3D Reconstruction from Silhouettes and Photoconsistency based on Depth Map Fusion

Cots Pujol, Helena
Curs 2018-2019

Director: GLORIA HARO
GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS
3D Reconstruction from Silhouettes and Photoconsistency based on Depth Map Fusion

Helena Cots Pujol

BACHELOR THESIS UPF / YEAR 2019

SUPERVISOR
Dr. Gloria Haro
Image Processing and Computer Vision Research Group
A la meva família,
Acknowledgements

First of all, I would like to express my gratitude to my supervisor Dr. Gloria Haro for all her support and guidance throughout the elaboration of this project and for introducing me to the field of 3D vision. Her help has been of great importance and it has been a pleasure to work with her.

I am grateful to the Image Processing and Computer Vision Group researchers and teachers at Pompeu Fabra University, they made me enjoy every lesson in class.

Finally, I would like to thank my family for giving me the opportunity to study what I like and support me during these four difficult and intense years.
Abstract

In this thesis, we present a multi-view 3D reconstruction system from silhouettes and photoconsistency information based on depth map fusion. The method receives as input a set of images from different points of view with their corresponding calibration parameters. A confidence volume is extracted from the silhouette information. From this initial volume, photometric consistency is computed for every voxel inside the volume with respect to every view in the dataset using DAISY descriptors. Then, a set of depth maps is built for every view and they are merged via Truncated Signed Distance Function. Finally, the Marching Cubes algorithm is used to extract a polygonal mesh of the 3D implicit surface.

Resum

En aquesta tesi, presentem un sistema de reconstrucció 3D multi-vista a partir de siluetes i informació de fotoconsistència basada en la fusió de mapes de profunditat. El mètode rep com a entrada un conjunt d’imatges des de diferents punts de vista amb els seus corresponents paràmetres de calibratge. S’extreu un volum de confiança a partir de les siluetes. A partir d’aquest volum inicial, es calcula la consistència fotomètrica per a cada voxel dins del volum respecte a cada vista mitjançant els descriptors DAISY. A continuació, es construeix un conjunt de mapes de profunditat per a cada vista i els fusionem a través de la Funció de Distància Signada Truncada (TSDF). Finalment, l’algorisme de Marching Cubes s’utilitza per extreure una malla poligonal de la superfície implícita en 3D.
Contents

List of figures xii
List of tables xiii

1 INTRODUCTION 1
1.1 Multi-View Stereo Reconstruction and Related Work 2
1.2 Method Overview 10
1.3 Dataset 12
1.4 The Pinhole Camera Model 13
1.5 Parallelization 16

2 DEPTH MAP ESTIMATION 17
2.1 Confidence Volume 17
2.2 Photoconsistencies 21
2.2.1 DAISY Descriptors 22
2.2.2 Photoconsistency Measure 24
2.3 Depth Prediction 25
2.3.1 Computation of rays 25
2.3.2 Depth Images 27
2.3.3 Bilateral Filtering 28
2.4 Image Artifacts 32

3 SHAPE ESTIMATION 35
3.1 Spatial Integration 35
3.2 Shape Mesh Generation 41
3.2.1 Marching Cubes Algorithm 41
3.2.2 Extracting the Mesh 44
3.2.3 Visualizing our result 45

4 EXPERIMENTS 47
4.1 Photoconsistency Measure Modifications 47
4.2 Depth Maps Parameters ........................................... 51
4.3 TSDF Modifications .................................................. 52
4.4 Confidence volume resolutions ................................. 55
4.5 Runtimes ............................................................... 57

5 CONCLUSIONS ......................................................... 59
List of Figures

1.1 Multi-view stereo pipeline [Furukawa and Hernández, 2013]................................. 4
1.2 Occlusion problem. [Furukawa and Hernández, 2013] .......................... 6
1.3 MVS four popular output representations, from [Furukawa and Hernández, 2013] .......................................................... 9
1.4 MVS algorithms from [Furukawa and Hernández, 2013] .......................... 10
1.5 Opencv [Corporation et al., 2019] ................................................................. 12
1.6 Frame examples from different cameras .................................................. 13
1.7 Pinhole camera model [Hartley and Zisserman, 2003] ............................ 14
1.8 Image and camera coordinate systems ..................................................... 15
1.9 World and camera coordinate frames ....................................................... 15
1.10 Runtime comparison on CPU [Beyer and Jeff Larkin, 2016] .................. 16

2.1 Voxel volume example. [Seitz and Dyer, 1997] ............................................ 18
2.2 Parameters of the volume where V is defined ........................................ 19
2.3 Confidence volume at different resolutions (number of voxels) ............... 21
2.4 DAISY descriptor. Image from [Tola et al., 2009] ....................................... 23
2.5 Three different rays from the same camera view ...................................... 27
2.6 (a) represents a 100-gray-level step perturbed with a Gaussian noise with \( \sigma = 10 \) gray levels. (b) represents the weights for a 23x23 neighborhood centered two pixels to the right of the step in (a). (c) shows the result of applying a bilateral filter with \( \sigma_r = 50 \) gray levels and \( \sigma_d = 5 \) pixels to (a). Image from [Tomasi and Manduchi, 1998]. ..................................................... 31
2.7 Depth maps before and after filtering, \( voxelsize = 0.01 \). ....................... 32
2.8 Artifact example in a depth image ............................................................ 32
2.9 Color discrepancies in the same object part ........................................... 33
2.10 Raw depth maps with different volume resolutions .................................. 34

3.1 Unweighted signed distance functions in 3D, from [Curless and Levoy, 1996]. ................................................................. 37
3.2 Generation of 3D distance fields from range images [Zach et al., 2007].

3.3 (a) An example of a 2D TSDF with the object surface in green.
(b) 1D TSDF sampled along the ray through p with t = 1000 mm.
Object surface is at zero crossing, from [Werner et al., 2014].

3.4 Cube Numbering [Lorensen and Cline, 1987].

3.5 14 Intersection Patterns [Lorensen and Cline, 1987].

3.6 VTK Library logo [Kitware, 2016].

3.7 MeshLab Logo. [Cignoni et al., 2008]

3.8 Meshes extracted from the TSDF fusion with $TD_{iso} = 0.1$.

4.1 TSDF iso-surface; $TD_{iso} = 0$, $\tau_{photo} = 0.4$.

4.2 Comparison between VH and $TD$ figure waist.

4.3 Two different photoconsistency measures for $TD_{iso} = 0$.

4.4 Fusion of depth maps created with two different values of $\tau_{photo}$.

4.5 $TD$ computed in two different ways.

4.6 Amount of reconstructed figure at $TD_{iso} = 0$ depending on $\mu$ parameter.

4.7 $TD_{iso} = 0.1$.

4.8 $TD_{iso} = 0$.

4.9 Mesh with TSDF iso-surface extracted at 0-value.
## List of Tables

2.1 DAISY descriptor parameters. ........................................ 23  
2.2 Depth predictions parameters. ...................................... 28  
2.3 Bilateral filter parameters. ........................................... 31  
4.1 Mean vs. Median photoconsistency measures in two voxels. ... 49  
4.2 Runtimes depending on voxel sizes and for each one of the steps. 57
Chapter 1

INTRODUCTION

A long time ago, being able to represent in a 2D image a real 3D scene was a technological breakthrough. Nowadays, new technological challenges are emerging and there is a strong interest in succeeding the other way around: from 2D views recover their corresponding 3D information. This problem has been present in the Computer Vision field for the last decades, and there are already methods good enough to succeed. The applications emerging from these technologies are increasing as more improvements are made, such as 3D computer graphics and video games, virtual and augmented reality, 3D printing, city modelling or art to name a few of them.

The origin of this project lies in one of my deepest passions: dancing. When I was younger, I interacted with games as Just Dance [Ubisoft, 2019] all the time, where you only get to follow a 2D flat figure of someone dancing. An idea came to my mind: what if you could learn choreographies as if you were in the same real scene? You could see the choreographer from the point of view that is best for you.

My first aim was to perform a 4D reconstruction, that is, get a visualization of the figure reconstructed in 3D at every time instant of the video. In the end, we have built a method that performs the 3D reconstruction of the figure in a static way for the first time instant but that is expandable to 4D. To carry out this reconstruction algorithm we based our method on Leroy, Franco and Boyer’s work [Leroy et al., 2017].
1.1 Multi-View Stereo Reconstruction and Related Work

In this first section our goal is to introduce the reader to the multi-view reconstruction concept explaining what is it and do a brief review of works related to this area. In the end, we want to have a global vision of the algorithms and methods that are currently used to compare them with those chosen in this thesis.

The problem of reconstructing 3D geometry from a set of images has been a challenge in Computer Vision field for at least three decades, but there have been improvements and new methodologies every year since then. There is a large range of applications benefited from these techniques, such as 3D mapping and navigation to e-commerce, city modelling, computer video games and movies, 3D object recognition, or cultural heritage archival, among others. However, only recently, these techniques have been achieved to be good enough to leave laboratory-controlled environment and still provide robustness, precision and scalability on large-scales.

We focus our interest in the challenging task of modeling the 3D geometry of real objects, as it is this thesis goal. The first differentiation of techniques we encounter is between active and passive image-based methods. The first ones, imply actively or directly interaction with the scene [Jarvis, 1983]. A few good examples of active 3D reconstruction methods widely used in the present can be LIDAR [Tse et al., 2008] [LIDAR, 2019], Kinect [S. Izadi and Fitzgibbon, 2011] [Microsoft, 2010], the use of RGB-D cameras for 3D mapping [Endres et al., 2014] or other types of scanning [Levoy et al., 2000] for many purposes, but we focus on passive ones.

Passive image-based methods

The principal characteristic of passive methods is that they do not interact directly with the environment but that they use conventional image sensors to capture the data. These methods can also be distinguished in three different categories: monocular, binocular and multi-view. The first one is the simplest to gather data but the hardest one to recover depth information from it. In monocular passive methods the data consists of a set of images taken from a single point of view with a fixed camera. In a biological comparison, monocular vision is the vision in which both our eyes are used separately. Then, the estimation of depth from a single monocular image is a hard task and it requires taking into account the global structure of the image. In [Saxena et al., 2005] they use monocular cues such as texture variations and gradients, occlusion, object relative sizes or defocus to estimate depth. On the other hand, binocular and multi-view stereo (MVS) methods
use, respectively, two or multiple views taken from different view-points. Binocular is the simplest case based on stereo correspondences, where two corresponding pixels from the two images represent the same point in the 3D space. MVS methods are an improvement of the two-frame stereo correspondence algorithms [Scharstein and Szeliski, 2002], it consists on taking more than two photographs from multiple view-points. If the number of images captured from in-between view-points is high, the robustness of the algorithms is increased [Strecha et al., 2008].

To sum up this first introduction, we can define a 3D image-based reconstruction as the most likely 3D shape representing a given set of photographs of the 3D scene, under several assumptions such as materials, viewpoints and lighting conditions. The MVS algorithms explained in this section consider the same input: a set of images and their corresponding camera parameters. An MVS algorithm is only as good as the quality of its input data, and the performance of the method depends on how robust are the approaches used in the algorithm.

In [Seitz et al., 2006] there is a wide comparison among several multi-view stereo reconstruction algorithms although new methods have appeared since then. They are compared on a database of images with a ground truth result (available online in http://vision.middlebury.edu/mview [Seitz et al., 2018]). It is difficult to compare and evaluate MVS algorithms because they vary significantly according to their assumptions, operating ranges and behavior. At the end, some of the most successful MVS algorithms are presented. The basic pipeline of an MVS algorithm is:

1. Collection of images (from multi-view points).
2. Compute camera parameters for each image.
3. Reconstruct 3D geometry of the scene from the images and their parameters.
4. Reconstruct materials of the scene (optional).
The MVS methods receive as data input images from calibrated cameras where the camera matrices of each view are known (see Section 1.5 from this Chapter). In the case of MVS where the camera parameters are known, the estimation of the 3D geometry of the scene is equivalent to solve the correspondence problem across the input images (such as Optical Flow problems [Baker et al., 2011]). These correspondences are also denominated as candidate matches, and the possible measures to find a likely candidate match are explored from now on.

