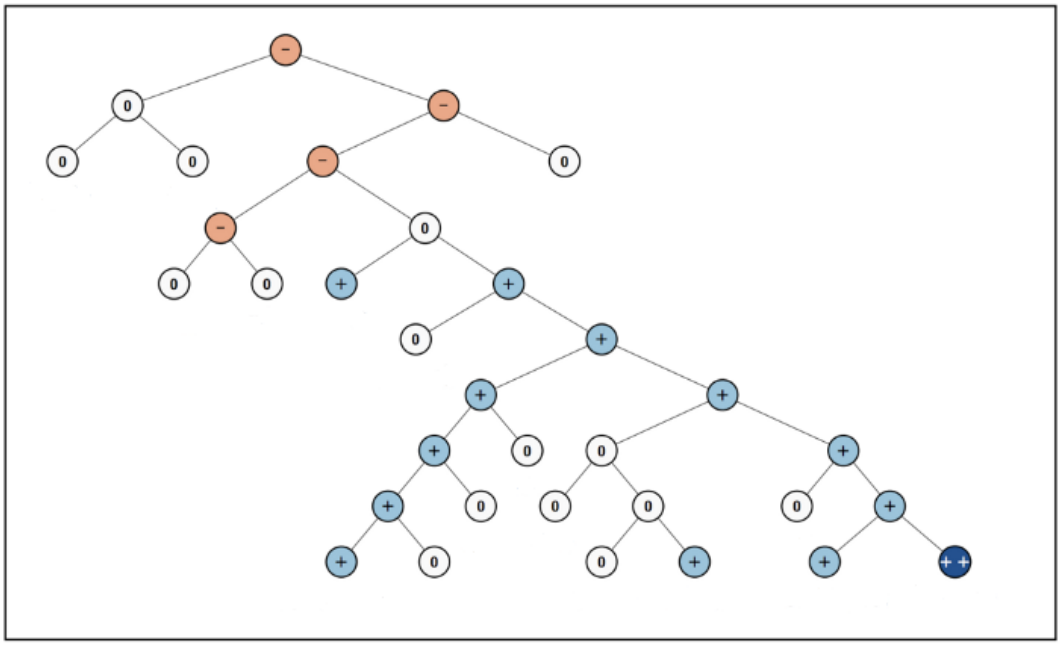

Figure 2.12: The scheme of a Recursive Neural Network. Image taken from [4].

Convolutional Neural Network

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 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

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

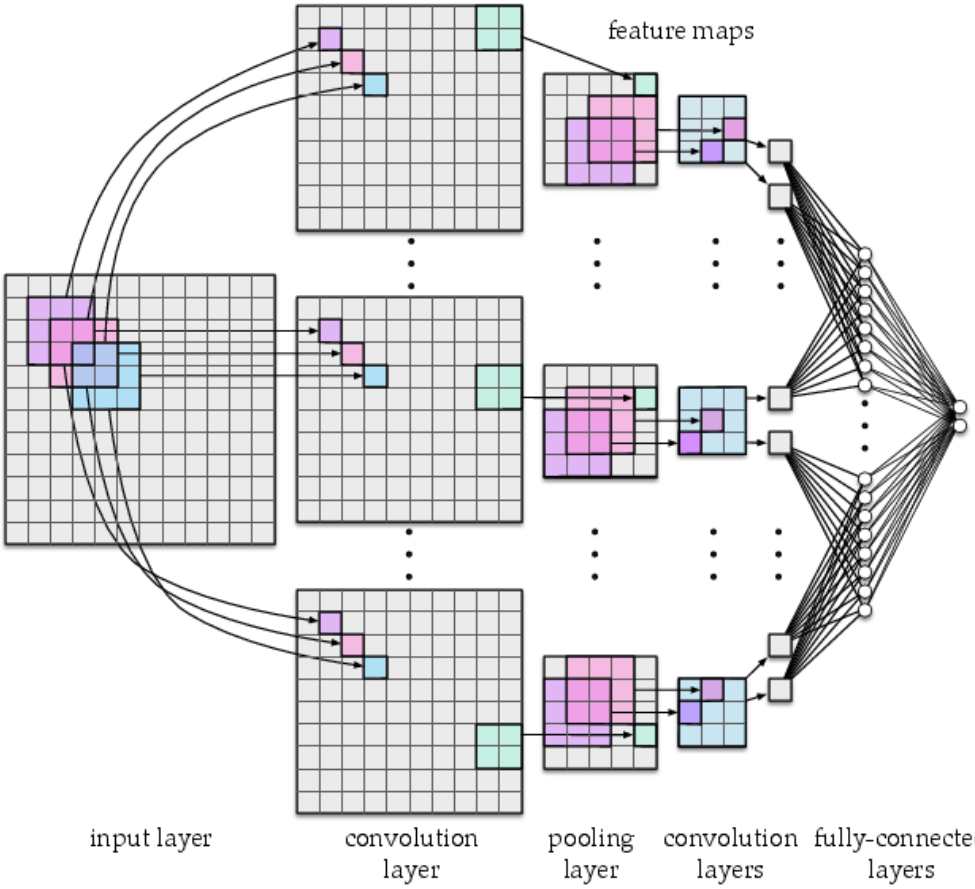

Figure 2.13: The scheme of a Convolutional Neural Network.

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 2.14: Faces reproduced by a GAN, given certain photos as inputs. Image taken from [13].

2.5 State-of-The-Art Systems

Having introduced the Neural Networks, we can now describe some of the most advanced techniques in the field of 3D reconstruction, which make use of learning-based methods to improve the performance in terms of quality of results produced. Those systems were also tested in the phase of project design, to evaluate the performances in order to decide if were suitable for the system to be implemented.

2.5.1 Convolutional Occupancy Network

Regarding Convolutional Occupancy Network [25], 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 [27]. The encoder extracts features and maps them, constructing a planar or volumetric representation to encapsulate local information from neighborhood.

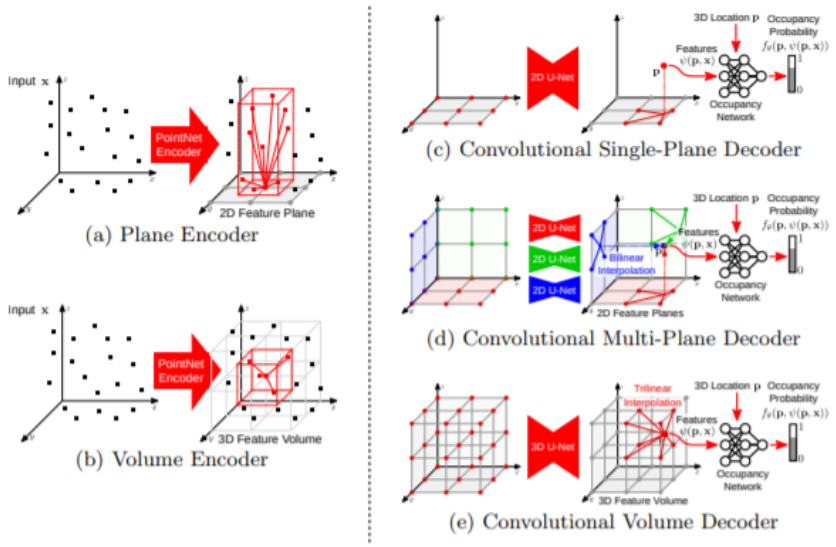

Figure 2.15: representation of the different types of encoder in the system.

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 [29], [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:

$$f_\theta(p, \phi(p, x)) \rightarrow [0, 1] \tag{2.2}$$

The final results of the reconstruction can be seen in (2.16)

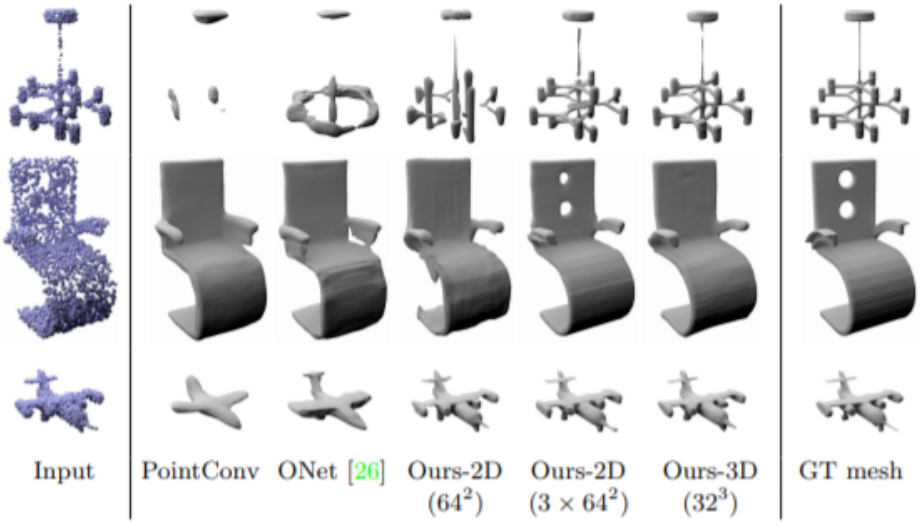

Figure 2.16: The comparison of result between different reconstruction's approach.

2.5.2 PIFuHD

PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization [30] 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.

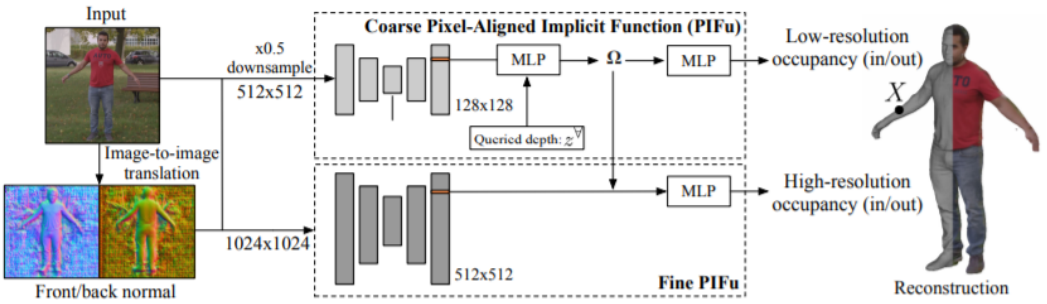

Figure 2.17: Schematic of the framework implemented in the PIFuHD system.

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:

$$f(X, I) = \begin{cases} 1, & \text{if } X\text{is inside mesh surface} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

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

$$f(X, I) = g(\Phi(x, I), Z), \tag{2.4}$$

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:

$$f^L(X) = g^L(\Phi^l(x_L, I_L, F_L, B_L,), Z) \tag{2.5}$$

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:

$$f^H(X) = g^H(\Phi^H(x_H, I_H, F_H, B_H,), \Omega(X)), \tag{2.6}$$

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.