There are two main approaches to solve the 3D reconstruction of an object, a figure or a scene: algorithms based entirely on images or those that make use of silhouettes [Mülayim et al., 2003, Zheng et al., 2009]. This last approach reconstructs a rough 3D shape from binary images which is usually used as a starting solution for other methods based on images [Franco and Boyer, 2003, Furukawa and Ponce, 2006] or even in conjunction with them [Cremers and Kolev, 2011]. On the other hand, the methods based on images are those that use, generally, measures of photographic consistency. Initial proposals for 3D reconstruction used meshes based on visual hull (VH) [Baumgart, 1974]. However, VH is not an accurate approximation or not suitable enough for scenes with many concave structures. In our method we use this technique since the authors [Leroy et al., 2017] consider that it helps to obtain a robust algorithm. In the following, we revise methods that use photoconsistencies, which are to this day the ones that have produced the best results.
Multi-view photo-consistency

Multi-view photometric consistency is the main information used in any MVS algorithm. It measures the consistency among the different projections of a 3D point in the available views. In detail, the photoconsistency of a pair of pixels in different views is measured as the similarity of some image features extracted at the pixels of interest (see Section 2.2.2 from Chapter 2). A lot of research has been done to improve and innovate those algorithms and their photo-consistency measures. We are going to talk about a few of the best or more currently used measurements.

In order to compute photo-consistency on a set of images, it is required that these images see the same 3D geometry. There are different photo-consistency measures, such as Sum of Squared Differences (SSD) or Sum of Absolute Differences (SAD), Normalized Cross-Correlation (NCC), Census transform, Rank filters, and others. Among these measures, zero-mean NCC is one of the most successful techniques used in MVS algorithms, but it is hard to work on color images. Both NCC and Census transform [Zabih and Woodfill, 1994] are two of the best performers due to the invariance to illumination changes. Some conclusions have been extracted on what the proper choices within this first set of measures would be: SAD and SSD are extremely fast to compute and easily adapted to be used in GPUs. As a result, to have a complete and robust measure, some algorithms combine SAD with one of both techniques NCC and Census transform. Other photo-consistency techniques involve using gradient descriptors such as SIFT [Lowe, 2004], GLOH or DAISY [Tola et al., 2008] to contribute to the measure using essential features of images as other measures cannot achieve. In section 2.2 of Chapter 2, we describe why the use of gradient descriptors is better for large-scale images and why we use it on our method.

Nevertheless, the above mentioned measurements are suitable when an approximation of the 3D shape is available (e.g. visual hull). Most of the cases where, for example, the amount of input images is large, we do not know beforehand what each one of the images sees since the 3D geometry itself is unknown. Different strategies have been explored to maximize photo-consistency through images despite the lack of true knowledge of their content.

In the early 2000s, work was done on the so-called space carving research, which was key to developing a visibility consistency theory [Kutulakos and Seitz, 2000, Furukawa and Ponce, 2006]. In the work presented by [Seitz and Dyer, 1997, Seitz and Dyer, 1999], they introduce a new methodology that avoids the problem of correspondences between images by working in a discrete space formed by voxels, which are traversed in a fixed visibility ordering. This new strategy can solve the occlusion problem (see Figure 1.2) by identifying a set of invariant voxels
which form a spatial and photometric reconstruction of the scene, and more importantly, fully consistent with the input images by evaluating them all from both inward and outward facing cameras.

Figure 1.2: Occlusion problem. [Furukawa and Hernández, 2013]

In [Kutulakos and Seitz, 2000] they also use a 3D volume partitioned into a 3D grid of voxels which is carved out to remove those voxels that are not photo-consistent.

In recent MVS methods, visibility is managed in a different way from the original work of space-carving. The now state-of-the-art MVS algorithms are capable of handling millions of images, and the estimation of such visibility is a critical step in order to achieve computational efficiency, so photo-consistency is calculated with a small number of images for each location. The process of visibility estimation for datasets with such large amount of images is usually divided in two phases, using clustering on the large amount of images to reduce the large-scale MVS problem into a sequence of small problems and then, perform a more fine visibility estimation over these sub-problems.

In large-scale scenes, the techniques that have been proved more suitable are the ones that use depth map fusion [Salman and Yvinec, 2009]. Thus, depth maps values for the input images can be computed from photo-consistency information and, if desired, combined with from silhouette extraction. In the following section we talk about the advantages of implementing this methodology into the MVS algorithms and the state-of-the-art works.
Depth maps on MVS algorithms

The use of depth maps reconstruction is one of the most popular choices due to flexibility and scalability of the technique. In the last years, there has also been improvements in the methodologies for the reconstruction of the depth maps. The best way to get started is to see the basis of these algorithms, and then we will mention some that are more innovative.

In Figure 1.4 there is a step before the depth map reconstruction named “Depth range initialization”, which can vary depending on the algorithm to use. In the method implemented, this step consists in computing the distance of voxels within the estimated 3D volume or visual hull (see Section 2.3.1 from Chapter 2). Then, the basic procedure is to evaluate photo-consistency values throughout this depth range (our rays) and, broadly speaking, choose the depth value with the highest photoconsistency value for every pixel in an image. This approach is proved to work well, demonstrated by [Hernández and Schmitt, 2004], and it is the same one that we follow from [Leroy et al., 2017] and used in our thesis. Hernández and Schmitt combine texture and silhouette information in a robust way from a set of calibrated color images and apply the research for proper depth values technique that we just mentioned. As a consequence, they are able to recover and reconstruct both 3D geometry and texture.

Since then, some improvements have been made and more robust photoconsistency depth maps algorithms have been developed. In 2007, Hernández and others [Vogiatzis et al., 2007] proposed a volumetric formulation that uses graph-cuts and a robust photoconsistency metric based on NCC to deal with occlusions. This new methodology can overcome the challenges that simpler algorithms could not achieve. On the other hand, Goesele, Curless and M. Seitz [Goesele et al., 2006] developed an algorithm that computes depth maps using a window-based approach, using a certain condition to return only good depth values. Such condition takes a certain threshold and discard the photoconsistency scores that fall under it.

Despite these improvements, sometimes the highest photoconsistency value does not correspond to the correct depth value when the scenes are complex. To solve this problem and other ones, some techniques based their work on Markov Random Field depth maps formulation [Kolmogorov and Zabih, 2001], which is quite popular for solving other Computer Vision problems. In 2008, some of the researchers mentioned above extended this MRF formulation in their algorithm [Campbell et al., 2008] to improve depth maps results. They use local maxima from photoconsistency curves for each pixel and, using MRF, the depth corresponding to this local maxima is assigned to the correspondent pixel.
3D output current representation techniques

Another way to distinguish between MVS algorithms is to look for which type of output representation they use. Here we provide a classification of the most popular output scene representations that are used by recent MVS algorithms that reconstruct the scene 3D geometry from photoconsistency metrics. Furukawa and Hernández [Furukawa and Hernández, 2013] define that the four most popular representations are:

- **Depth maps.** As seen before, photoconsistency information can be used to create depth maps for every image in the input data. Then, the output representation can be constructed by merging all the depth maps to see the fully geometry of the scene based on depth map representation. Again in 2010, Campbell, Vogiatzis, Hernández and Cipolla proposed an algorithm to improve the depth maps quality used in current MVS techniques [Campbell et al., 2008] by removing their outliers.

- **Point cloud.** Sometimes point cloud representations are very similar to a textured mesh model due to its dense appearance, but it is only a dense set of independent 3D colored points. This kind of representation can be seen at Furukawa and Ponce work [Furukawa and Ponce, 2010], who proposed a patch-based approach to reconstruct and represent a surface. They also propose a method to turn the patch model result into a mesh.

- **Volume scalar-field.** Also understood as a volumetric depth map fusion technique as in the work by Curless and Leroy [Curless and Levoy, 1996] using signed distance functions (SDF) for that purpose. A SDF is computed for every depth map labelling the negative results as inside the scene or surface and the positive as outside. Then, the weighted average is performed and stored in a 3D scalar field. The final surface can be extracted as the iso-surface at zero value of the scalar field using Marching Cubes algorithm [Lorensen and Cline, 1987]. However, depth maps on opposite sides interfere each other in the function field. Consequently, Hernández, Vogiatzis and Cipolla try to solve this effect using a probabilistic evidence in their work [Hernández et al., 2007] and the result is seen in Figure 1.3.

- **Mesh.** Many MVS algorithms represent their final outputs with meshes. The most suitable methods to use this genre of representation are the ones that use volume carving approaches due to the high quality output meshes. Two common algorithms used to obtain these meshes are Marching Cubes algorithm [Lorensen and Cline, 1987] or marching tetrahedron algorithm. Moreover, the resolution of the final mesh can be fixed by the volume grid resolution, e.g. the voxel size. Hernández and Schmitt, in the same work we
used to talk about silhouette information [Hernández and Schmitt, 2004], use an octree-based carving method with a posterior marching tetrahedron meshing algorithm. In their work they test the algorithm on several datasets and using different output representations.

In Figure 1.3, there is a reconstruction example for each of the four representations explained above, performed by the state-of-the-art MVS algorithms cited at each type of output’s explanation.

![Diagram of MVS output representations](image)

Figure 1.3: MVS four popular output representations, from [Furukawa and Hernández, 2013].

Most MVS algorithms follow a single path to reconstruct 3D geometry, while others combine steps to create more robust algorithms. The following diagram (Figure 1.4) shows the pipeline that most MVS follow, although there are always exceptions. One of these exceptions would be an approach that reconstructs a mesh directly from the volume of photoconsistency using the volumetric fusion [Vogiatzis et al., 2007]. Regarding the approach that we follow in our thesis, it is marked in color and summarizes the basic structure of our pipeline.
1.2 Method Overview

In this thesis we follow the work by Leroy, Franco and Boyer [Leroy et al., 2017], “Multi-View Dynamic Shape Refinement Using Local Temporal Integration”. This method is about spatio-temporal reconstruction via depth maps fusion with Truncated Signed Distance Fields [Werner et al., 2014]. Our work uses the spatial reconstruction part of their method in order to perform the figure recovery in a similar way without taking into account the temporal domain. We also use a volumetric representation of the surface within a 3D volume formed by what are called voxels. Although we follow the steps in this work, we add or modify
particular approaches for our particular case.

In this work we explain the theory behind the reconstruction steps but also how the implementation is done, as it is a struggle we faced during the various steps of the method. Here, we present an overview of the principal steps performed to accomplish the goal:

- **Collecting the images.** From a dataset of multi-view images we store them in a way that are separated by cameras. In this work, we do not resize the images because we do not want to loose any detail. If the speed was prioritized before the result, we should reduce the size of the images, since processing large images requires more time. In this process the camera parameters associated to each one of the views are computed or extracted in any form.

- **Projecting the figure.** Using the camera parameters associated to the views from our dataset, we can get the 3D space where the figure is represented by those parameters.

- **Discrete volume.** A discretized volume by voxels is found to delimit the 3D scene extracted before in order to create an initial visual hull of the 3D figure that needs to be reconstructed. We want to adjust this volume as much as possible to the figure to save computation time in further steps. Explained in section 2.1 from Chapter 2.

- **Depth maps estimation.** We follow step by step the proposed method from the paper we mentioned and we add our own modifications. Every view of our dataset is transformed into a depth map of the real figure, more or less precise, that will be used in reconstruction. We also experimented with different manners to create these depth maps to choose the most suitable one. Depth maps creation is explained in Chapter 2 and the experiments in Chapter 4.

- **Spatial integration.** Also denominated depth maps fusion as it is where all the depth information for every view is put in common in a particular way. In this case, we use Truncated Signed Distance Fields [Werner et al., 2014] to fuse depth values according to the photoconsistency information for every voxel in the visual hull. This is the last step of the reconstruction algorithm and we get an iso-surface ready to transform to a mesh. This spatial reconstruction is detailed in Chapter 3.

- **Mesh extraction.** At the end, we visualize the 3D iso-surface from the depth maps fusion extracting it as a polygonal mesh. To do so, we use the implementation of the Marching Cubes algorithm in the C++ open-source
library VTK [Kitware, 2016]. Automatically, the output mesh is in Polygon File format (PLY).

In order to implement most of the steps explained above, we take advantage of the tools that C++ open-source library OpenCV provides [Itseez, 2019].

![OpenCV](image.png)

Figure 1.5: Opencv [Corporation et al., 2019].

### 1.3 Dataset

The collection of images and data to treat as the input to the multi-view stereo setup is relevant in order to define the posterior steps of the method. According to [Furukawa and Hernández, 2013], there are three main types of imagery collection used to MVS setups:

1. Get image from a laboratory setting.
2. Small-scale scene captures in an outdoor environment.
3. Large-scale scene captures in an outdoor environment.

First MVS setups started with laboratory image collections, [Faugeras and Keriven, 1998], due to the facility to control light conditions and camera calibration, i.e. using static different points of camera or robotic arms. Eventually, as improvements on MVS algorithms were developed, researchers also moved to a small-scale outdoor scenes, [Hornung and Kobbelt, 2006, Habbecke and Kobbelt, 2007, Vogiatzis et al., 2007], such as a part of a building, a bench or a fountain and then, even more challenging, large-scale scenes [Furukawa et al., 2010, Vu et al., 2009] as an entire building, an airplane or a city.

These significant changes were not only due to the developments and advances in the field of MVS. It was the result of a combination of factors like new hardware to capture better images, with more power of calculation that allowed to use a great number of images and, moreover, improvements on algorithms of estimation of scalable cameras (like Structure-from-Motion).

The first idea was to capture my own images to use them in this project. For the time being, we decided to focus on the implementation and leave it for the
last. Unfortunately, there was not enough time to do it but it will remain as future work.

In this project, then, the collection of images and computing camera parameters for each image is not necessary as it is provided by the dataset “girl dance” sequence we get from the INRIA 4D repository [INRIA, 2017], corresponding to a woman dancing in a room. This dataset is formed by the video frames of eight cameras from different points of view used for the recording, the silhouettes corresponding to each one of all the frames and the projective camera matrix $P$ of each view (see Section 1.5). Below there is an example of what the frames of the video from different points of view look like.

![Frame examples from different cameras.](image)

Figure 1.6: Frame examples from different cameras.

### 1.4 The Pinhole Camera Model

In the previous section, it was mentioned that the dataset provides the corresponding camera matrices $P$ for each one of the cameras. It is important to understand what this $P$ means and how it is used in the reconstruction.

“A camera is a mapping between the 3D world (object space) and a 2D image” [Hartley and Zisserman, 2003]. Once the 2D image collection is properly set up, the next step in every multi-view system pipeline consists in estimating the camera parameters associated to its corresponding images. The simplest camera model is the pinhole camera model (Figure 1.7) which defines a 3x4 matrix. This matrix, frequently denoted as $P$, is known as the camera projection matrix which is used to project points in the 3D world to a pixel location in the 2D image plane.

$$\mathbf{x} = P\mathbf{X},$$

where $\mathbf{X}$ is a point in the 3D space and $\mathbf{x}$ a pixel in the image plane, both expressed in homogeneous coordinates [Karlsson, 2006] [Jia, 2018]. Matrix $P$ can be decomposed into the product of two other matrices: the camera calibration
matrix, $K$, and the matrix of extrinsic parameters, $[R|t]$: 

$$P = K_{3\times3}[R_{3\times3}|t_{3\times1}] = \begin{pmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_y \end{pmatrix}.$$  

(1.2)

$K$ is the *calibration matrix* containing the intrinsic parameters of the camera: being $(f_x, f_y)$ the vertical and horizontal focal lengths, $(p_x, p_y)$ the coordinates of the principal point $p$ in Figure 1.7 – where the principal axis meets the image plane – and $s$ a skew factor, usually zero. Matrices $R$ and $t$ contain the external parameters of the camera: its position and orientation with respect to the reference system of the world.

![Figure 1.7: Pinhole camera model [Hartley and Zisserman, 2003].](image)

The center of projection is denoted as *camera center* or *optical center*. The focal length, expressed in pixel units, is the distance between this center and the image plane or *focal plane*. When digital images have squared pixels, it is assumed that $f_x = f_y$. The *principal axis* or *principal ray* is defined as the line from the camera center perpendicular to the image plane.

Under this camera model, a point in world space with coordinates $\tilde{X} = (X, Y, Z)^T$ is mapped to a point on the image plane $\tilde{x} = (x, y)^T$ where the line that joins $\tilde{X}$ and the camera center meets the image plane, as shown in Figure 1.7. In practice, it may not be assumed that the origin of coordinates in the image plane is at the principal point, see Figure 1.8. If the reference system of the camera coincides with the reference system of the world space, it can be seen that:

$$x = f_x \frac{X}{Z} + p_x, \quad y = f_y \frac{Y}{Z} + p_y.$$  

(1.3)
which describes the transformation from 3D world coordinates to 2D image coordinates of the point $X$.

![Image of Image and camera coordinate systems.](image)

Figure 1.8: Image and camera coordinate systems.

However, the assumption that the reference system of the camera coincides with the reference system of the world used in equation (1.3) is not always the case. Instead, the information about the position of the camera with respect to the reference system of the world is extracted from the extrinsics matrix $[R|t]$, where $R$ is the rotation matrix and $t$ the translation vector, from which it can be extracted the orientation and position of the camera. See Figure 1.9.

![Image of World and camera coordinate frames.](image)

Figure 1.9: World and camera coordinate frames.

To recapitulate, matrix $P$ is defined by the instrinsics and extrinsics parameters of the camera and it is used to transform the world coordinates to image coordinates. In this thesis, the dataset provides us these camera projection matrices directly. Matrix $P$ can be used as well to recover the camera center or origin $C$. Since pinhole camera model belongs to the finite projective camera case, it can be assumed that $PC = 0$ and then, $C$ is the null space of $P$. 

15
To extract the position of the camera, the singular value decomposition (SVD) is computed over the projective camera matrix $P$. This means that $C$ is the last column of $V$ in the SVD decomposition of $P = UDV^T$.

1.5 Parallelization

The computational cost is very high due to all the mathematical calculations. Consequently, the runtime of the program is too long. In order to speed up the code, we used OpenMP [OpenMP, 2018] to parallelize some processes. This parallelization is done along the 8 threads of the CPU despite that the first try was to do it in GPU. After a lot of research in NVIDIA-GPU with OpenMP and unsatisfactory implementation tests, we decided not to waste more time and use the CPU. As it can be seen in Figure 1.10 from [Beyer and Jeff Larkin, 2016], using 8 threads instead of one we can accelerate the process 3.52 times.

![Figure 1.10: Runtime comparison on CPU [Beyer and Jeff Larkin, 2016].](image)

Later in this document, we will examine that, despite some processes are accelerated, the problem of not obtaining immediate results makes it difficult to find the optimal parameters for the method.
Chapter 2

DEPTH MAP ESTIMATION

The first step of this method consists in creating depth maps for the input images of the videos. During this step, the time dimension will be dropped out temporarily given that the depth image is built independently per frame. The principal scheme could be described as the search for the best potential depth candidate along a viewing ray of any image pixel with respect to a photoconsistency measure. Within all the existing techniques, it is chosen to use a confidence volume based on the silhouette information. According to [Leroy et al., 2017], this method increases precision and reduces false positives. The following structure should be followed:

1. Extract the confidence volume $V$.
2. Compute DAISY descriptors.
3. Estimate pairwise photometric discrepancy and photoconsistency measure.
4. Compute viewing rays.
5. Build depth maps images.
6. Apply a bilateral filter to improve depth maps.

2.1 Confidence Volume

In order to start computing the steps mentioned above, it is necessary to prepare properly the data. It is assumed that a set of $N$ images $\{I_i\}_{i=1}^N$ observed with a set $S$ of calibrated cameras with known camera projection matrices $P$ is given. Camera matrices are used to compute projections $\{\pi_i\}_{i=1}^N$ and camera centers $\{c_i\}_{i=1}^N$, in
the manner in which it is explained in the first chapter Section 1.5. And finally, a set of silhouettes \( \{ \Omega_i \}_{i=1}^N \), which may not be completely precise.

Before deepen in the method itself, it is wise to first know how the confidence volume looks like. In this particular case, a discretized volume of cubic voxels is used, whose related parameters will be discussed later. In Figure 2.1 it is shown a 3D volume of cubic voxels seen from above surrounded by cameras in different points of view. This kind of framework is similar to the same in this method.

![Figure 2.1: Voxel volume example. [Seitz and Dyer, 1997]](image)

The main difficulty of using a discretized volume is the loss of precision of the figure to reconstruct. In order to increase the resolution, the number of voxels used to represent the volume shall increase too, but it also rises the execution time and memory.

In our case, the set of silhouettes \( \{ \Omega_i \}_{i=1}^N \) provided define a confidence volume to construct a 3D visual hull which is assumed to contain the observed object, i.e. the girl. The visual hull can be seen as the intersection of all the possible cones projected from every camera or view containing the object. In order to find this object in the space and following what [Leroy et al., 2017] proposed, the confidence volume \( V \) is defined as:

\[
V = \{ x \in \mathbb{R}^3 : \exists^{\geq \alpha} i (\pi_i(x) \in I_i) \land \exists^{\geq \beta} i (\pi_i(x) \in \Omega_i) \},
\]

where \( x \in \mathbb{R}^3 \) is a voxel and it projects to \( \alpha \) images or more and to \( \beta \) silhouettes or more. The parameters \( \alpha \) and \( \beta \) are two constants defined by the user in order to restrict poor depth predictions. With respect to the visual hull, this volume \( V \) is a dilated version of what it would be in the space region formed by the set of
images $\alpha$. Depending on how good the silhouettes are extracted from the images, it is a preferred choice to take into account fewer silhouettes than images.

In order to build this confidence volume $V$, a set of parameters has to be defined. First of all, we define the size of the volume in voxel coordinates (different from the world space ones), the offset where this volume stands in the world space and the size of the cubic voxels inside the volume. In Figure 2.2 these parameters are showed.

![Figure 2.2: Parameters of the volume where $V$ is defined.](image)

$(Nx,Ny,Nz)$ determine the amount of voxels in the coordinates $x,y,z$ respectively as well as the size of the starting volume. These values are defined in voxel coordinates, not in the world space framework. On the other hand, the offset $(Ox,Oy,Oz)$ is defined in world space coordinates. Finally, the voxel size ($vs$ in the Figure) that is key to seek for precision.

At the end of the algorithm below, the result is a 3D volume of voxels with sizes $Nx$, $Ny$, $Nz$, where the voxels belonging to $V$ are set to 1, and also an array filled with the indexes of these voxels. From now on, the mentioned array will be used to compare the voxels in further sections and algorithms. A general idea of how these two data structures are computed can be summarized below:
Algorithm 1: Compute the Confidence Volume

Data: Images, silhouettes, matrices P and volume parameters.