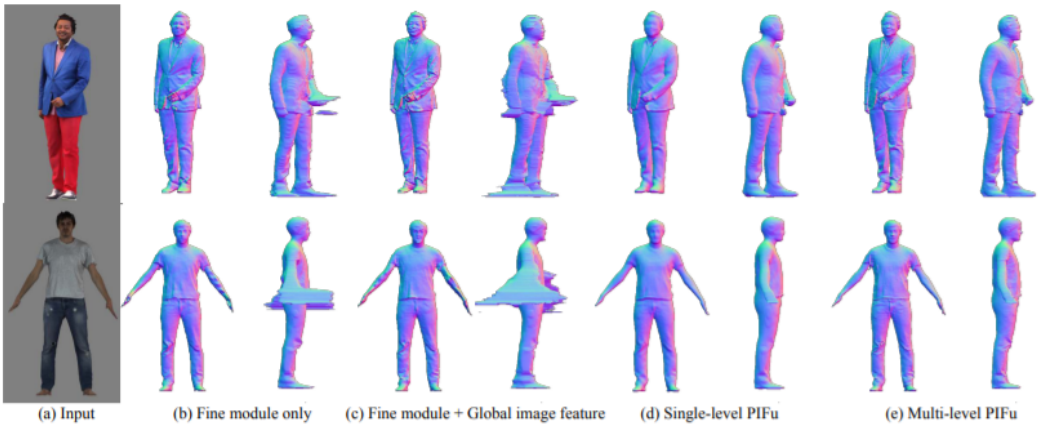

Figure 2.18: Result obtained trying different design of the system.

2.5.3 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

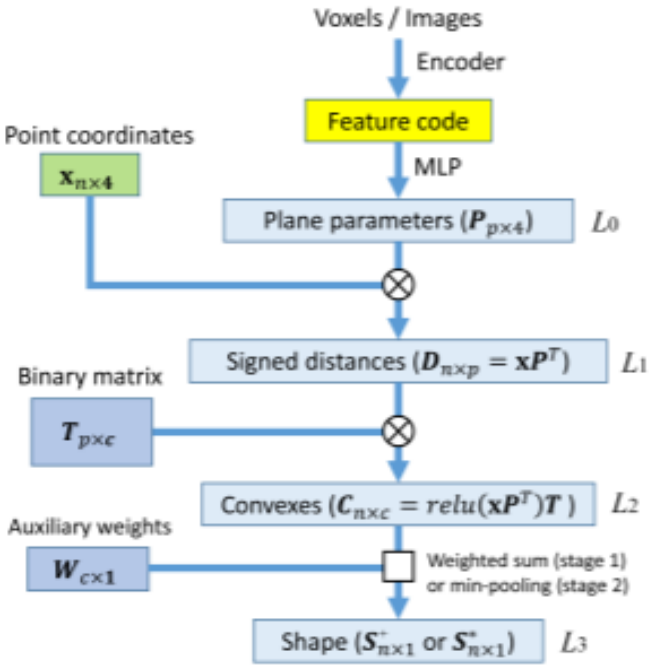

Figure 2.19: The organization of BSP-Net.

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:

$$C_j^*(x) = \max_i(D_i, T_{ij}) \begin{cases} < 0, & \text{inside} \\ > 0, & \text{outside} \end{cases} \tag{2.7}$$

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:

$$S^*(x) = \min_i(C_i^+(x)) \begin{cases} = 0, & \text{inside} \\ > 0, & \text{outside} \end{cases} \tag{2.8}$$

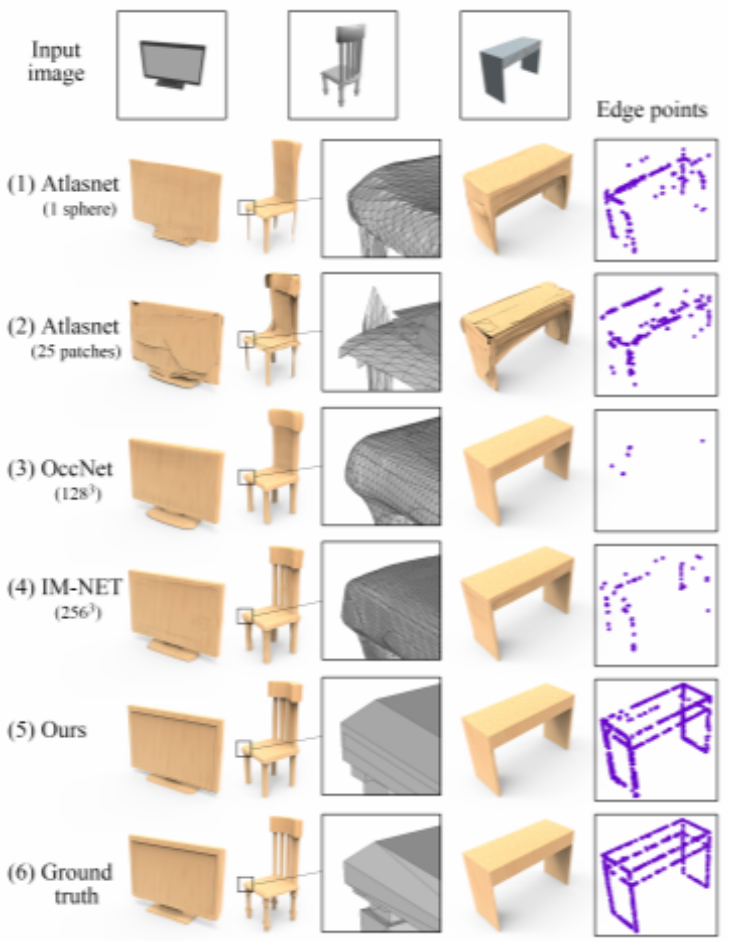

Figure 2.20: The result obtained by BSP-Net, compared to other object reconstruction networks and the ground truth.

2.6 Virtual Reality

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 informa-

tion 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 failure(fig. 2.21).

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 2.21: A medical students performs a simulation of treatment on a virtual patient. Image taken from [28].

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.

2.7 Augmented Reality

Virtual Reality 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. 2.22), or the staff training, or still the automatic detection and visualization of imperfections in products in a factory and so on.

Figure 2.22: 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 [19].

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.

3 Reconstruction System

The system was firstly designed to match 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 vertices and faces and could be divided into different pieces, therefore there was a need for the reconstruction to host 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 actions in around one minute. The starting step towards the realization of the project is to agree about a suitable reconstruction system which would satisfy the system requirements.

3.1 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. Moreover, it is also interesting to highlight and explain the reasons which led to the final decision. The tested systems were:

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

The implementation and testing were carried out on each of those systems on different class of objects. Starting from Convolutional Occupancy Network [25], this system was extremely similar to BSP-Net but it has substantial differences in the used algorithm and in the required input for the system. Convolutional Occupancy Network re-

quired as input a pointcloud file of the object which was extremely demanding, considering the architecture of the 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 Occupancy Networks was not at 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 in order to switch the focus of reconstruction from people to objects and undergo to intensive training to perform the operation of weighting of every neuron of the Network. This operation is

extremely heavy from a computational point of view and is 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 reasons, 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 good. 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 is 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 decreased significantly, giving, however, a result which was extremely poor, incomplete and in the most cases also different from the original. In the end, having as a requirement that the reconstruction process would take no more than a few dozens of seconds, it was prohibitive to force the user to take thirty photos and wait for more or less one hour just for a single reconstruction; for these reasons Meshroom was discarded. Finally,

the BSP-Net [5] 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 extremely convenient, given our system architecture. Hence, it resulted as the best choice between the considered solutions and it was decided to implement it in the project.

3.2 System Architecture

The Project has the final goal to produce a user-friendly application on smartphone for the reconstruction of 3D models of real objects and place them in the real environment exploiting the Augmented Reality. The Application allows the user to take photos of different objects, visualize them in the application, and manipulate 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.

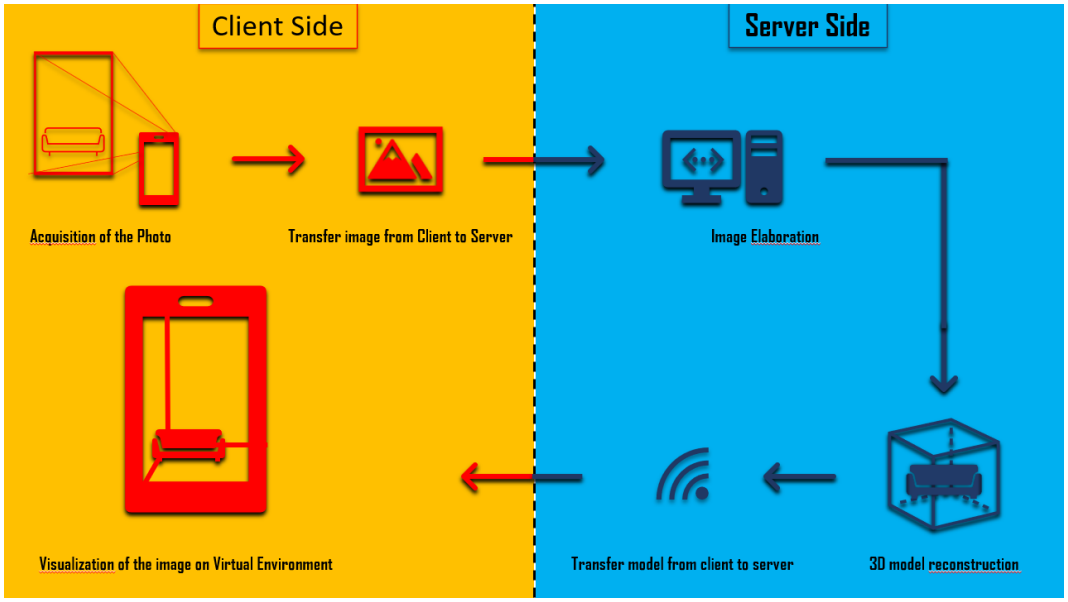

Figure 3.1: The general pipeline of the system.

3.3 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 large. 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 navigate the photos they have taken.

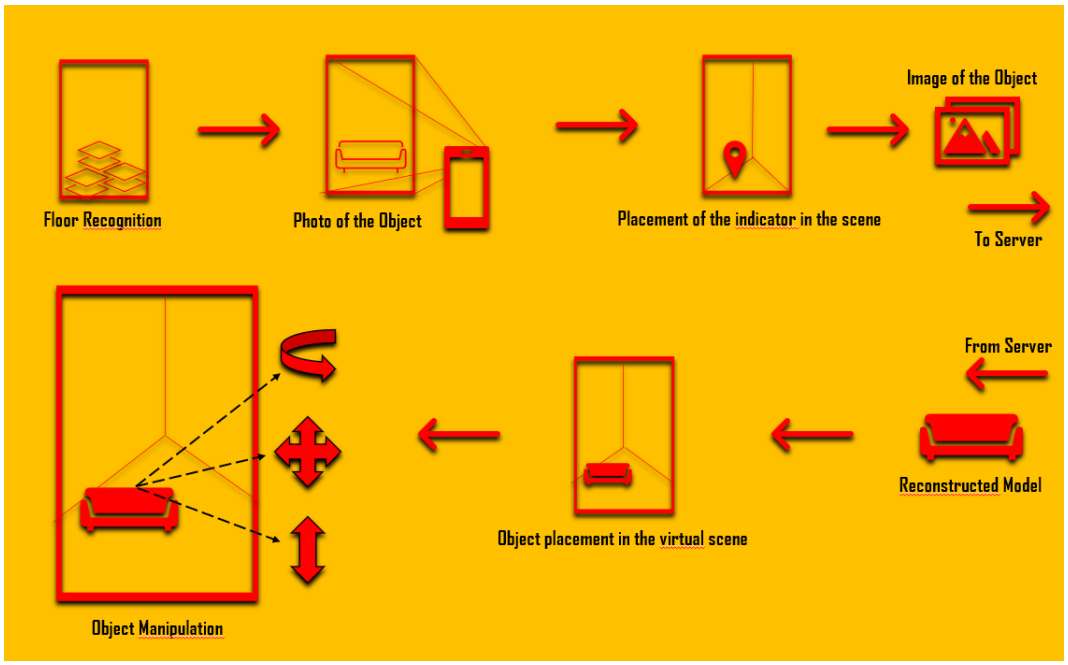

Figure 3.2: The scheme of the Client.