Result: Confidence Volume and array of voxel indexes.

parameters initialization: \( N_x, N_y, N_z, \text{voxelsize}, \alpha, \beta, \text{offset coordinates}; \)

for each voxel in volume do
    Transform voxel coordinates into world coordinates:
    \[
    \text{worldcoord} = \text{voxelcoord} \times \text{voxelsize} + \text{offsetcoord} + \text{voxelsize}/2;
    \]
    Convert to homogenous coordinates:
    \[
    \text{homogcoord} = (\text{worldcoord}, 1);
    \]
    for first frame in each camera view do
        Compute voxel projections with camera projections matrices P:
        \[
        \text{projectioncoord} = P \times \text{homogcoord};
        \]
        Transform to 2D cartesian coordinates.
        if (2D coordinates projects to current image) then
            Increase image counter;
            if (2D coordinates projects to current silhouette) then
                Increase silhouette counter;
            end
        end
    end
    if (image counter \( \geq \alpha \) and silhouette counter \( \geq \beta \)) then
        Add the index of current voxel to the array.
        If we want to visualize the volume \( \rightarrow \) Set voxel to 1.
    end
end

The voxel size parameter used to compute the confidence volume has an important role for the rest of the method: the smaller the voxels we use, the more accurate we be the rest of the steps of the algorithm. As more voxels are considered, more depth information can be retrieved, which will be discussed hereafter in the document. Figure 2.3 shows a comparison with static \( N_x, N_y, N_z \) parameters and varying the voxel size from 0.1, 0.05, 0.025 to 0.01 from left to right. In this particular case, \( \alpha \) is set to the total number of cameras, 8 in our dataset, and \( \beta = \alpha - 1 \). These two values will be fixed for the rest of the method.
Figure 2.3: Confidence volume at different resolutions (number of voxels).

After testing several values, the size of voxel that begins to give an acceptable depth accuracy would be 0.013 or less. The different runtimes it takes depending on the size are detailed in Experiments, Chapter 4.

2.2 Photoconsistencies

There are many ways to compute depth maps as explained in the Introduction Chapter. In this method, we used a photoconsistency measure throughout the images and then used voxel information. In this section, we will go into detail of how the photoconsistency is computed and prepared for its posterior use on building the depth images. This process can be described in a few steps:

1. Compute image descriptors $D$ with DAISY.
2. Evaluate pairwise photometric discrepancy $g$ of voxels using the descriptors.
3. Calculate photographic consistency $\rho$ for every voxel using $g$.

In order to predict depth for every pixel through viewing rays, the photoconsistency measure is evaluated along the ray based on pairwise photometric discrepancy. To this day, the Normalized Cross Correlation function (NCC) has been widely used to evaluate similarities [Oswald et al., 2014] [Vogiatzis et al., 2007]. Recent advances made on image descriptors result in an improvement in gradient descriptors with respect to other similarities measures such as NCC, SIFT, GLOH or DAISY, some references on local descriptors can be studied on [Lowe, 2004], [Mikolajczyk and Schmid, 2003], [Mustafa et al., 2016], and for DAISY two important papers like [Tola et al., 2009] [Tola et al., 2008]. Compared with NCC,
similarity measures for gradient descriptors take into account many more features of the image that make them more robust to noise and pixel color variations.

According to what [Leroy et al., 2017] state, DAISY is preferred over SIFT and GLOH as it is said to give better results in this context, citing textually from [Tola et al., 2009]: “(...) We therefore introduce a new descriptor that retains the robustness of SIFT and GLOH and can be computed quickly at every single image pixel.”

In order to use these descriptors to compute the pairwise photometric discrepancy and the photo-consistency measure, the already implemented code from [Tola, 2009] is applied to our images. In the following subsection the DAISY descriptors are explained in more detail.

### 2.2.1 DAISY Descriptors

The first key property about DAISY is that large patches can be used with an acceptable computational cost without much performance loss due to the convolution of orientation maps to compute the bin values using Gaussian kernels. It is no longer necessary to use small patches techniques to reduce artifacts generated by occluded regions. To solve this issue, the method proposed using different masks at each location and use Expectation Maximization framework [Borman, 2006] to select the best one.

SIFT and GLOH descriptors are both computed over local regions, summarizing, they use 3D histograms which dimensions belong to image spatial dimensions and to the image gradient direction. That way, every pixel belonging to a local region contributes to the histogram with a corresponding weight. This pixel’s weight depends on three main features: its location in the region, its orientation and the norm of the image gradient at the pixel location. DAISY computes these descriptors at every pixel. In order to do that, the weighted sums of gradient in the histogram bins are replaced by convolutions of gradients in specific directions with several Gaussian filters. As a result, at each pixel location, “DAISY consists of a vector made of values from the convolved orientation maps located on concentric circles centered on the location, and where the amount of Gaussian smoothing is proportional to the radii of the circles.” [Tola et al., 2009]. The previous quote can be represented in Figure 2.4. At the end, this local descriptor provides good results even using small images for stereo reconstruction and robustness to surface orientation and warping.
For more information, DAISY descriptor is explained in [Tola et al., 2009] and comparisons between other gradient descriptors are included. In order to use the code provided by Tola in [Tola, 2009], the user has to set a few parameters. In this thesis, we do not want to deepen in DAISY descriptor, so the default values proposed by the author are used in our experiments. In the following table the parameters set by the user and used in our algorithm are specified.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius (rad)</td>
<td>Distance from the center pixel to the outermost grid point</td>
<td>15</td>
</tr>
<tr>
<td>Radius Quantization (radq)</td>
<td>Number of convolved orientation layers with different standard deviations</td>
<td>3</td>
</tr>
<tr>
<td>Angular Quantization (thq)</td>
<td>Number of histograms at a single layer</td>
<td>8</td>
</tr>
<tr>
<td>Histogram Quantization (histq)</td>
<td>Number of bins in the histogram</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2.1: DAISY descriptor parameters.

For a voxel $x$ inside the confidence volume $V$, and given two images $I_i$ and $I_j$, where $i$ and $j$ are two different cameras, the pairwise discrepancy $g_{i,j}(x)$ at this voxel is computed this way:

$$g_{i,j}(x) = (D_i(\pi_i(x)) - D_j(\pi_j(x)))^2,$$  \hspace{1cm} (2.2)
which defines the Euclidean distance between two descriptors $D_i$ and $D_j$ where $D_i(\pi_i(x))$ and $D_j(\pi_i(x))$ represent the DAISY descriptors (1D vectors of data) at the pixel location obtained by projecting the 3D point $X$ onto images $i$ and $j$. Those descriptors are vectors with a certain length, where its Euclidean distance must return a single value associated to a voxel. In order to get this value, the squared norm is computed over these two vectors.

### 2.2.2 Photometric Consistency Measure

Once the pairwise photometric discrepancy is computed for each voxel in $V$ and for every pair of views, the next step is to find the photometric consistency measure $\rho_i(x)$ for each voxel with respect to all the cameras. It is computed as a normalized robust measure of image descriptors discrepancy at $x$. The following measure is useful to consider all the discrepancy values and not only the minimum one.

$$\rho_i(x) = \sum_{j \in C_i} \omega_j W(g_{i,j}(x)),$$

(2.3)

where: $\omega_j$ are the normalized values of $\omega_j = \cos \theta_{ij}$ that weights camera contributions around camera $i$ and using the angle $\theta_{ij}$ between the centers of the cameras $i$ and $j$. This angle $\theta$ is a restrictive measure which helps to determine which cameras are suitable for contributing to the photometric consistency measure. For instance, cameras whose centers are highly separated might not coincide on photometric consistency for the same voxel as the view would be quite different, then, $C_i$ represents the subset of cameras $j$ such that $\omega_j > thr_{\theta}$:

$$C_i = \{ j \in S : \cos \theta_{ij} > thr_{\theta} \},$$

(2.4)

where $thr_{\theta}$ is set to 0.7 by [Leroy et al., 2017]. Finally, in their work they define $W()$ as a Parzen Window [Parzen, 1962]. However, in our method we use a Gaussian filter due to its simple implementation defined as:

$$W(g_{i,j}(x)) = \exp \left( \frac{-(g_{i,j}(x))^2}{2\sigma^2} \right).$$

(2.5)

We choose $\sigma$ equal to or less than 1 in order to let voxels with high photometric consistency contribute to the measure and restrict the ones with bad photometric information. The best score that the $\rho$ could achieve is 1, when all cameras in $C_i$ present the same image descriptors at the same voxel $x$, and 0 the worst.
2.3 Depth Prediction

In the previous section, a 3D volume of voxel photoconsistencies $\rho$ is created for those voxels inside the confidence volume $V$. This information is used to build the depth maps. The concept of rays is introduced to go through all the voxels that project to each one of the pixels in an image. The objective is to predict depth for each pixel in every silhouette along its ray using maxima of the photoconsistency.

2.3.1 Computation of rays

This first section explains how the rays are computed to achieve the goal mentioned in the introduction of this chapter. First of all, a Look Up Table (LUT) of voxel distances is created: for each one of the voxels inside the confidence volume $V$, we store the distance from the center of the voxel to the center of every camera $c_i$. In the pseudo-code below it is explained step by step how this distance LUT is calculated:

\begin{algorithm}
\caption{Compute LUT of voxel distances}
\begin{algorithmic}
\STATE \textbf{Data:} Camera centers and indices of voxels inside $V$.
\STATE \textbf{Result:} 3D matrix of voxel distances for every camera.
\FOR {each camera $i$}
\STATE Get the camera center $c_i$.
\FOR {each voxel $x$ in 3D volume}
\STATE Compute voxel index.
\IF {voxel is inside $V$}
\STATE Transform voxel coordinates to world coordinates $x_i$.
\STATE Compute and store distance ($\|x_i - c_i\|$) of voxel $x$ for camera $i$.
\ENDIF
\ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

Once the distances for the inside voxels are computed for every camera, we can proceed to compute another LUT of rays. The two main steps in order to compute the correspondent rays for each one of the images from different views are:

1. Create rays LUT for each image $i$ and pixel $p$.
2. Sort each pixel ray according to the distances of the voxels (distances LUT).

At the end, each one of the pixel rays will contain the indices of the voxels projecting to that pixel and its corresponding distances with respect to the correspondent
camera. To better understand how these both steps are done, their corresponding pseudo-codes are detailed below.

**Algorithm 3:** Create rays LUT for each image $i$ and pixel $p$

**Data:** Distances LUT, indices of voxels inside $V$ and pixels coord. where voxels project.

**Result:** Rays LUT without order by distances.

```plaintext
for each camera $i$ do
    for each voxel $x$ in 3D volume do
        Compute voxel index.
        if (voxel is inside $V$) then
            Get at which pixel $p$ the voxel projects to.
            Add voxel index and its distance to $raysLUT[i][p]$.
        end
    end
end
```

The pixel coordinates (row and column) corresponding to each of the projections of the inside voxels are calculated and saved previously in the method. This way, whenever these coordinates are needed throughout the method we do not need to re-calculate them, thus saving more computing time.

**Algorithm 4:** Sort each pixel ray according to the distances of the voxels

**Data:** Rays LUT without order.

**Result:** Rays LUT ordered by voxel distances.

```plaintext
for each camera $i$ do
    for each pixel $p$ in the image $i$ do
        Sort voxels of $raysLUT[i][p]$ from less to more distance.
    end
end
```

Again, the resolution the user chooses at the beginning of the algorithm will influence the rest of the method results. The rays of the pixels will be more accurate as more voxels form them. The algorithm would have more possibilities to choose the best depth candidate for the pixel. In Figure 2.5, there is an example of how our rays look like. Red, purple and yellow voxels project to 3 different pixels, these voxels and their correspondent distances are used to determine the depth of the pixels where they project.
2.3.2 Depth Images

The previous subsection provides us the proper material to build the depth maps for each one of the silhouettes from different views. As explained in the introduction of this section, the approach is to search for those voxels in rays where the photoconsistency is maximum. In order to avoid false detections of maxima far from the surface, the method stops when the accumulated photoconsistency reaches a threshold.

To express this concept in a mathematical manner we use the following formula (2.6) that defines the best depth candidate $d^p_i$ along ray $r_i(p, d)$ leaving camera $i$ through pixel $p$:

$$d^p_i = \begin{cases} 
  d_V(p) & \text{if } \max_{d \in [d_V(p), d_{\text{max}}]} \rho_i(r_i(p, d)) < \tau_{\text{photo}}, \\
  \arg\max_{d \in [d_V(p), d_{\text{max}}]} (\rho_i(r_i(p, d))) & \text{otherwise.}
\end{cases} \quad (2.6)$$

Analyzing the possibilities of the depth candidate, $d_V(p)$ is the first depth value in the ray $r_i(p, d)$ inside the confidence volume $V$, therefore the distance of the first voxel of the ray. The parameter $\tau_{\text{photo}}$ is the minimum photoconsistency value to trust the photoconsistency values along the ray, if we fall below this threshold the first distance value is chosen. Finally, a search limit $d_{\text{max}}$ is defined:
\[
\int_{x=d_{V}(p)}^{d_{\text{max}}} \rho_{i}(r_{i}(p, x)) \, dx \leq \rho_{\text{max}},
\]
(2.7)

where \(\rho_{\text{max}}\) is defined by the user. This parameter performs the function mentioned before of limiting surface penetration along rays.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_{\text{photo}})</td>
<td>[0.20, 1]</td>
</tr>
<tr>
<td>(\rho_{\text{max}})</td>
<td>big (~50)</td>
</tr>
</tbody>
</table>

Table 2.2: Depth predictions parameters.

In Table 2.2, we define the only two parameters that the user has to define for the algorithm. If the photoconsistency threshold is close to zero, it means that unless the maximum of photoconsistency along the ray is very low, the distance of the voxel where the photoconsistency is maximum is taken. Otherwise, we consider \(d_{V}(p)\) as the pixel’s depth. On the other hand, although \(\rho_{\text{max}}\) does not have much effect on the result, it helps us to define the maximum depth to be considered and, in practice, to stop the algorithm at a certain point.

In [Leroy et al., 2017], superpixel clustering on images (SLIC [Achanta et al., 2010]) is used to speed up the process of computing depth maps and to add spatial consistency. Instead of using this clustering approach, we decided to use bilateral filtering to fill those pixels without depth information and to smooth the overall final depth image. In this way, spatial and color information is also taken into account. In our case, using a bilateral filter helps when the number of voxels along rays is not sufficient to compute a reliable depth for all pixels in the silhouettes.

2.3.3 Bilateral Filtering

Bilateral filtering technique is widely used in image processing frameworks. Its main use is the reduction or removal of noise from a signal in images. Unlike other techniques with similar purposes, the bilateral filter is a nonlinear filter that performs spatial averaging while preserving the edges. This was an innovation at the time, because researches put a lot of effort to develop techniques which could help to smooth images without losing the edges. However, to take advantage of this performance, the filter parameters must be selected carefully as they affect the results significantly.
The first approach was presented by [Tomasi and Manduchi, 1998]. The bilateral filter depends on spatial distance and intensity difference. They state that two pixels can be closer if they occupy nearby spatial locations or similar if they have nearby values, i.e. intensity values that perceptually are similar. In this paper they define range filtering, which averages image values with weights that decay with dissimilarity. This kind of filter is called nonlinear because their weights depend on image intensity or color.

To define this concept, a general function describing the bilateral filter applied to a pixel \( x \) of an image \( I \) is defined as:

\[
I_{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) G_{\sigma_r}(\|I(x_i) - I(x)\|) G_{\sigma_s}(\|x_i - x\|),
\]

where \( \frac{1}{W_p} \) is a normalization factor:

\[
W_p = \sum_{x_i \in \Omega} G_{\sigma_r}(\|I(x_i) - I(x)\|) G_{\sigma_s}(\|x_i - x\|).
\]

\( G_{\sigma_r} \) or \( \text{intensity kernel} \) is a Gaussian kernel that gives more weight to pixels having similar intensity. \( G_{\sigma_s} \) or \( \text{spatial kernel} \) is a Gaussian kernel that gives more weight to pixels spatially close in the image. \( \Omega \) is the window centered at \( x \), so that \( x_i \) is another pixel inside this window. In our case, we want to perform an intra-image reconstruction using bilateral filtering on the raw built depth images using information of their correspondent RGB original images. There are two main reasons to do that: first of all, because it may happen that not all pixels of the silhouette have depth information because of the volume discretization; the other one, that this depth information may not be completely reliable only from voxel distances and photoconsistencies. Our purpose is to improve pixels depth averaging depth information using the bilateral filtering with RGB image as reference and smooth the final image. Equations (2.8) and (2.9) can be written more specifically to our case:

\[
w(i, j, k, l) = \exp \left( -\frac{(i - k)^2 + (j - l)^2}{2 \sigma_r^2} - \frac{\|I_{\text{RGB}}(i, j) - I_{\text{RGB}}(k, l)\|^2}{2 \sigma_r^2} \right),
\]

(2.10)
and after calculating the weights:

\[
I_F(i, j) = \frac{\sum_{k,l} I_{\text{depth}}(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)},
\]

(2.11)

where \(I_F\) is the filtered depth of pixel \((i,j)\). The weight function \((2.10)\) is computed using color information from the original images. In this example, \((i,j)\) are the coordinates of the pixel that needs to be filtered using its neighbouring pixels, and \((k,l)\) the coordinates of a neighbouring pixel inside the window \(\Omega\). \(I_{\text{RGB}}\) of a pixel gets the color value and intensity of the three (RGB) dimensions of the image. These values can have two different intensity ranges: \([0,1]\) or \([0,255]\), in our images the second range is used so that \(\sigma_r\) can have values in the same intensity range. Parameters \(\sigma_r\) and \(\sigma_s\) are the smoothing parameters, which characterize the bilateral filtering in a way that gives more importance to near values in both domain and range. On one hand, the spatial kernel, \(\sigma_s\), corresponds to a set of pixels that define the distance from the pixel to be filtered that is considered closer to it – like a radius –, and if any pixel is far from that distance, its Gaussian weight would be smaller. The total size of the window \(\Omega\) should be at least twice of \(\sigma_s\). As this spatial parameter increases, the larger features get smoothed. On the other hand, the range kernel, \(\sigma_r\), defines the maximum difference between intensity or color values to consider them similar to one another. If the difference is bigger, Gaussian weights are smaller. As the range parameter increases, Gaussian becomes nearly constant over the intensity interval of the image.

Those Gaussian weights are used in equation \((2.11)\) to compute the filtered depth value in pixel \((i,j)\): using depth values of the neighbouring pixels \(I_{\text{depth}}(k, l)\) inside the window multiplied by its correspondent weight and normalized as in equation \((2.10)\).

Now that the formulas are explained, in Figure 2.6 there is an example of how bilateral filter can work when there is noise in an image and there is an edge that needs to be preserved in that particular region.
Figure 2.6: (a) represents a 100-gray-level step perturbed with a Gaussian noise with $\sigma = 10$ gray levels. (b) represents the weights for a 23x23 neighborhood centered two pixels to the right of the step in (a). (c) shows the result of applying a bilateral filter with $\sigma_r = 50$ gray levels and $\sigma_d = 5$ pixels to (a). Image from [Tomasi and Manduchi, 1998].

Choosing the filter parameters is a delicate decision because it implies to know which range of values is good for our particular case. In Subsection 2.4, Figure 2.9 shows that intensities in same parts of the figure are pretty different. In order to preserve edges but to skip fake differences at the same time, the range parameter should be big enough. The rest of the values of the parameters are extracted based on execution tests to see which range gives best results.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_s$</td>
<td>[5, 10]</td>
</tr>
<tr>
<td>$\sigma_r$</td>
<td>[20, 40]</td>
</tr>
<tr>
<td>window size</td>
<td>$&gt; 2\sigma_s$</td>
</tr>
<tr>
<td>restrictive iterations</td>
<td>[30, 50]</td>
</tr>
<tr>
<td>Whole image iterations</td>
<td>[5, 10]</td>
</tr>
</tbody>
</table>

Table 2.3: Bilateral filter parameters.

In Table 2.3, some values for the parameters of bilateral filtering are set. When we do the whole image iterations, the parameter $\sigma_r$ is set smaller than the original one. This is done to have the edges more defined once all pixel images have reliable depth values.

In order to filter the raw depth maps, i.e. Figures 2.7a and 2.7c, we first perform a few iterations taking into account only those pixels where the depth was not previously computed on the raw image, to fill them with a depth value. Secondly, we perform a couple more of iterations with the whole filtered depth map to smooth pixel depth values. The depth maps shown in Figure 2.7 are performed over a 3D confidence volume with a voxel size of 0.01. As more voxels in the confidence
volume, the more pixels will have depth in raw depth maps. Then, in Figure 2.7b and 2.7d below, final depth maps are compared to their raw depth maps.

![Figure 2.7: Depth maps before and after filtering, \( \text{voxel size} = 0.01 \).](image)

In order to visualize image depth maps, we have to transform to the gray-scale range \([0, 255]\) from whatever range the depths have. Within this range, pixels corresponding to voxels close to the camera should be darker and, vice-versa, pixels from voxels far from the camera should be brighter.

### 2.4 Image Artifacts

Sometimes, artifacts appear in raw depth images and show inaccurate depth values, leading to a wrong interpretation when the bilateral filter is applied. An example from these artifacts can be seen in Figure 2.8.

![Figure 2.8: Artifact example in a depth image.](image)
These erroneous depth pixels may be due to the poor resolution of the original images or because the resolution of the confidence volume is not good enough. In Figure 2.9 it can be appreciated that in the same part of the object, two neighboring pixels have a huge intensity difference although they should be at a similar depth. This may cause mistakes when Gaussian weights from bilateral filter (equation (2.10)), leading to an erroneous depth pixel estimation.

![Figure 2.9: Color discrepancies in the same object part.](image)

(a) 50 gray value  
(b) 80 gray value

In order to reduce image artifacts from depth images, the first attempt would be to apply a Gaussian Blur filter [Gedraite and Hadad, 2011, Chen and Ma, 2009] to the original images before computing depth values. Although it does not completely solve the problem, it helps to balance those differences in intensity between pixels that should not be that high.

Despite blurring the image, some artefacts continue to appear in the depth maps. This might also be due to the quality of the photoconsistency measures and, most likely, the amount of voxels that belong to the visual rays. This last fact can only be solved by increasing the resolution of the 3D volume. To confirm this theory, we tested the algorithm with different confidence volume resolutions to see how the depth maps vary according to the voxel sizes. Figure 2.10 shows that as more voxels are considered the better the depth maps are.
Figure 2.10: Raw depth maps with different volume resolutions.
Chapter 3

SHAPE ESTIMATION

In the previous chapter, depth maps are computed for all the views $i$. If we want to spatially reconstruct the figure, in a static way, we do not need the rest of the frames from the video. However, to apply spatiotemporal reconstruction in order to refine the figure it is necessary to have depth maps estimated for all cameras $i$ but also for all frames $t$. In this manner, we could fuse depth information over space and time to recover the surfaces meshes of the object at any time instant $k$.

In this thesis, we implemented the first part: spatial integration using the first frame of the video for every camera. In this way, we could spatially reconstruct the figure at any time $k$ of the video by changing the $k$-value. To further refine the result, [Leroy et al., 2017] propose a spatio-temporal reconstruction approach by using a temporal window, which not only takes into account spatial information but also the temporal one. However, despite that in our work we explain what we have implemented – the spatial reconstruction –, it is easy to expand it to 4D following the steps that the authors proposed.

Then, after spatial reconstruction is performed, we use the Marching Cubes algorithm to extract a mesh from the 3D implicit form (in Section 3.2). At the end, we show and comment some results obtained with this algorithm.

3.1 Spatial Integration

This step is where all the information gathered on the steps in the previous chapter is put in common in a way that a surface mesh is extracted. We consider a single frame (a single time instant $k$) and the spatial integration of the depth maps $d_i$ for all cameras in the data set at $k$. The are several ways to perform this fusion, the
paper we base our work on mentions a few references of other works they might have followed that have similar objectives but in different contexts. The statement that all these techniques have in common is that the more information about the shape and measurements you can fuse, the better and precise the result would be.