3.4 Reconstruction

The 3D reconstruction of the object is performed 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 its own result as input to the next block. When the server is started, it establishes immediately a connection with the client and waits to receive information from it. When those are received, first it decapsulates them in a legible way and pass those to the first block of the pipeline. The blocks start by segmenting the photo in classes of objects, and producing a grayscale image, where each level of gray correspond to a class of object. The next part, The Mask Extraction, processes the image created in the previous step and produces a binary mask to apply to the original photo, resulting in a new image with white background and the object positioned in the center. After the mask extraction, the masked photo is then passed to the object reconstruction block based on BSP-Net, where the program reconstructs the object present in the image into a 3D model, deducing its volume with the aid of a Binary

Space Partitioning(BSP). In this section of the pipeline, the mesh may have several imperfections, therefore the model undergoes through an operation of remeshing to adjust and improve the overall quality of its shape. Finally, the model is almost ready to be sent to the server but, before the actual forwarding of the data, another information needs to be found out. The object, when it is photographed, has a specific orientation in the space relative 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.

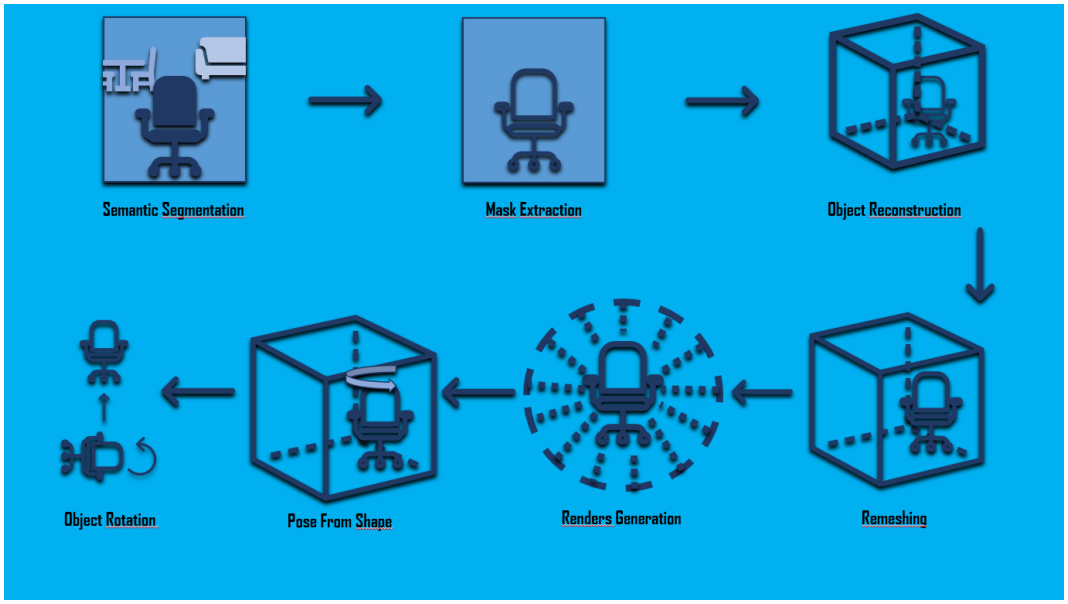

Figure 3.3: The scheme of the Server.

3.5 Acquisition and Placement Side

3.5.1 Organization and Composition

The main purpose of the client is to provide an accessible environment for the user to reconstruct and visualize the real scene 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 functionalities and the Unity Game Engine. Google Arcore is a software development kit provided by Google for

the creation of Augmented and Virtual reality applications. It makes it possible by tracking the phone with respect to the world with six degrees of freedom, and it is capable to take over the dimension and position of flat object like floor or tables and also estimate the light condition of a particular environment, adapting to it. Unity is a multi-platform game engine with the original purpose to allow the development of video games and interactive contents such as applications or animations. This program utilizes the C-Sharp language to program contents inside it, using an object-oriented approach, and embedding some specific functions for the manipulation of items and events inside the Unity environment. The project is mainly managed by 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.

3.6 The Lyfecycle

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

3.6.1 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, visualize the gallery or switch to the reconstruction mode. A simple flowchart regarding the main operations of the Main window can be seen in Fig.3.4.

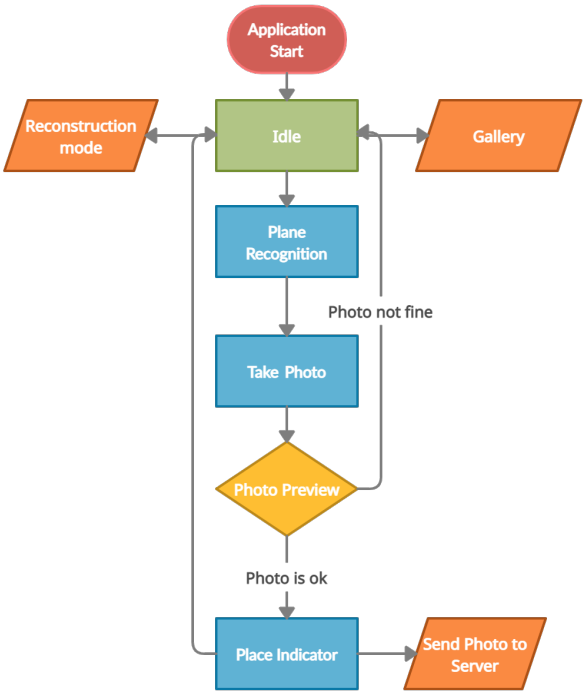

Figure 3.4: Flowchart of the Main window.

When the application starts, it is important to help the application to

recognize the floor or a flat surface, otherwise the user will not be able to place indicators and objects in the scene. When a photo is taken, the program shows a preview of that and if it is not in line with the standard of the user, the image will be removed and it will return to the photo mode to retake the photo. If the photo is good enough for the user, the program will automatically disable all the User Interface(UI) in the scene and the user will be forced to place an indicator in the scene in correspondence to the photographed object. In this way, an anchor will be placed whose information will be stored to track where the corresponding reconstructed model will have to be placed. Simultaneously to the placeholder positioning, the program will encapsulate the index of the photo(which also corresponds to the index of the anchor placed) and the photo, handing them over to the TCPConnectionHandler in order to be sent to the server.

3.6.2 The Reconstruction Mode

The Reconstruction mode or window is where the user can visualize the 3D models reconstructed by the server and manipulate them and it is accessible through the switch from the photo mode.

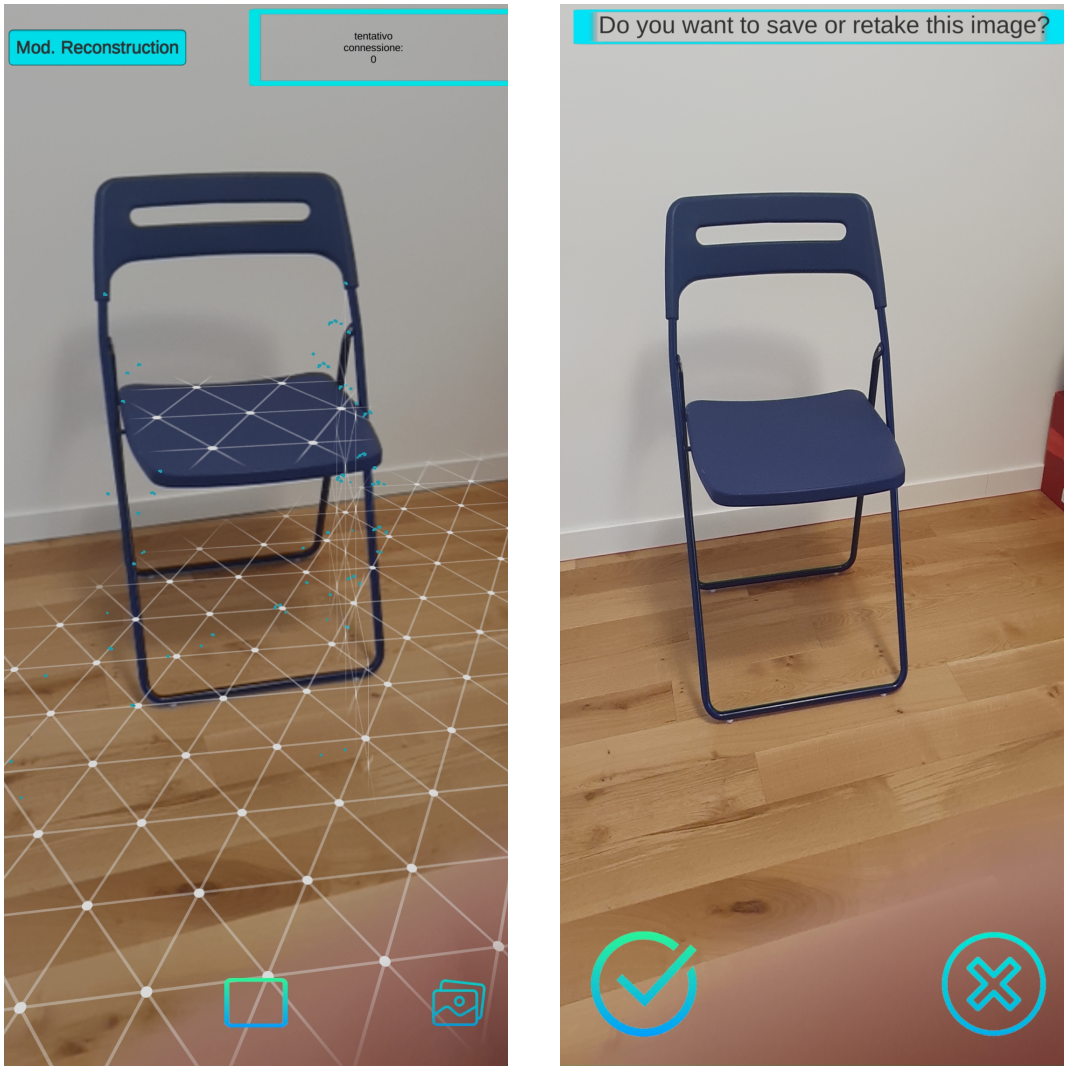

Figure 3.5: Showcase of the Main Windows on the left and the preview of the photo taken on the right.