In [S. Izadi and Fitzgibbon, 2011], a real-time 3D reconstruction and interaction using a moving depth camera is performed by KinectFusion. Many of the works that perform 3D reconstructions from images are prone to holes, and to avoid this, we try to get the images to cover the entire surface to rebuild. Instead, KinectFusion provides the possibility for a user to hold a Kinect camera in motion to create detailed 3D reconstructions of an indoor scene. It basically uses depth data to estimate the 3D pose of the sensor of the camera and then estimate the 3D model. As it can be seen, depth information is essential in most of reconstruction methods. On page 6 of this same paper a volumetric representation is used to integrate 3D information into voxels using a similar technique that we will use in this thesis (signed distance function [Osher and Fedkiw, 2003]). A distance field, in a simple way, is an implicit surface representation, in a manner that describes the space around the surface. As it is not our goal to enter into detail about how KinectFusion works, for further information the reader is referred to the paper mentioned.

Another example of 3D reconstruction can be seen in [Newcombe et al., 2015], which performs dynamic fusion doing a reconstruction and tracking of non-rigid scenes in real-time. In this case, a fusion of depth maps is also computed but using warp-fields. As the example above, it uses signed distance functions but for other purposes. For more details see the reference mentioned.

The last example of 3D reconstruction to mention is [Curless and Levoy, 1996], which is widely mentioned in recent research papers about surface reconstruction from depth data. It proposes an algorithm which uses cumulative weighted signed distance function into a voxel grid where the final mesh is extracted as the zero iso-surface (similar to [Newcombe et al., 2015] but using range images). This work and the other two mentioned above, inspired [Leroy et al., 2017] to use the similar technique for their depth maps reconstruction. In Figure 3.1 there is a graphical representation about how the reconstructed surface is interpreted and it can also be applied to our reconstruction. Later in the document this idea is developed.
Indeed, we benefit from the strategy of the Truncated Signed Distance Function (TSDF). The TSDF is a volumetric representation of a scene that integrates multiple depth images taken from different viewpoints into a 3D implicit surface. The signed distance function (SDF) was previously proposed to reconstruct a 3D surface from multiple range images [Curless and Levoy, 1996]. In our case, a 3-dimensional environment is represented in a 3-dimensional grid of cubic voxels with the same size. As in other steps in this paper, the position of each voxel is defined by its center. In order to construct the TSDF, there are two relevant values to calculate for each voxel $x$: the $sdf_i(x)$ and the voxel weight. The first value $sdf_i(x)$, denominated $\eta(x)$ in equation (3.1), is the signed distance or the relative distance between the voxel center and the actual nearest object surface in this voxel. Equation (3.1) defines the signed distance precisely denoting its final result as $F_i$. In front of the object or surface, the values are defined as positive (outside). Behind the surface (inside the 3D object), the distances are represented by negative values. Finally, the surface interface or, as we mentioned, the 3D implicit surface of the object is defined by the zero-crossing where the values of TSDF change sign.

\[
F_i(x) = \begin{cases} \min(\mu, \eta(x)) & \text{if } \eta(x) \geq -\mu \\ \emptyset & \text{otherwise,} \end{cases}
\]

\[
\eta(x) = d_i(\pi_i(x)) - ||c_i - x|| \tag{3.1}
\]

In detail, the function $\eta(x)$ is computed as the difference between the depth es-

![Diagram of unweighted signed distance functions in 3D](image)

Figure 3.1: Unweighted signed distance functions in 3D, from [Curless and Levoy, 1996].
timation of the pixel where voxel $x$ projects to with respect to camera $i$ ($d_i$) and the actual distance between the center of the voxel $x$ and the camera center $c_i$. In this equation only one parameter must be set by the user: $\mu$. This is a delicate parameter to set as it decides whether the signed distance is good enough or not. In [Zach et al., 2007] work, this parameter, denominated as $\delta$ in the paper, is defined as the 1% of the diameter of the reconstructed surface of the object. In my thesis and in our experiments, we decided to first give this parameter the value of the difference between the depth on the front of the surface of the object and half of it (manually checked). However, we have seen that it was too little and in the Experiments chapter we develop which value should be better. In the equation before, the signed distance function $\eta$ is truncated by this parameter $\mu$. In order to graphically see this truncation, we use Figure 1 of the mentioned paper (Figure 3.2 here) where the parameters are similar to ours and the operation of the truncation is well represented:

![Figure 3.2: Generation of 3D distance fields from range images [Zach et al., 2007].](image)

In Figure 3.2, due to the truncation procedure, the TSDF can have values between the interval [-1,1]. Instead, in our experiments, the TSDF can take values between [-$\mu$,+$\mu$].

If the visual hull computation is done by taking into account all the cameras for every voxel, the value $-\mu$ should not appear because at least one camera will contribute to the voxel $F_i$ value.

The second relevant values of the voxel are the camera weights at that voxel. In this case, this is linked to the photoconsistency of the voxel respect to the camera $i$, see equation (3.2). If the photoconsistencies $\rho$ of voxels are reliable, it behaves as a confidence measure for every $F_i$ value. Then, for a voxel $x \in \mathbb{R}^3$, the final value of the Truncated Signed Distance Function $TD(x) \in \mathbb{R}$ is defined as the weighted average of all camera predictions $F_i(x)$, for $i \in C$: 38
$$TD(x) = \frac{\sum_{i \in C_x} \rho_i(x) F_i(x)}{\sum_{i \in C_x} \rho_i(x)},$$

(3.2)

where $C_x = \{ i \in S : F_i(x) \neq \emptyset \}$. If $d_i$ in equation (3.1) is undefined at $x$, meaning it has no depth value, then the camera $i$ does not contribute to the TSDF. If no camera contributes to the TSDF at voxel $x$, but $x$ is inside the visual hull $V$, then the implicit surface is considered as inside, i.e. $TD(x) < 0$. See algorithm 5 to evaluate all possible cases.

In the work from [Werner et al., 2014], the computation of the TSDF and its parameters are explained well in detail and extensively. From this work, an interesting graphical representation to understand what has been exposed is seen in Figure 3.3. In Fig. 3.3(a) the TSDF($x$) of the voxel grid is encoded by color, whereas the TSDF of 3.3(b) is sampled along a viewing ray. The $\text{cam}_z(x)$ parameter is our $\|c_i - x\|$.

Figure 3.3: (a) An example of a 2D TSDF with the object surface in green. (b) 1D TSDF sampled along the ray through $p$ with $t = 1000$ mm. Object surface is at zero crossing, from [Werner et al., 2014].

In the following pseudo-code, there is an extended explanation of how this truncated signed distance function is computed. In our implementation, for voxels $x$ inside the visual hull where all the cameras contribute with an empty value in $F_i$, we set $TD(x) = -\mu$. The reason is to have a $TD$ defined in all voxels in order to be able to extract the iso-surface of level 0. Moreover, we check whether if the depth value is defined for a particular pixel or not before computing $F_i$. Both measures are security conditions to avoid having undefined values. However, in practice, we checked that the algorithm never enter these conditions.
Algorithm 5: Compute TSDF of object’s surface.

Data: Depth maps filtered and voxel photoconsistencies.

Result: 3D iso-surface from the TSDF.

parameters initialization: $\mu$;

for each voxel $x$ in volume grid do
  if ($x$ inside Visual Hull) then
    $p_{\text{norm}} = 0$ and $\sum F_i = 0$;
    $\text{cameraCounter} = 0$ and $\text{emptyCounter} = 0$;
    for each camera $i$ do
      Check if the voxel projection to camera $i$, $p$, is inside the image.
      Get $d_i \leftarrow$ depth value in $p$.
      if ($d_i$ is defined at $x$) then
        cameraCounter++;
        Compute $\eta = d_i - \text{distancesLut}(x)$;
        if ($\eta \geq -\mu$) then
          $\sum F_i += \rho_i(x) \times \min(\mu, \eta)$;
          $p_{\text{norm}} += \rho_i(x)$;
        else
          emptyCounter++;
        end
      end
    end
    if (cameraCounter == 0) then
      No camera contributes to the TSDF but voxel inside the VH:
      $\text{TD}(x) = -\mu / 2$;
    else if (emptyCounter == number of cameras) then
      $\text{TD}(x) = -\mu$;
    else
      $\text{TD}(x) = \frac{\sum F_i}{p_{\text{norm}}}$;
    end
  else
    $\text{TD}(x) = \mu$;
  end
3.2 Shape Mesh Generation

One of the fundamental problems in 3D reconstruction is the creation of high quality meshes. In the previously mentioned work by [Curless and Levoy, 1996] the authors proposed an algorithm which uses signed distance functions to accumulate surface information into a $n$-dimensional voxel grid. Then, as in section 3.1 is explained, the final 3D mesh is extracted from the Truncated Signed Distance Function $TD$ that results from the depth map fusion. In particular, it is extracted as the zero level (zero iso-surface) of the $TD$ function.

The Marching Cubes algorithm (MC) [Lorensen and Cline, 1987] is used to extract the volumetric surface model from the 3D implicit form coming from the resulting $TD$ as zero iso-surface extraction. This formulation of using cumulative signed distance fields over a voxel grid for a posterior extraction via zero iso-surface is spreadly used among researchers, [Curless and Levoy, 1996] [Zach et al., 2007].

Marching Cubes strategy has proven to be pretty efficient and fast, and has been widely used in several applications over the last decades. However, alternative solutions exist and solve the limitations of MC. The 3D shape discretization into cubic cells can produce an approximation of the final mesh with stretched or small triangles. For this reason and to have further precision, [Leroy et al., 2017] use another surface extraction strategy: Voronoi Tessellation (VT) [Wang et al., 2016b] or Centroidal Voronoi Tessellations (CVT) [Wang et al., 2016a]. This strategy provides regular shape tessellations and more accurate approximations of the implicit forms to transform. It is a good alternative to MC especially when working with dynamic scenes, where CVTs have desired properties such as the accuracy, regularity and invariance to rigid transformations.

As our main objective was to build an efficient and adequate algorithm to reconstruct a 3D surface, we chose the Marching Cubes approach due to its versatility and efficiency balance.

3.2.1 Marching Cubes Algorithm

The Marching Cubes algorithm creates triangle meshes from 3D implicit functions. The word *marching* comes from the concept of the iterations that the algorithm performs over a uniform grid of cubes. It takes eight neighbour locations (vertices) at each iteration – in our case the values of the vertices correspond to the values of the voxels of the $TD$ – forming a cube (Figure 3.4). A user specified value (iso-value $TD_{iso}$) is used to locate the surface and create the triangles. With
The algorithm determines how the iso-surface intersects a corresponding cube and then marches to the next cube.

To find the intersection of the surface in a cube, a binary value \( v_i \) is assigned at each vertex (voxel) \( x_i \) of the cube depending on the iso-value \( TD_{iso} \). A 1 is assigned if the data value at that vertex exceeds or equals this iso-value, meaning that the vertex is inside or on the surface. On the other hand, cube vertices with data values below this threshold correspond to vertices outside the surface and are set to 0.

\[
v_i (x_i) = \begin{cases} 
1 & \text{if } TD (x_i) \geq TD_{iso} \\
0 & \text{otherwise} 
\end{cases}
\] (3.3)

If all the eight vertices of a cube are assigned a value of 1, it means that the entire cube is inside the surface. Similarly, if all voxels are set to 0, it means that the cube lies outside the surface. In both cases, there is no intersection of the surface through the cube. The iso-surface will intersect a cube where one vertex is inside the surface \( (v_i = 1) \) and another vertex outside it \( (v_j = 0) \), where \( j \neq i \) and neighbour of \( i \).

![Cube Numbering](image)

Figure 3.4: Cube Numbering [Lorensen and Cline, 1987].

Since there are eight vertices in each cube which can have value 1 or 0, inside or outside, there are only \( 2^8 = 256 \) ways of intersecting with the surface. When the two different symmetries of the cube are taken into account, the problem is reduced to 128 cases: vertices greater than the surface value \( TD_{iso} \) can be interchanged with those vertices less than this value, being equivalent. Therefore, including the rotation symmetry property to these cases, it reduces the problem
into 14 final patterns (Figure 3.5). The simplest pattern, 0, occurs when all vertices are above (or below) the iso-value and, as there is no intersection with the surface, it produces no triangles. The next pattern, 1, occurs when one vertex is separated by the surface from the other seven, resulting in a triangle defined by three edges intersections (edges denominated by $e_i$, see Figure 3.4). In the figure below the rest of the 14 patterns are shown. If we combine by complementary and rotation properties this patterns, we get the 256 possible cases. These triangles are the faces of the output mesh while the intersections with the cube’s edges $e_i$ are the different vertices of the mesh.

![Figure 3.5: 14 Intersection Patterns [Lorensen and Cline, 1987].](image)

For each cube, the eight binary values corresponding to the eight vertices of the cube are concatenated into an index vector using the vertex numbering configuration seen in Figure 3.4, containing one bit for each vertex. This index is used to identify each of the cases shown in Figure 3.5 and to tell which edge of the cube the surface intersects. Linear interpolation is performed to identify the exact position of the intersection along the edge (i.e. the position of the triangle vertices). Nevertheless, higher degree interpolations show little improvements over linear interpolation.

The final step of the algorithm calculates a unit normal for each triangle vertex.
This normal is used by rendering algorithms to produce Gouraud-shaded images. A surface with constant density value has a zero gradient component along the surface tangential direction. In consequence, the direction of the gradient vector, \( \vec{g} \), is normal to the surface. To estimate \( \vec{g} \) at the surface of interest, the gradient is first computed at each cube vertex \((i, j, k)\) as follows:

\[
\begin{align*}
g_x(i, j, k) &= \frac{D_e(i+1, j, k) - D_e(i-1, j, k)}{\Delta x}, \\
g_y(i, j, k) &= \frac{D_e(i, j+1, k) - D_e(i, j-1, k)}{\Delta y}, \\
g_z(i, j, k) &= \frac{D_e(i, j, k+1) - D_e(i, j, k-1)}{\Delta z},
\end{align*}
\]

(3.4)

where \( D_e(i, j, k) \) is the density value of \( TD \) at the voxel coordinates \((i, j, k)\) and \( \Delta x, \Delta y, \Delta z \) are the lengths of the cube edges – as our voxels have the same size these parameters are constant. The gradient \( \vec{g} \) divided by its norm produces the unitary vector representing the normal at each vertex required for rendering. This normal is linearly interpolated to the point of intersection.

In summary, Marching Cubes algorithm creates a surface mesh from a three-dimensional data set using these steps:

1. Divide the 3D iso-surface input into slices.
2. Read four slices, scan two and create a cube from four neighbours vertices (voxels) on one slice and the four ones on the other.
3. Calculate an index for each cube by comparing the eight voxel values of \( TD \) at the cube vertices with the iso-value \( TD_{iso} \).
4. Using the index, find the intersecting edges from a precalculated table.
5. Using scalar values from \( TD \) at each edge vertex, find the surface-edge intersection via linear interpolation.
6. Compute a unit normal at each cube vertex. Interpolate this normal to each triangle vertex.
7. Finally, we have the output triangle vertices and vertex normals.

### 3.2.2 Extracting the Mesh

To extract the mesh in our project, we use the C++ open-source VTK library [Kitware, 2016] which has an implementation of the Marching Cubes algorithm.
In order to use it, we have to save the iso-surface resulting from the depth map fusion $TD$ in a VTK file format as it is the kind of input file it requires. It returns the resulting mesh in a Point Cloud format file (PLY).

The algorithm receives the iso-surface value, normally zero, as an input parameter. This allow us to test several iso-values to extract the resulting surface and see how good it was reconstructed. In the following section we show some results with different iso-values.

### 3.2.3 Visualizing our result

The MC algorithm returns the mesh written in a PLY file, so to visualize it we use the open-source mesh processing tool MeshLab [Cignoni et al., 2008]. As we can choose the iso-value desired to extract the surface, we are able to only visualize those voxels $x$ where $TD(x) \leq TD_{iso}$.

In the Experiments chapter we see many outputs depending on the parameters, but here we choose the best result obtained with our method displaying the mesh corresponding to the extraction of the iso-surface resulting from the TSDF at $TD_{iso} = 0.1$, displaying $TD(x) \leq 0.1$. In the results we can appreciate a full reconstruction of the figure in both cases: Figures 3.8a and 3.8b show the resulting fusion of the TSDF using a confidence volume formed by those voxels that project to all the silhouettes except one; on the other hand, Figures 3.8c and 3.8d are the resulting meshes from the TSDF depth map fusion using a confidence volume with voxels that project to all of the silhouettes. The difference between both meshes is pretty noticeable. Having $S$ as the set all the cameras or views in the dataset and $\alpha$ and $\beta$ parameters defined in Equation 2.2:
Figure 3.8: Meshes extracted from the TSDF fusion with $TD_{iso} = 0.1$. 

(a) $\alpha = S, \beta = S - 1$  
(b) $\alpha = S, \beta = S - 1$  
(c) $\alpha = \beta = S$  
(d) $\alpha = \beta = S$
Chapter 4

EXPERIMENTS

In this chapter, we present several experiments carried out in order to improve the accuracy of the method. We show some results and explain what changes we performed with respect to the original paper [Leroy et al., 2017]. One of the main problems we face is the low number of cameras (views) we have of the scene, which makes the good extraction of the depth maps difficult.

Our aim is to understand which parts are causing the errors that appear in the reconstruction process and, then, try to make the necessary changes to fix them. Here, we describe some alternative strategies and comment on whether they are better options or worse. It is necessary to clarify that due to the execution time required to obtain reliable results, we have suffered a lack of time to obtain robust conclusions on this matter. The ultimate goal is to have the complete figure located at the zero iso-surface but being different from the one in the visual hull (Figure 2.2).

4.1 Photoconsistency Measure Modifications

Once the mesh is visualized via Marching Cubes algorithm, we can see the reconstructed figure in 3D from the iso-surface resulting of the algorithm. Observing the parts of the figure that are not well reconstructed, it leads us to think that some parts of the method do not perform well. From here, some alternatives are sought to what has already been implemented to see if it can be improved. We assume that photoconsistencies may not be completely reliable within the confidence volume. In the figure below, we see the VTK surface at iso-value 0:
It is clearly nearly missing half of the figure, and the parameter $\tau_{photo}$ is very relevant, but it will be discussed deeply later. First of all, we focus our attention to the reconstructed part in order to decide if it seems well reconstructed or not. To do so, we compare the same part in both figures of the $TD_0$ and the visual hull (VH).

In the figure above we can appreciate that both surfaces are not the same. In Figure 4.2a the surface of the reconstructed 3D object approaches better to what the real figure should be. Although a voxel is photo-consistent from several views, even if it is occluded, it does not imply that the depth estimated for that voxel is correct. In order to have a better depth estimation it would be interesting to have more views where every voxel would be visible. The method we used is pretty robust to outliers and occlusions but the dataset we tested has few different viewpoints. However, we try to find the best configuration for the algorithm to increase the accuracy of the method.

The first approach we perform is finding another robust photoconsistency measure, so we apply a median filter to the values of $g_{i,j}$ for $i \in C_i$ ($g_{i,j}$ defined in
the equation (2.2)) maintaining the Gaussian filter \( W(g_{i,j}) \). Instead of doing the weighted average of the descriptors differences \( g_{i,j} \), we take the median value of them (Eq. (4.3)). We still take into account the angle between camera centers \( \theta_{ij} \) as it is a good constraint to keep the better ones. Our first thought was that the weights \( w_j \) in equation (2.3) could be adversely affecting the average. Mathematically, the normalized weighted mean and the median are defined as follows:

\[
\text{weighted mean} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}, \quad (4.1)
\]

\[
\text{median} = \begin{cases} 
\frac{x_{N/2} + x_{N/2+1}}{2} & \text{if } N \text{ is even} \\
 x_{(N+1)/2} & \text{if } N \text{ is odd} 
\end{cases}, \quad (4.2)
\]

where \( N \) is the number of samples and the \( x_i \) are ordered in increasing order. Instead of using these weights, in the median we just choose the median element’s value of the sorted elements of \( g_{i,j} \) without averaging them with any weight.

\[
\rho_i(x) = \text{median}_{j \in C_i}(W(g_{i,j}(x))) \quad (4.3)
\]

The median measure is usually performed to avoid outliers, and in this situation we wanted to obtain the proper photoconsistency value for a particular voxel. In the table below we can compare the photoconsistency values for two particular voxels \( x \) with respect every camera \( i \) using both measures.

<table>
<thead>
<tr>
<th>Measure</th>
<th>( \rho ) values in a particular voxel for each camera ( i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (Eq. 2.3)</td>
<td>0.76, 0.64, 0.69, 0.84, 0.89, 0.86, 0.85, 0.73</td>
</tr>
<tr>
<td>Median (Eq. 4.3)</td>
<td>0.71, 0.63, 0.75, 0.87, 0.91, 0.88, 0.84, 0.78</td>
</tr>
<tr>
<td>Mean (Eq. 2.3)</td>
<td>0.49, 0.84, 0.84, 0.72, 0.86, 0.81, 0.89, 0.69</td>
</tr>
<tr>
<td>Median (Eq. 4.3)</td>
<td>0.48, 0.71, 0.94, 0.69, 0.89, 0.86, 0.91, 0.82</td>
</tr>
</tbody>
</table>

Table 4.1: Mean vs. Median photoconsistency measures in two voxels.

By comparing voxel values we cannot conclude anything relevant as the results are pretty similar. A better way to see it is to extract the iso-surface with the \( TD_{iso} \) value at 0 using both measures to get a clear idea of whether the change on the calculation of the photoconsistency is better or worse. We should mention that both measures are robust due to the Gaussian function applied to \( g_{i,j} \) in both cases.
Thanks to the images in the figure comparison we can state that using the median in this case does not give us any improvement. In Fig. 4.3a, there are more parts missing than in Fig. 4.3b, and it also has a hole above the legs that is not in the figure on the right. Moreover, the shape on the left is not as close to the real object as the shape on the right figure. These results indicate that the weights used in the photoconsistency measure are more useful than we thought at the beginning. To conclude, from now on we will continue to make the experiments using the weighted mean for $\rho$ values.

Another approach to increment the precision of the final iso-surface is to modify the mentioned parameter $thr_{\theta}$. As explained, this parameter is what restricts the contribution of views to $\rho$ using their angles as conditioners. In this way, we try to use only those views that can see the intensity of the voxel in the same way. As we have few cameras and weights that correspond to them, we try to reduce a little the threshold for the cosine of the angle $\theta_{ij}$. This way, we let more views to participate into the weighted mean. Although some views may not be optimal, their weights will balance the result in a proper way. The ideal values of this threshold reside between 0.5 and 0.8.
4.2 Depth Maps Parameters

In the first section we mentioned the importance of the parameter $\tau_{\text{photo}}$, and in this one we will discuss its reasons. This parameter is used when computing the depth maps (Equation (2.6)). A thresholding is performed to determine if the estimated depth for a particular pixel is proper or not by checking the photoconsistency value. If the accumulated photoconsistency along the voxels in a pixel’s ray falls below this threshold, we consider that it as non-reliable. As a consequence, the depth at the pixel is assigned to the first depth $d_v$ in the ray. Otherwise, if it falls above the threshold, the depth for the pixel is set to the estimated one.