The flowchart of the cycle of the reconstruction mode, can be seen in fig.3.6. When the user enters in reconstruction mode, it is assumed that he/she has already taken pictures of different object in the room

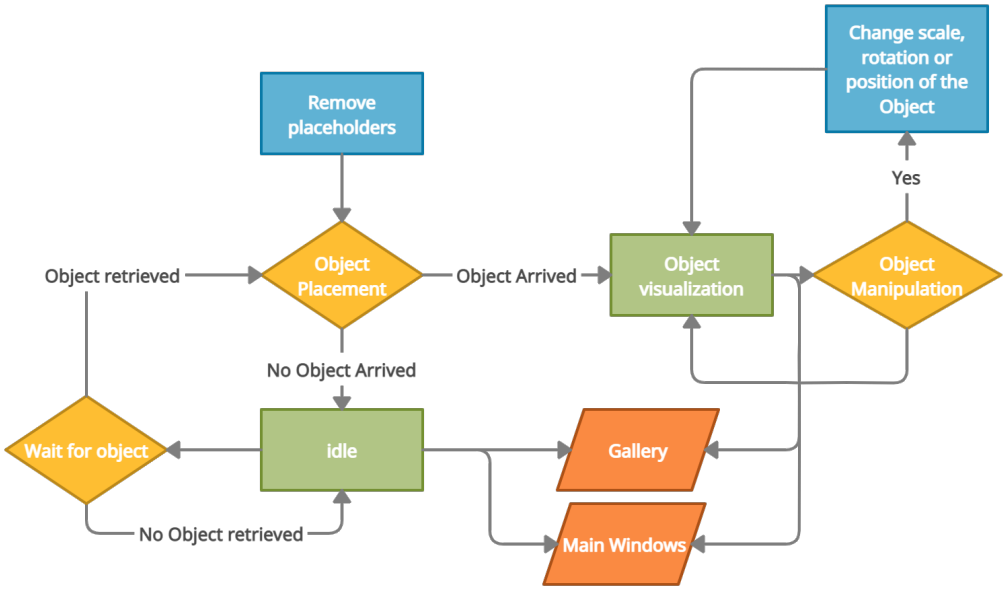

Figure 3.6: Flowchart of the reconstruction window.

and the server has returned at least one object. If it has not, the reconstruction mode will be inaccessible. 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 were previously placed. From here, he can tap on an object to make appear the object handler panel. This panel allows the rotation, scaling or translation of the object.

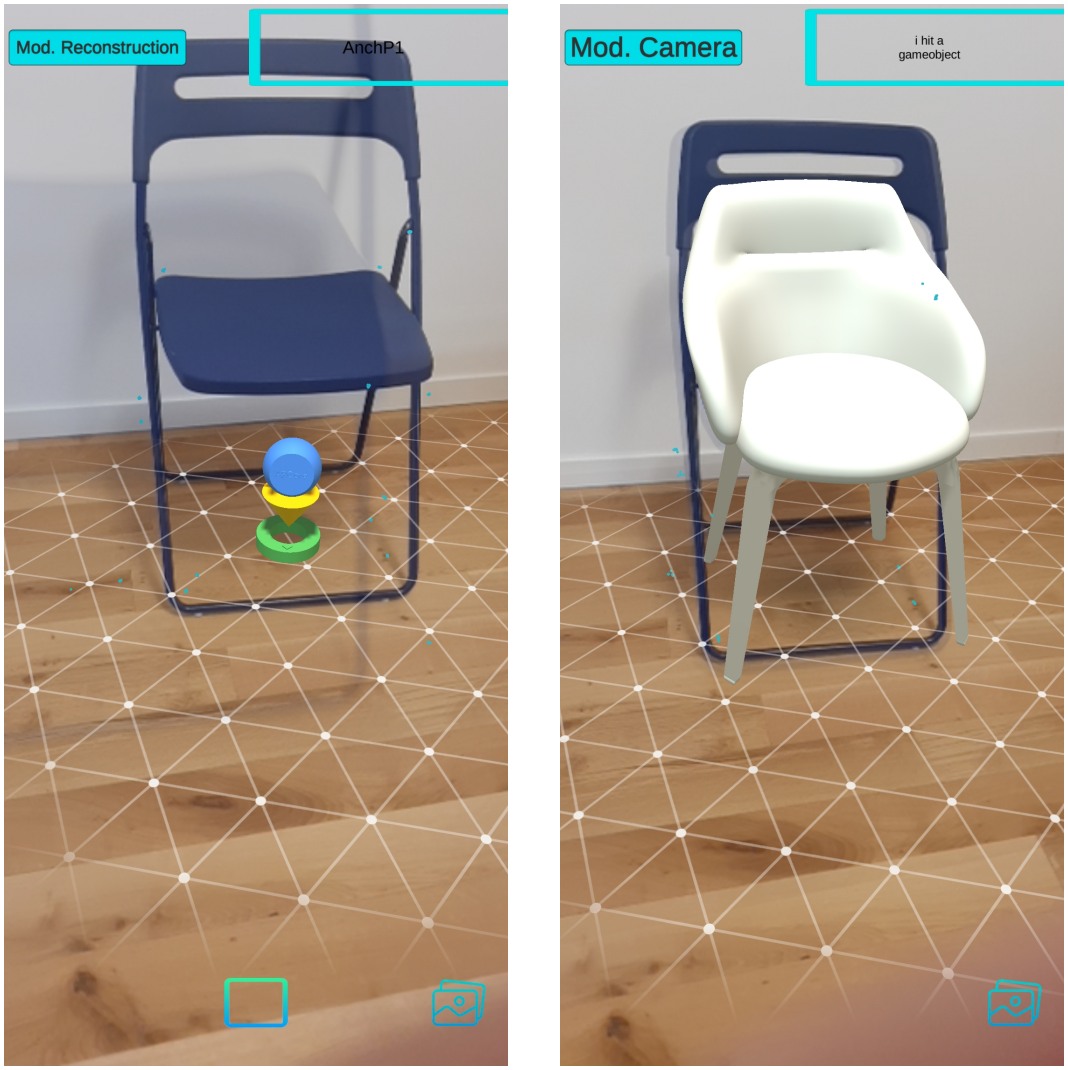

Figure 3.7: Placeholder placement on the Photo mode on the left and correspondent substitution with 3D model in the Reconstruction window on the right. The 3D model was the one used during the preliminary tests on the application.

3.6.3 The Gallery

The Gallery has a simpler workflow than the other two, and its main purpose it to visualize the images taken as a reminder of all the ob-

jects that were already photographed. This window consists of three buttons, two to navigate the gallery, the back and forward button, and one to return to the main windows.

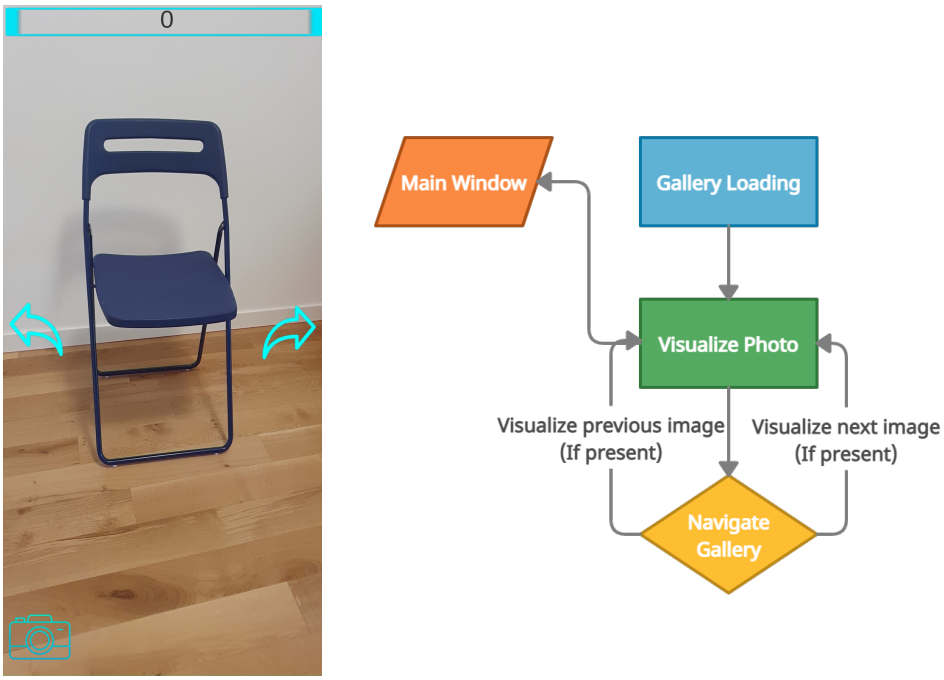

Figure 3.8: Showcase of the Gallery (on the left) and flowchart of the gallery (on the right).

3.6.4 The RGB image

A consideration should be done concerning the RGB image as not standard format for the camera of the Android devices. Instead, Android uses the YUV format, which is a color format used to code

images and video and is employed to mask transmission or compression error by adopting a reduced crominance bandwidth. This format is represented by three values, the Y which represents the luminance, whereas U and V that represent the chrominance of an image. The YUV format is used to indicate an analog value of a photo, and when it is digitalized, the format can be defined as YC_BC_R that represent respectively the deviation from the gray on the blue-yellow axis and on the red-cyan axis. The complete denomination of the Android standard is defined as YUV-420-888 where 888 indicates the number of bit used for each value of R (red), G (green) and B (blue) and 420 stands for the subsampling of the chrominance aspect (U-V), respect the luminance (Y), defining an actual ratio of 4:2:0 with a subsampling of the UV value of one half respect of the Y value. The zero should indicate that no value of V are sampled in this format, but actually, this format utilizes a single interlacing plane for the chrominance of U and V, forming a single UV plane that is subsampled of one half respect the luminance value, removing the needs of the third plane.

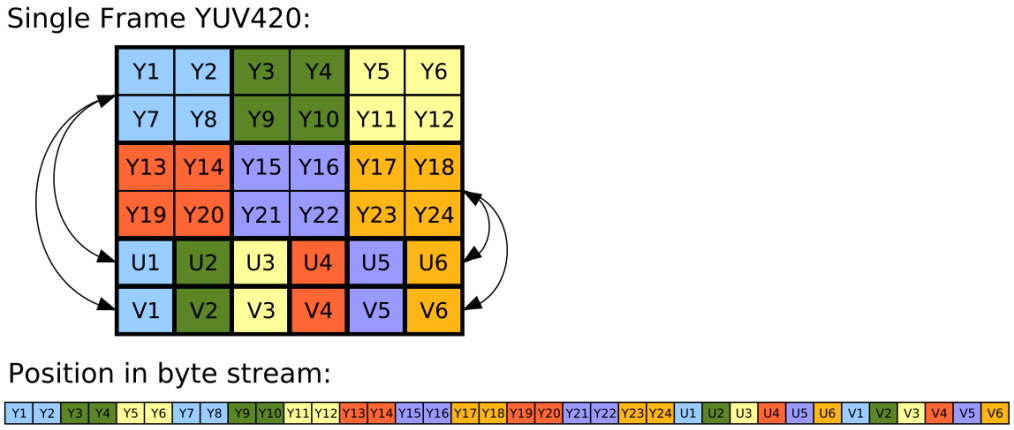

Figure 3.9: The distribuition of the YUV pixels in a RGB converesion. Image taken from [12].