Therefore, $\tau_{\text{photo}}$ acts as a decision limit on photoconsistencies by deciding whether they are good or bad for a particular pixel depth value. As we mentioned at the beginning, the most reliable photoconsistencies are found in the legs and the waist down, on the contrary, there are more problems in the upper part of the body. We guess that these problems come from the high RGB difference values seen by the front and backwards. If we look at the equation (3.1), the value of $\eta$ is determined by the difference between the estimated depth and the real depth value. What it happens in the head area is that the depth values estimated by the pixels are greater than the real ones, causing a positive difference value for $\eta$ and, as a result, a positive value to the $TD$. This is the main reason why we do not have a good reconstruction of the upper part of the figure for $TD$ values less than or equal to 0.

When we increase the value of the parameter $\tau_{\text{photo}}$, most voxels are left with the initial estimated depth in the visual hull (without using photoconsistency information). This benefits the parts of the body where photoconsistency is bad but it
disadvantages the part of the legs where the photoconsistencies were good enough to improve the depth of the VH. For this reason, we see more reconstructed parts of the surface in Figure 4.4b than in Figure 4.4a but the quality of the legs reconstruction is not that good. In the end, we have to find a value that would provide a balance between the amount of reconstructed surface and the quality of the reconstruction.

4.3 TSDF Modifications

Due to certain unexpected results of the implicit surface on the “zero-crossing” area extracted with the Truncated Signed Distance Function, it leads us to meditate on what the potential causes may be. The first solution to which we turn is to avoid doing the weighted mean with the photoconsistencies of voxels in case they contain errors or are not entirely reliable. Instead, we perform the mean only with the $F_i$ values.

$$TD(x) = \frac{\sum_{i \in C_x} F_i(x)}{|C_x|},$$  

(4.4)

where $|C_x|$ indicates the cardinality of set $C_x$. A priori, this will not be better than the average photoconsistency if the weights of photoconsistency are not really bad. In this case, we compare the two figures at $TD_{iso} = 0$ and their contours to clarify this doubt. In Figure 4.5 we compare the iso-surface resulting from the depth map fusion with all the rest of parameters equivalent. The two contours are at the same height in the figure.
At first glance, we might think that by using the mean without averaging it with the photoconsistencies the more figure is recovered at iso-value 0. This could be the case, but what must be also taken into account is the quality of the reconstruction. In this aspect, the quality of the shape of the figure on the right (Fig. 4.5b) is far worse than the other. The surface of this figure is thicker and expanded, which is closer to the initial Visual Hull. However, performing the average with the values of photoconsistencies for those voxels inside the VH we get a more realistic surface (Fig. 4.5a) and closer to what the ground truth figure would be. On the other hand, it is difficult to extract good conclusions from the comparison between both contours. However, the blue contour (Fig. 4.5c) corresponding to the average mean figure is more accurate than the pink contour (Fig. 4.5d). As a conclusion, if we would prefer quantity over quality, we could use raw mean instead the average to get more surface reconstructed.

As changing the way of performing the average of $F_i$ does not improve the implicit form of the surface, we make another approach. Returning to equation (3.2), the average with photoconsistencies, let us take another value of $\mu$, making it
bigger. This parameter is responsible for truncating the signed distance function \( sdf \), so it is quite decisive in the results of the final \( TD \). Instead of taking the value mentioned before in the explanation of this parameter, now we choose to give it a value a bit bigger or equal than 1. The reason of this change is to let more negative differences of \( \eta \) (3.1) to participate into the average mean for \( TD \) function (3.2). This way, we compensate the positive values of \( \eta \) due to erroneous miscalculated depths.

![Image](image_url)

(a) TD with \( \mu = 0.4 \)  
(b) TD with \( \mu = 1 \)  
(c) TD with \( \mu = 1.5 \)

Figure 4.6: Amount of reconstructed figure at \( TD_{iso} = 0 \) depending on \( \mu \) parameter.

As mentioned earlier, this parameter behaves as a maximum limit of the amount of difference between an estimated depth and the actual depth. If the estimation error is very large in a particular voxel, it would mean that its contribution to the \( TD \) is unreliable and we want to discard it.

In Figure 4.6, we compare the zero iso-surface resulting from the depth maps fusion with different values for \( \mu \). With a value too small (Figure 4.6a), it is likely that we will truncate too soon and discard relevant or necessary values to contribute to the weighted mean. On the other hand, a value too large (Figure 4.6c) may lead to the acceptance of outliers, or large estimation errors. Therefore, we need a sufficiently large but at the same time restrictive value, so we stay with \( \mu = 1 \).
4.4 Confidence volume resolutions

As mentioned earlier in several sections, the resolution that we use in the confidence volume affects the rest of the algorithm. A simple case where we could see this effect would be when we calculate the optimum depth value throughout the rays: the more voxels belonging to a ray, the better the algorithm can choose the one that best corresponds to the pixel. Thus, the final reconstruction result is also affected. To see some effects of this condition, we analyze how the final result of the algorithm varies due to this resolution constraint.

In order to compare how the resolution affects the result, the other parameters used in the algorithm should be the same or the most similar as possible. In Figure 4.7 we compare two iso-surfaces resulting from the $TD$ implicit form with $TD_{iso} = 0.1$. For this iso-value, using a voxel size of 0.014 is not enough to have a full reconstruction. However, if the voxels size in the confidence volume is 0.01, we can have a complete reconstruction for the same iso-value (see the top of the surface). In addition, Figure 4.7b seems to be less thicker and more distant from the original visual hull.

![Figure 4.7](image)

(a) voxelsize = 0.014 (b) voxelsize = 0.01

Figure 4.7: $TD_{iso} = 0.1$

Then, in Figure 4.8 we use the same result as the figure before but showing the resulting iso-surface at a different iso-value: $TD_{iso} = 0$. The difference between both figures is a lot bigger, and it shows clearly that an increase in resolution in the confidence volume increases directly the accuracy of the final result.
Another aspect of the resolution of the visual hull is given by the condition of the projection of voxels within this volume to a certain number of silhouettes ($\beta$). The more restrictive (high) this value is, the closer the VH will be to the real surface (but also the more sensitive to miss-detections in the silhouettes). For instance, in the following figure we compare the difference of using a confidence volume where all voxels must project to all the silhouettes of the dataset (Fig. 4.9b) or projecting into all the silhouettes except one (Fig. 4.9a). Although the surface of the right is thinner and closer to the actual 3D figure, if we are so restrictive with voxels inside the visual hull the reconstructed amount of iso-surface at $TD_{iso} = 0$ is not as much as the one on the left.

Figure 4.9: Mesh with TSDF iso-surface extracted at 0-value.
4.5 Runtimes

Another handicap we encountered to test the best method for our algorithm was the time it took to show the final result with proper parameters. In order to have a good result, the resolution of the confidence volume should be high enough to perform a precise depth calculation. The main problem is that the more resolution we use, the more time the algorithm takes to calculate all the steps of the method. This project has been run on a laptop with a 4-core processor: 2.7 GHz Intel Core i7. Some of the steps are performed with parallel execution via OpenMP directives [Beyer and Jeff Larkin, 2016], such as the LUT distances and the LUT rays, in order to use the four nuclei of the CPU at the same time and thus accelerate the process.

In Table 4.2, we show the execution times for every step of the algorithm depending on the voxel size, i.e. the resolution we choose. We are interested in performing this test with a size equal or smaller than 0.013, since larger sizes do not give reliable results.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Voxel Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.013</td>
</tr>
<tr>
<td>Visual Hull</td>
<td>8s</td>
</tr>
<tr>
<td>Daisy descriptors and voxel coordinates</td>
<td>1h 12min</td>
</tr>
<tr>
<td>( g_{i,j} )</td>
<td>10min</td>
</tr>
<tr>
<td>( \rho_i )</td>
<td>10min</td>
</tr>
<tr>
<td>DistancesLUT</td>
<td>1h 23min</td>
</tr>
<tr>
<td>RaysLUT</td>
<td>1h 23min</td>
</tr>
<tr>
<td>Depth maps</td>
<td>19s</td>
</tr>
<tr>
<td>TD</td>
<td>34min</td>
</tr>
<tr>
<td><strong>Overall algorithm</strong></td>
<td><strong>5h</strong></td>
</tr>
</tbody>
</table>

Table 4.2: Runtimes depending on voxel sizes and for each one of the steps.
Chapter 5

CONCLUSIONS

The main objective of this project was to implement a multi-view 3D reconstruction system from silhouettes and photoconsistency information based on depth map fusion. We have accomplished this main goal, and now we comment if the results are what we expected, the strengths and the weaknesses of the algorithm implemented among other ideas.

We have implemented a 3D reconstruction method which uses the silhouettes from all the views to extract an initial confidence volume. Then, photoconsistencies are computed throughout these views for every voxel in the confidence volume to create depth maps for every view. Finally, depth maps are fused using Truncated Signed Distance Functions. This pipeline was extracted from [Leroy et al., 2017], where they use innovative techniques and state-of-the-art methods which allowed us to implement a system robust enough. Using their work as a baseline, we performed some modifications to adapt the steps to our possibilities, e.g. a reduced number of views. We also had to define some parameters of the algorithm whose values were not indicated on the paper. In order to find the values that offered a good performance, we had to test them until finding the good ones.

The main difference between the paper we followed and our implementation is the use of time cues contributing to the TSDF and implicit surface creation, where we have only considered the spatial integration step. Most of the modifications we made were to get an efficient implementation due to our time constraint, for example, we performed bilateral filtering instead of superpixel clustering or Marching Cubes algorithm instead of Voronoi algorithm.

The results obtained from the algorithm show a 3D reconstruction good enough of the real figure although there are certain aspects that could be improved. At the zero iso-surface from the depth map fusion, the reconstruction of the figure
is not as accurate as we expected it to be. These unexpected results may be due to the insufficient number of viewpoints, leading to an inaccurate depth at some voxels. However, we believe that increasing both the resolution of the initial confidence volume and the number of intermediate points of view the result would be significantly more similar to the real 3D object.

We mentioned some of the weaknesses of our algorithm but they do not imply that the techniques we used are bad. In fact, the use of photoconsistency information has been proved to be one of the most successful approaches in MVS reconstruction systems (see Related Work (Section 1.1) on Chapter 1). Moreover, the use of DAISY descriptors, that are robust features, makes the photoconsistency measure to be robust. The combination of silhouettes cues and photoconsistency to build depth maps makes the overall method robust enough to achieve a proper result.

During the preparation of this project, we noticed that the initial expectations were too high, as we did not have much prior experience in the field. For this reason, a range of possibilities is opened for future work either to solve the weaknesses mentioned and as to improve the algorithm. The first point to improve would be the refinement of the surface using temporal cues by performing spatio-temporal integration (4D) to the TSDF. This use of temporary information through the frames of the video helps to solve problems of complex holes and concavities that can be found in the figure. This would be one of the main improvements to be made if one wants to continue with the proposed algorithm and following the method proposed by [Leroy et al., 2017]. Based on the implementation we have used, this extension can be easily aggregated. Due to the poor estimation of depth values in certain complex regions, an idea to improve depth maps would be to perform depth calculation using regions of pixels instead of doing it pixel by pixel independently. Another interesting possibility would be to paint the final mesh of the algorithm built with Marching Cubes in order to see the texture of the object and provide a more realistic result. In Runtimes section (Chapter 4), it was shown how much time it takes to the algorithm to calculate all the steps. Although we implemented some OpenMP directives [Beyer and Jeff Larkin, 2016] to speed up the process, we still could adapt more the algorithm to benefit from parallel computing at a higher level via OpenMP. Alternatives like NVIDIA’s CUDA Toolkit [Nickolls et al., 2008] to execute the same tasks on GPU instead of CPU could lower significantly the computational time. At the end, we wanted to test our method with multiple datasets having more views than the one we used, but it remains as future work as well due to lack of time.

The implementation of the algorithm explained in this thesis is available at https://github.com/helenacots/3D_Reconstruction. The code of this
implementation is easily extendable to the 4D approach we mentioned. Regarding the mesh extraction step which uses Marching Cubes algorithm, we did not implemented it and we used the corresponding classes from the VTK library. The code is written in C++ language, using the OpenCV library.
Bibliography


66