Therefore, there was the need to transform the camera format in RGB in order to be saved in the gallery and sent to the server and the function to provide this was coded and embedded in the TakePhoto() function. We can report the actual extract of the conversion as follows:

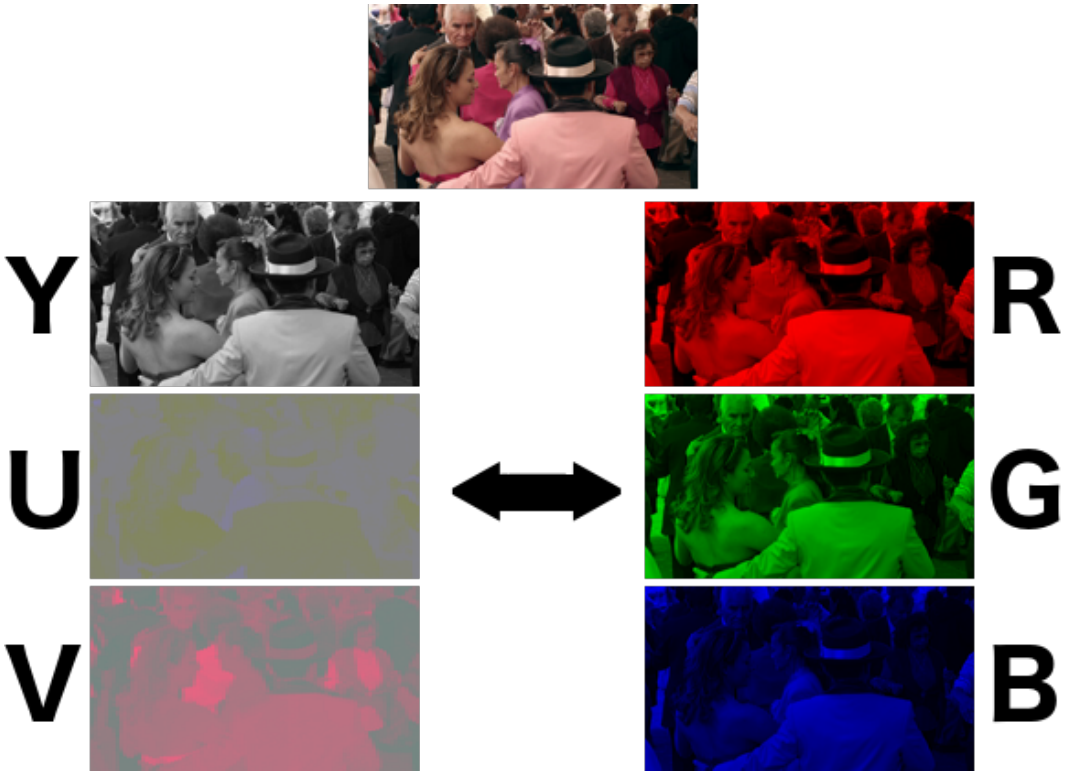

Figure 3.10: Representation of the different YUV planes respect the RGB ones in a generic photo. Image taken from [10].

```
1 Texture2D tex = new Texture2D( width, height, ...
      TextureFormat.RGB24, false );
2
3 byte[] bufferY = new byte[image.Width * image.Height];
4 byte[] bufferUV = new byte[image.Width * image.Height /2];
5 System.Runtime.InteropServices.Marshal.Copy(image.Y, ...
      bufferY, 0, image.Width * image.Height);
6 System.Runtime.InteropServices.Marshal.Copy(image.U, ...
      bufferUV, 0,image.Width * image.Height/2);
7 Color c = new Color();
```

```
8  for (int y = 0; y < image.Height; y++)
9  {
10     for (int x =0; x<image.Width;x++)
11     {
12         float Y = bufferY[y * image.Width + x];
13         float U = bufferUV[(y/2) * (image.Width) + (x/2)*2];
14         float V = bufferUV[(y/2) * (image.Width) + (x/2)*2+1];
15
16         c.r= Y + (1.370705f * (V-128f));
17         c.g= Y - (0.698001f * (V-128f)) - (0.33633f*(U-128f));
18         c.b= Y + (1.732446f * (U-128f));
19
20         c.r /= 255.0f;
21         c.g /= 255.0f;
22         c.b /= 255.0f;
23
24         if (c.r < 0.0f) c.r = 0.0f;
25         if (c.g < 0.0f) c.g = 0.0f;
26         if (c.b < 0.0f) c.b = 0.0f;
27
28         if (c.r > 1.0f) c.r = 1.0f;
29         if (c.g > 1.0f) c.g = 1.0f;
30         if (c.b > 1.0f) c.b = 1.0f;
31         tex.SetPixel(image.Width-1-x, y, c);
32     }
33 }
```

As described in the function, the buffer of the two vectors (Y and

UV) are instantiated using the ratio of 4:2:0, then two loops along the image width and height is performed and the three planes are divided as shown in the photo (3.9), then the actual conversion is performed. This conversion is defined "full range" because it produces values comprehended in the all RGB interval $[0, 255]$ and beyond, hence the results are normalized in the interval of $[0, 1]$. The last operation is performed since, being the chromatic scale of the YUV larger than the RGB, some values could be under 0 or over 1 after the normalization. Eventually the pixels are applied to the Texture that will represent the image.

3.6.5 The Object Panel

As already mentioned, on the reconstruction windows is possible to manipulate the object. This function was developed for users who want to try to change the configuration of the 3d models in the virtual room and to add some dynamical trait to the application. In the reconstruction windows, when the screen is touched, a line is virtually casted from the point of the screen to the virtual environment in the scene. If this ray hit an object, is searched its index value and it

is passed to the ObjectHandler script that will associate the control panel to it. Exploiting the Unity possibilities, using two sliders, it is possible to rotate and scale the object. In this panel it is also possible to change the target object to manipulate by simply tapping the other objects in the scene. A different mechanism is applied when the user wants to move the object. If the user is in the control panel of a particular object and hit the button to move the object, the function will lock the object until the button is pressed again and will disable the two sliders for scaling and rotation. When the object is locked, the user can freely move the object by dragging it in the scene or by touching the area where he wants to move it. It is important to remark that the object is not allowed to be moved in sections where the AR plane is not present.

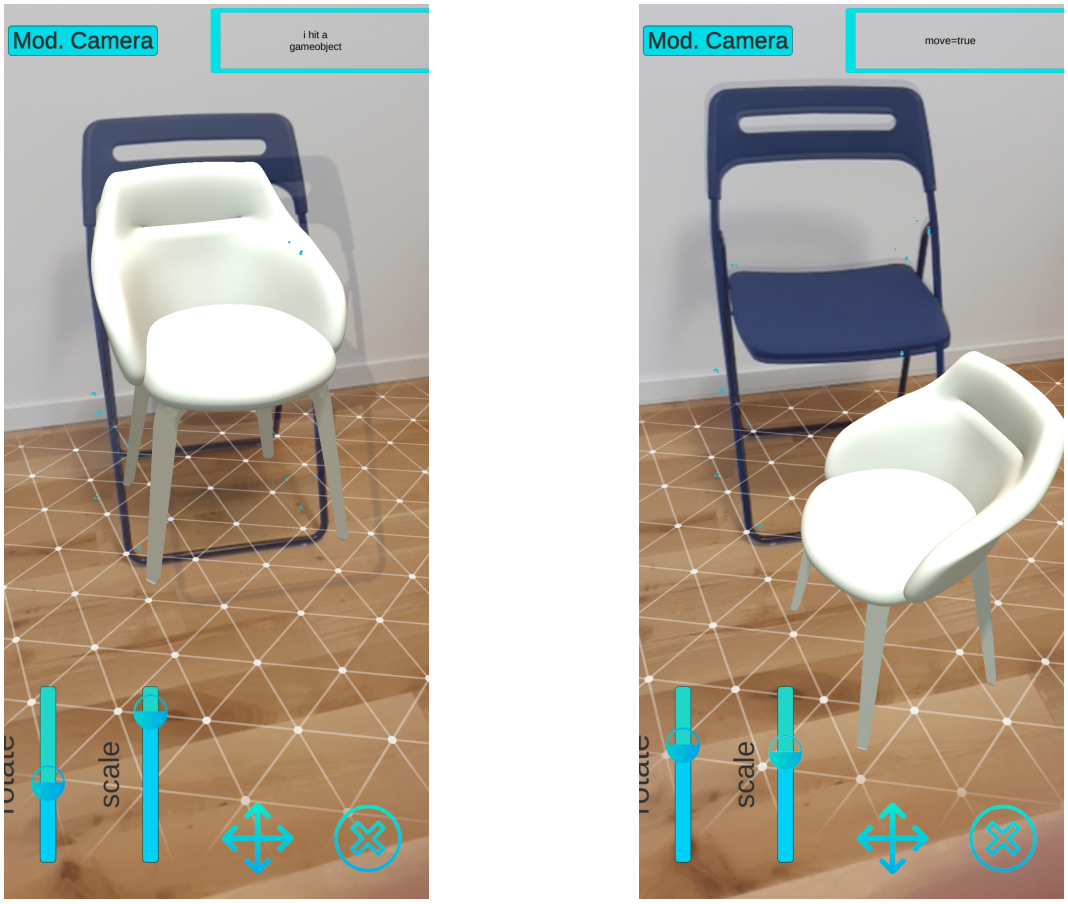

Figure 3.11: Showcase of the manipulation panel of an object before manipulation(left) and after manipulation(right). The 3D model was used to perform preliminary tests of the applications.

3.6.6 The Model definition

An aspect to highlight is that the model received by the server and reconstructed by the “Objreceived()” function is not prepared to be instantiated, and must be treated before it can be actually shown in the

scene. When the model is retrieved, a function, called ObjConstructor has the task to complete the model adding all parts needed.

```
1       public void objcontructor(receiver rec)//receiver rec
2           {
3               GameObject GO = rec.g; //is the gameobject
4               int s = rec.index;
5               int rotation = rec.rot;
6               GameObject ex = Instantiate(GO); //this is only ...
                    the object;
7               BoxCollider boxCollider = ...
                    ex.AddComponent<BoxCollider>();
8               boxCollider.size = new Vector3(60,60,60);
9               boxCollider.center = new Vector3(0,40,0);
10              GameObject e = Instantiate(new ...
                    GameObject(),ex.transform.position, ...
                    ex.transform.rotation);  //this is the container
11              e.transform.rotation = new ...
                    Quaternion.Euler(0,rot,0)
12              e.transform.position = ex.transform.position;   ...
                    //same position as gameobject
13              ex.transform.parent = e.transform;              ...
                    //object is now parent of the container
14              GameObject ShadowQuad = Instantiate(SQuad);     ...
                    //instantiate the shadowquad
15              ShadowQuad.transform.parent = e.transform;      ...
```

```
                //shadowquad is now parent of the container
16              ShadowQuadHelper shadowQuadHelper = ...
                    e.AddComponent<ShadowQuadHelper>();
17              e.transform.position = Anc[s].transform.position;
18              e.transform.rotation = Anc[s].transform.rotation;
19              e.transform.tag =``example'';
20              e.transform.parent = Anc[s].transform;
21              OBJs.Add(e);
```

In the code above, the function takes as input a structure called receiver, that will be discussed in the section[4.7]. In particular the gameobject is instantiated and a box collider is attached to it. A function called Shadowquad to cast the shadows in the scenes to improve the augmented reality experience, and the correspondent gameobject are instantiated and attached to an empty gameobject, which will store also the model. The final structure referred to as "e" in the code, contains the model with the boxcollider, the Shadowquad script and the object Shadowquad. This gameobject is finally translated to the position of the correspondent anchor using the index contained in the receiver.

3.7 The Model Decoding

The model is received from the server as part of a json structured message, which contains various information along with the model. After the complete reception of the message, the entire model consists of a single string whose length depends on the dimension and complexity of the object modelled by the BSP-Net. The actual reconstruction of the model is performed using a Unity plugin called ObjImporter whose function is to reconstruct the meshes of a model contained in a text file. Normally, a wavefrom obj model consists of a list of information that are characterized by a letter at the start of the line (i.e. ”v” for vertices, “f” for faces, “vn” for vertex normal and so on), indicating the type of the info given in that line, followed by the actual info, thus is important for an .obj file to have its information correctly disposed and splitted in different lines. As the model is received, and stored in a single string, it is not possible to have lines, even though the newline characters are inserted where needed. The first step in order to successfully reconstruct the object, is to write a temporary text file with all the object information stored in the Android application files. The path of the file is stored and called for the actual recon-

struction and, after the object instantiation, is deleted from the phone to avoid unnecessary memory usage. The ObjImporter act as a dictionary, reading the file, and storing together the information under the same initial letter, then, Unity provides a mesh constructor for simple mesh, which is used by the program to create, little by little, the entire mesh. Finally, as the material file, which is the file containing info about texture and Unity material, is not provided because no material is created by BSP-Net, it creates a default material and apply it to the model, to be visible in the scene. The model is then prepared to be instantiated in the scene as described in section 3.6.6.

3.7.1 Data Transfer Protocol

The exchange of data between Client and Server is the crucial part of the application and is performed with the utilization of the sockets. The Client, being programmed in Unity, forces the handling of data and the sender/receiver structure in C-Sharp. The connection, the reception and all the operations performed to encapsulate and decapsulate data are handled by a script called TCPConnection, whose function is to establish a communication stream with the server, knowing

a priori its IP and port number and creating a socket. The connection is established automatically when the application starts and it is closed when the client closes it. The client has the task to provide to the server two types of information, which consist in the bytes of the photo and the index of the object. 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 passes the information to the TCPhandler script, in order to incapsulate those 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 a static structure to allow the correct read and extraction of them and then converted into a string. Eventually a stream is opened and the data are sent.

```
1  public class sender //class to send photo
2          {
3              public int index;
4              public byte[] photo;
5              public sender(int ind,byte[] ph)
6              {
7                  index = ind;
8                  photo = ph;
9              }
10         }
```

The class sent from Client to Server

After the server receives the data and elaborates them, it returns another JSON string which contains various types of data. Those data are decapsulated and decrypted by the client, which already knows the structure and the organization of them. The operation of reception is performed by reading the stream, storing them in a byte array and calling a function called “Decapsulating()”. This function takes as input the byte array and performs an ASCII decapsulation to retrieve the corresponding string. In order to be sure that the message is complete, a read operation is performed, to check if the final characters of a JSON message, which corresponds to the closing brace,

is present. The message is then stored in a structured class directly matching with the JSON format. A text file containing the lines of the model are stored in a temporary folder of the user device, while the other three information are passed as global variables in the main script along with a Boolean value set to true. This value, called "isobjarrived", has the task to inform the main program that an object is ready for the reconstruction and is set to False by default. When the TCPHandler script changes the variable to True, the program calls the function "ObjArrived()" which is responsible for the model reconstruction function defined in section 3.7.

The data received contain three types of information: The first is the 3D model, which is basically the model destructured in its basic components, an index, that match the indicator index, defining where it will have to be instantiated and a value of rotation that represents the difference of the degree of the object with respect to the camera when the photo was taken. This value has the purpose to instantiate the model with the same rotation as the real object in the scene.

```
1  public class receiver //class to retrieve info from server
2          {
3              public string model;
4              pubic string name;
5              public int index;
6              public float rotation;
7              public receiver(int ind, string mod, string n, ...
                   int rot)
8              {
9                  model = mod;
10                 name = n;
11                 index = ind;
12                 rotation = rot;
13             }
14         }
```

The Class received from the Server

3.8 Reconstruction Side

3.8.1 Server Design and organization

The server was developed in a parallel thesis work title “Automatic reconstruction of indoor environments for the sharing of AR and VR spaces” The Server is constituted by a program on a PC and manages

the various phases of the object reconstruction from the reception of the image, to the actual reconstruction of the model. The entire process was programmed using the Python language, which is the most used language for machine learning and neural networks applications. The Server is composed by different blocks that have each a specific task to perform, in order to generate all information needed by the client. The entire process starts from the decapsulation of the Client information and finishes with the actual dispatch of the information needed by it. The problem was of a complex nature and in order to simplify it, it was decided to decompose the main task in little subtasks, which we will refer to as modules. The modules have specific sections to manage and are organized as a pipeline, where the output of the preceding one is necessary for the operation of the successive, but each one, performs its task in an independent manner. The composed pipeline is organized this way, and it is composed to have a total of seven modules, which are:

1. Semantic Segmentation

2. Mask Extraction

3. Object Reconstruction

4. Remeshing

5. Rendering Generation

6. Rotation Estimation

7. Object Rotation

The basic functioning of the system can be seen in the fig.3.12

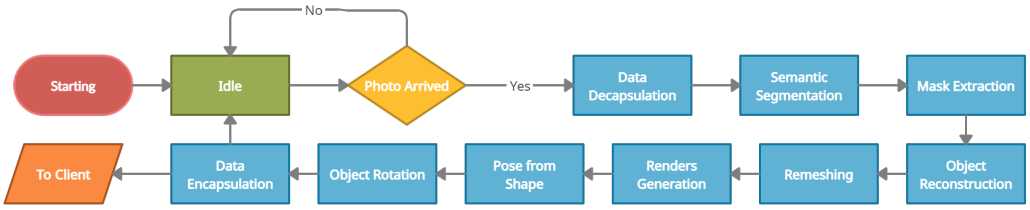

Figure 3.12: The flowchart of the pipeline of the server.

3.8.2 Semantic Segmentation

The semantic segmentation is the starting point of the pipeline, where the image retrieved from the client is first passed. Its behaviour is based on the Mseg paper [18] and it has been adapted in order to be compatible with Windows System, given the fact that its original working environment was supposed to be a Linux-based one. The main goal of this module is to perform a semantic segmentation, that

consists in the recognition and classification of the various objects in a specific photo by aggregating each pixel in the corresponding class of what they represent. The input of the system is constituted by the photo of the object that we intend to reconstruct, given 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 assigns a label to each one of them. The process is capable to detect and distinguish different types of objects such as chairs, tables, monitors, floor, walls and so on. Each class of object recognized and labeled, are defined by the specific coloring applied on them. The final process returns a colored mask which fits with the pixels of the specific objects. This process is performed in order to distinguish and to indicate to the next module which object is the actual target for the reconstruction.

Figure 3.13: The segmentation produced by the Mseg system, confronted with other programs.

This method was modified according to the pipeline requirement by removing those labels, since the mask extraction (the next module of the pipeline) is based only on the centered object of the scene and ignores all the other parts; according to this principle there is no need to classify the objects of the scene and therefore the classification is redundant and has been removed. Also, it has been assigned a grayscale 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.

Figure 3.14: The original photo (left) and its grayscale segmentation with the modified MSEG (right).

3.8.3 Mask Extraction

Now that the objects inside the picture have been detected and a corresponding grayscale value has been assigned to each of them, the one which is centered in the frame can be extracted. With a sampling mechanism is possible to define the centered point and its neighbor-

hood and this will be considered as a subset of pixels belonging to the current object that we want to reconstruct. Given the fact that these subsets have been labelled with a certain color in grayscale by the semantic segmentation, we will consider all the pixels of the same color as pixels of the object in consideration, using a flood-fill algorithm that visits recursively the pixel's neighborhood and checks which ones have the same gray value of the first one.

At the end of the process we obtain a new file which contains a set of gray pixels on white background, which consists in the binary mask of the object that we want to extract. The mask extracted will be finally applied to the real photo, multiplying the original image with the binary mask through a "bitwise-and" operation which preserves only the main object and deletes the background, obtaining an inverted-color image that, was "bitwise-not" processed, with a clear view of the extracted object (example in fig. 4.7).

3.8.4 Object Reconstruction

This procedure is the central and most important step in the pipeline, which has the goal to effectively build the mesh of the object shown

Figure 3.15: The binary mask of a chair (left) and the result of the result obtained applying the mask to the original photo (right).

in the mask passed from the precedent module. The algorithm utilized was described and implemented in the [5] and is called BSP-Net, whose inner workings were explained in 2.5.3.

Among the different system tested, The BSP-Net resulted as the best one in terms of computational time and quality of the mesh produced and is based on an unsupervised network model since during the training it does not need of any convex shape decomposition. The BSP-Net algorithm requires certain constrains on the input that are taken into

account in order to produce an acceptable result, such as a picture with a high quality (normally the photo passed from the client has a resolution of 1080x1920), and with the main object relatively close, so as it occupies from 40 to 70 percent of the image and is centered in the scene. The procedure retrieves convexes in an autonomous manner by the structure of planes and exploits it to build a BSP-tree. The overall procedure happens to produce cases in which the output mesh is messy due to the fact that the object is too complex, but in the vast majority of times the output is successfully reconstructed with a mesh that is more than passable. The output mesh is represented by a structure containing arrays with faces, edges and vertices and it is saved as a Polygon File Format (.ply).

Figure 3.16: The reconstructed object using BSP-net.

3.8.5 Remeshing

The obtained model will not be flawless, hence it is important to fix it in order to fill some voids or align some intersection between faces and vertices of the mesh that can be incorrect due to some critical issues derived from different factors like the model orientation or the luminosity of the environment when the photo was taken. The model undergoes through a remeshing algorithm to improve his mesh struc-

ture over its entire surface. This procedure is performed using different algorithms performed by Blender, which is a modelling software and it is particularly suitable for this type of operation. Before the remeshing, the object is converted from a ".ply" format, to a ".obj" format which is easier to handle and accepted by the Unity environment on the client. The Blender script is responsible for the format conversion of the model, as well of the operation of "smoothing" on vertices, the remeshing and the alignment of the faces of the model.

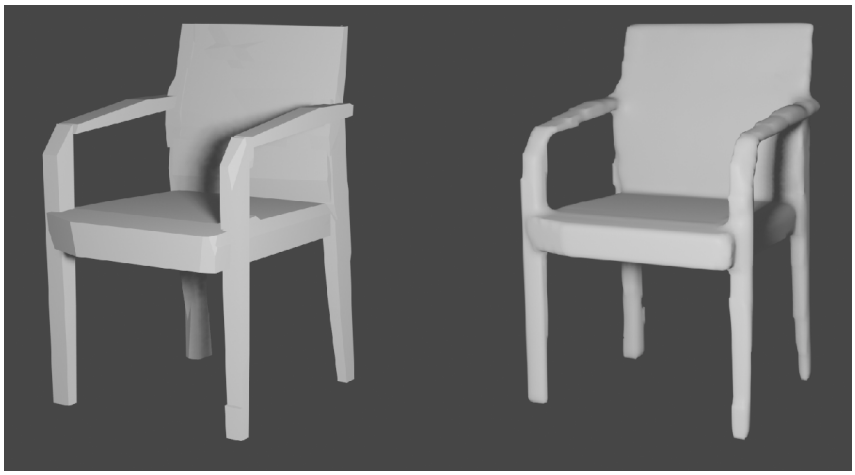

Figure 3.17: The difference between an object subjected to a remeshing operation before (on the left) and after (on the right).

3.8.6 Rendering Generation

The rendering Generation is necessary for the rotation estimation module to work. As already mentioned, the model instantiated in the scene, will not have a precise indication of its orientation, therefore these two additional steps perform some operations on the object to find the relative rotation between the object orientation in the photo and the camera. In order to perform this, an algorithm called Pose From Shape is employed, and conveniently splitted between this and the next module. This module has the goal to generate of an appropriate number of renders of the 3D model with the usage of Blender. The model is uploaded in Blender and set in a space surrounded by a fixed source of light and with a camera which rotates around the object, taking shots at every iteration. After this first iteration, the camera rotates completely around the vertical axes, standing parallel to the ground of the Blender virtual environment. Then another complete rotation around the model is performed, taking the snaps with a slight slope of the camera, in order to change the point of view of the rendered images. Due to the expensive computational load graving on the GPU and CPU during the rendering of the images, to avoid

creating a bottleneck in the system, the iteration and the slope of the camera are adjusted to optimize the rendering generation of the single frames. The number of image rendered in this step is around two hundred and are stored in a folder, ready to be utilized in the next step.

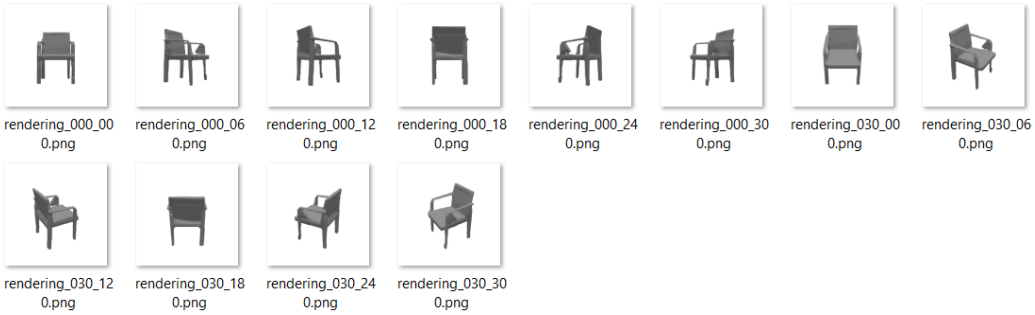

Figure 3.18: The Result of rendering generation in a Blender environment.

3.8.7 Rotation Estimation

While the previous module was a preparation of the data needed, this module is the actual implementation of a slight adaption of the code PoseFromShape from [34] This adaption was performed to be optimized and integrated in the pipeline. The main idea is to find the most similar pose of the object in the photo, trying to deduct it from the different images of different orientation of the corresponding model.

The model takes as input the folder of rendered images obtained in the previous module and its accuracy depends on the number of different possible orientations given to it, which in other words means that more images are fed to it the more accurate prediction it will have as a result. After the algorithm find the image which best fits among the others, it returns 3 parameters: azimuth, elevation and in-plane rotation, where only the azimuth value will be employed in order to instantiate the 3D model with the appropriate orientation.

3.8.8 Object Rotation

As final step, the remeshed object is rotated by 90 degrees along the x-axis in order to map the coordinates from the Blender environment to the Unity one and thus reconstructing the object with the correct orientation in the scene. These steps are crucial for the application as the model instantiated on the detected floor by Google ARCore, which defines the touching point between the object and the plane using the model axis as reference, would be placed in the wrong way with its forward axis pointing downward. Now that the object reference axes were been aligned and the rotation has been performed the 3D model,

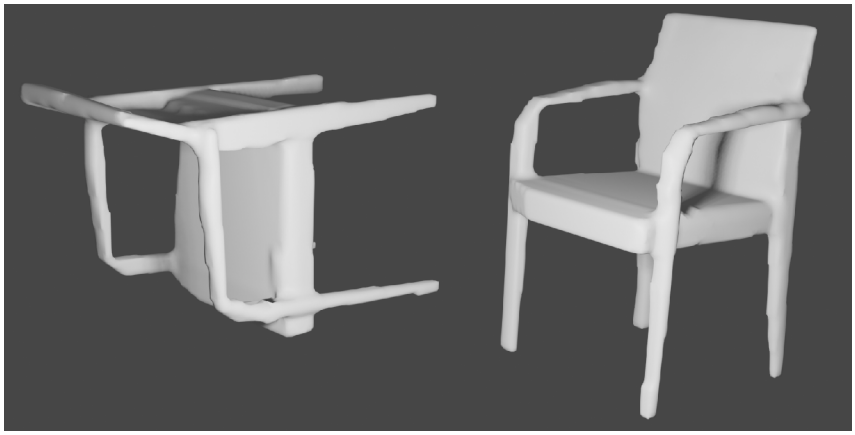

Figure 3.19: The model before(left) and after(right) the rotation of 90 degree on its x axis.

it can be sent from the “Reconstruction Side” to the “Acquisition and Placement Side” which finally will accomplish 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.

3.8.9 Server Connection

The server is a standard Python server which establishes a local connection over a specified port with the C-Sharp client. The entire process starts when it accepts the connection request from the client, starting 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 "{ }", it is possible to check them in order to check if the message is completely received. 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.

4 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 Lamps, Tables, Chairs, Cars and Planes. Since the system was designed to works on indoor environment, 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 4.1: 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.

Figure 4.2: Twelve reconstructions of different models. In the photo is possible to see two column with each the starting photo, the mask of the object and the 3D model after remeshing.

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.4.3. These problems lead to the creation of a wrong 3D model which represents the attempt of reconstruction extracted in the mask. Therefore, it is interesting to point out the results produced by the different errors that can affect the pipeline.

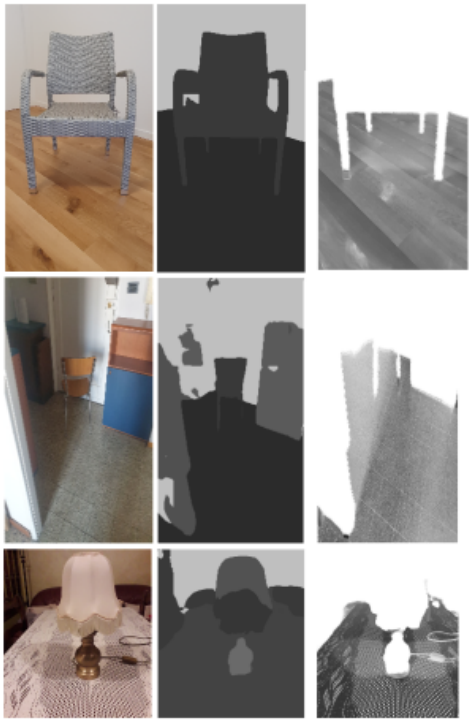

Figure 4.3: Extraction errors due to having the object not being at the perfect center of the scene.

The first step to be taken into consideration is the semantic segmentation, being also the starting step of the process, which partition the entire photo, defining its components with a specific level of gray. This step is crucial, as 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, as it was established that depends on the GPU installed in 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 results in the saturation of the memory. There are several possible errors caused by the segmentation step, which could be the missed detection of the hole present between the different parts of the object, or the exact opposite, that is when the segmented photo contains holes not present in the real object. Another possible mistake could be the aggregation of different objects with the main one, transforming entirely the structure of the model. The example of those errors are shown in Fig.4.4

Figure 4.4: Different types of Mseg error: on top, the erroneous recognition of a hole, in the center the loss of recognition of the holes and at the bottom the wrong definition of object 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 allow an acceptable

reconstruction. The network was trained using photos of synthetic model(rendering of digitalized 3D models, not photos of real objects) with low resolution(128x128), probably due to the fact that the training phases were quite expensive in terms of computational time and small images can lead to comparable results saving a huge amount of time that could be wasted training the neural network. 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 was trained.

Figure 4.5: Image reconstructed in normal condition(left) and with the adding of padding around it.

As a matter of fact, BSP-NET build the model starting from a two-axis point of view and try to extract the third dimension from it, thus the capability of infer the missing dimension is greatly affected by the orientation of the object in the photo. After a couple of test, was clear how the best reconstructions occurred when the object photographed was rotated of an angle varying from 30°to 45°respect its y-axis, as shown in Fig.4.6.

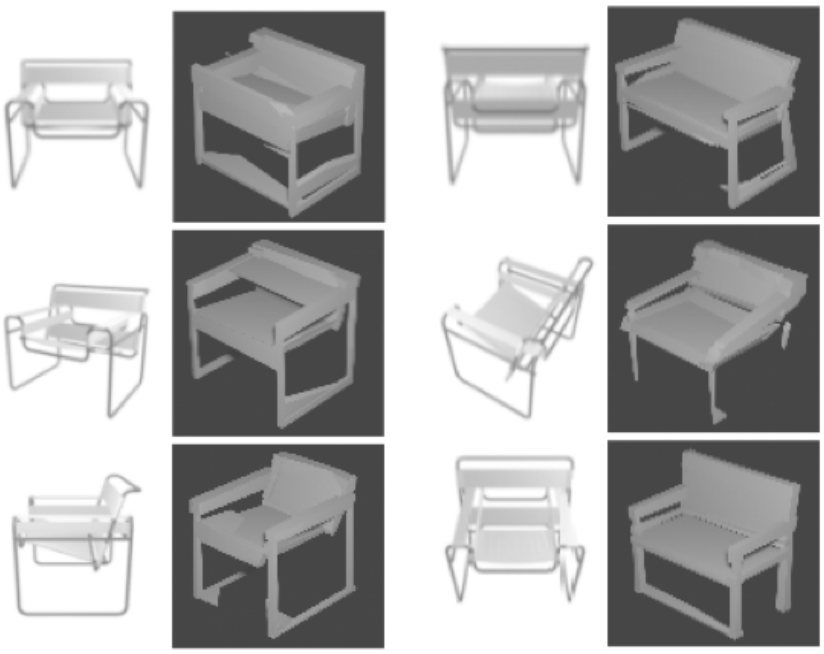

Figure 4.6: Reconstructions of the same object starting from different photo angulation.

Another aspect of the network is the fact that, having been trained using “perfect models”, the differences in luminosity given by the envi-

ronment can alter the object and its reconstruction, producing models of real object less accurate respect the synthetic one. This problem causes the disruption of the mesh, sometimes adding pieces that are not present in the photo or with a slight different structure.

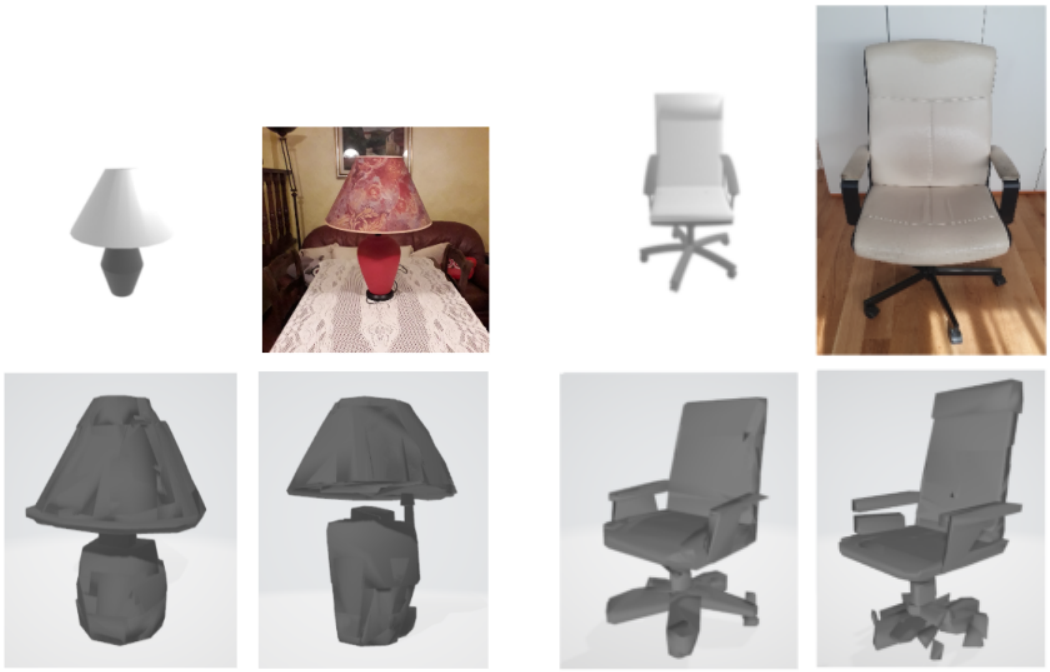

Figure 4.7: Comparison between two similar object reconstructed using photos of synthetic and real object. We can denote as some details are missing, incomplete or defined in a wrong manner.

The next issue detected concerning this network is caused by the wrong perception of the object dimension, where the actual dimension of the object are not fully deductible in the photo and thus are not respected. This phenomenon causes the network to change the

object reconstruction by lowering or increasing its dimension or even change the actual type of the object, relying on the shape of the object used for its training.

Figure 4.8: On the top: Reconstruction error caused by wrong perception that change the type of the object (from chair to armchair).On the bottom: Reconstruction error caused by erroneous perception of the object dimension(the back of the chair is more elongated than it should be).

As completion of the overview we also present cases in which the BSP-NET is not capable to reconstruct the model from the photo due to the lack of information or the higher complexity in the object shape.

This results in the effort of the network to define a similar object resorting to the information stored during the training phase, showing as the system is slightly biased and sometimes limited to the models already known and not fully capable to extract the proper shape of the object.

Figure 4.9: Cases when the object is wrongly reconstructed.

Another interesting step of the pipeline that was taken into consideration was the remeshing, which consists in an operation of smoothing

applied to the original mesh to produce clearer result. Although this procedure is not capable to alter 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 were not present in the original one, resulting in a degradation of the overall outcome.

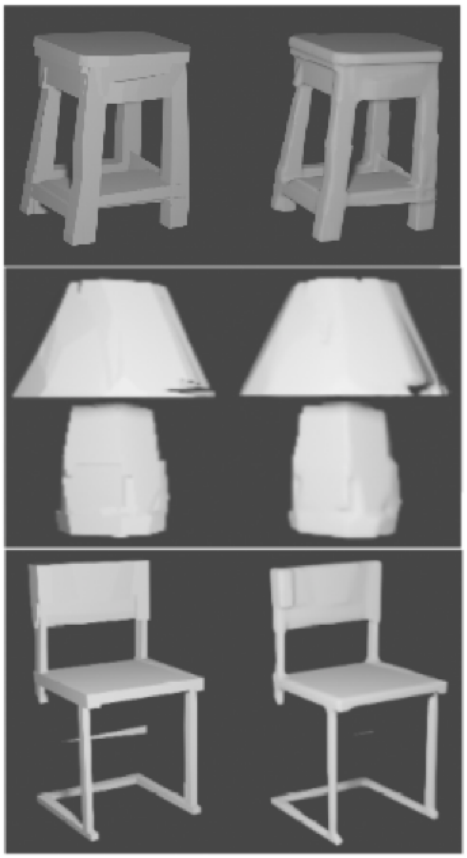

Figure 4.10: Comparison between the original mesh of the model(on the left) and the object mesh after the remeshing(on the right).

Finally, it was taken into consideration how the application works and respond considering the possible causes of failure and strain. 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 perform the detection of the floor without, operation which are performad without any fail. The first problem arise when the application is active for several minutes, considering that, during this time, the floor detection is constantly active and continue to search for floor, which eventually led to an overpopulation of vertical and horizontal plane, sometimes not existent. The large number of floor constitutes an impediment to the utilization of the app, in that, 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 before the point of interest or also preventing the model to be moved due to the collision of the planes around him.

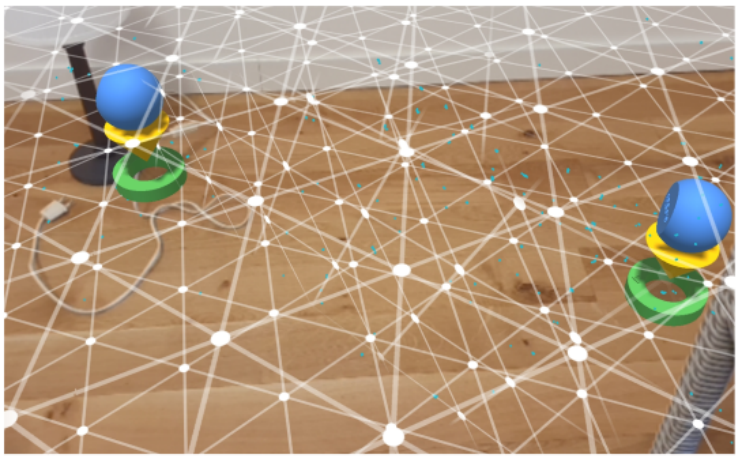

Figure 4.11: Multiple intersecting planes created by the AR Core session .

For what regard the reconstruction of the model in the scene, it freezes the application for less that a second, only for the first time 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 in particular, it can happen that the model have its scale marginally different respect the size of the real object, on account of the fact that the 3D model size is defined using an average value between the different dimensions of the object tested.

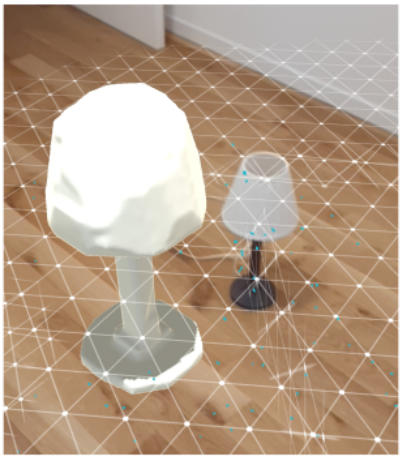

Figure 4.12: The real object compared to the model with an erroneous scaling applied.

As the application was 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 presented examples of scene reconstruction using different objects, positions and configurations.

Figure 4.13: scene reconstructed using two different objects(on the left) and five different object(on the right). The top photo on the left and the on on the bottom on the right show the position of the placeholders in the environment.

Figure 4.14: Scene reconstructed using multiple instance of the same object.

5 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 pipeline reconstruction in order to sum up the possible implementation that could further improve the entire system described in this essay 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, making clear that the application performs quite well when it faces one of these objects.

According to the goal task of the project, in order to achieve better results, a way is 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, was not possible to access to required resources in terms of hardware and

computational capability. In fact to successfully achieve an improved training, the needed component should be quite powerful, since it affect the computing cost in terms of time which exponentially grow using older hardware; for instance, using an NVIDIA geForce RTX 2080 Ti GPU to trains the model, it will require more than 5 days non-stop 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 benchmark. Therefore, with a high quality hardware and a considerable amount of time for training, it should be interesting 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 led to the possibility to represent the output shapes through difference operations rather than a union of different parts, bringing to a wider generalization approach that can express complex and concave or convex details. For what regard 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 was evinced by the results how the principal cause of a low quality reconstruction is due to 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 make the network acquire the capability to segment more complex objects. Another improvement can be performed on the remeshing part, which algorithm could be upgraded to detect and fill the empty gap that sometimes are present in some models due to an error of reconstruction(i.e. a chair with a back with a hole in the middle, which was not present) and eliminate unnecessary parts which are not connected to the main mesh. At last, 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 depht between the object and the other parts of the environment, achieving a better result in terms of segmentation.

Finally, is possible to debate about the Application, which is the interaction point between the user and the reconstruction pipeline. As mentioned, it is possible to take photo with the smartphone camera and the resolution of it was set to 1080x1920 to be successfully resized and cropped by the server. If the BSP-Net quality enhancing mentioned before could be applied, it could be also possible to higher the photo quality to the actual camera resolution of most phones, which can be around or over 4K, providing a large number of detail helpful for a realistic model reconstruction.

Another problem of the actual application is given by the scaling size of the model instantiated, which is defined as an average value between the different sizes of the object tested and deployed in the scene, thus, could be helpful to develop a function to automatically create a bounding box around the object whose photo is taken in order to be able to adjust the scale of the model which will overlap it.

For what regard the model instantiated in the scene, there are little

amelioration that can be applied in order to increase the experience of the user in the application. One of these could consist in the introduction of a menu to change the appearance of a specific model from a set of texture available, as now the only texture automatically applied is a default white material, which is a Unity standard for model which not have linked any specific material to it. Concerning the manipulation of the object, which allow a selected object the possibility to move, rotate and scale it, occurs that, in the circumstance in which different number of object reconstructed are present, could be difficult to recognize the actual object subjected to the manipulation, hence, could be interesting to introduce an indicator or marker of some sort, which univocally point or highlight the object which is being manipulated.
Another improvement regards the Gallery, which not provide anything beside the image visualization of the photo taken by the app, then it should be possible to provide additional functionalities by adding the possibility to delete a specific photo, eliminating along the indicator and model linked to it using its index, and a function to save the images in the internal storage of the smartphone.
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 reconstruct it, and moving towards to 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, decompose each one of the from the background, reconstructs each one of them independently and sends them back to the client. At lasts, as the application has not a specific purpose, and it could be possible to implement a system to export the entire scene currently visualized in Augmented Reality, in order to transfer the position, dimension and rotation of all object inside it in a different environment, as a PC, in order to be visualized and modified.

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] *Displaying video colors correctly*. URL: https://blogs.gnome.org/rbultje/2016/11/02/displaying-video-colors-correctly/.

[11] 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.

[12] *Estrazione immagini in bianco e nero dal formato NV21 della fotocamera Android*. URL: https://www.it-swarm.jp.net/ja/android/android%E3%82%AB%E3%83%A1%E3%83%A9%E3%81%AEnv21%E5%BD%A2%E5%BC%8F%E3%81%8B%E3%82%89%E7%99%BD%E9%BB%92%E7%94%BB%E5%83%8F%E3%82%92%E6%8A%BD%E5%87%BA%E3%81%99%E3%82%8B/971495912/.

[13] *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/.

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

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

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

[17] 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.

[18] John Lambert et al. "MSeg: A Composite Dataset for Multi-Domain Semantic Segmentation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 2876–2885. DOI: 10.1109/CVPR42600.2020.00295. URL:

https://doi.org/10.1109/CVPR42600.2020.00295.

[19] *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/.

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

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

[22] 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.

[23] R. A. Newcombe et al. "KinectFusion: Real-time dense surface mapping and tracking". In: *2011 10th IEEE International Sym-*

posium on Mixed and Augmented Reality. 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[24] 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.

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

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

[27] 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.

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

[29] 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.

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

[31] 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.

[32] 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.

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

[34] Yang Xiao et al. "Pose from Shape: Deep Pose Estimation for Arbitrary 3D Objects". In: *CoRR* abs/1906.05105 (2019). arXiv: 1906.05105. URL: http://arxiv.org/abs/1906.05105.