Università di Catania

UNIVERSIDADE DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Report of the Final Degree Project presented by

Stefano Di Patti

To obtain the Bachel or in Computer Science

3D Reconstruction from Multiple 2D Images without Camera Calibration

September, 2020

Final Degree Project nº: E 19/20-90

Advisor: Arno Formella

Area of knowledge: Languages and Computer Systems

Departmant: Computer Science

Abstract

This paper presents a prototyping tool that creates a 3D model out of several different images taken with a consumer camera retrieving a specific object. It represents a significant ambit of Computer Vision for different applications, for example: robot route, object recognition and scene understanding, 3D modeling and animation, mechanical control and clinical analysis. Recovering the lost dimension from just 2D images has been the goal of multiview stereo, which have been extensively investigated for many years. The images of the object can be obtained in two ways: by using *multiple cameras at fixed angles* or *single camera and varying the angles*.

Different approaches has been used in order to solve this problem based on geometric perspective, voxelisation, surface-based algorithm, shape-from-silhoutte methods or deep learning. This study proposes a mixture of image-based reconstruction based on binocular stereo view with the simulation of a stereo system and aims to partially reconstruct the 3D shape of the object using a single camera and various angles approach.

Contents

List of Figures

… If a machine is expected to be infallible, it cannot also be intelligent.”

Alan Turing

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Arno Formella, whose patience and expertise were invaluable in formulating the research questions and methodology.

I would also like to thank Alma Gómez Rodriguez for her disponibility during my Erasmus+ period.

I would like to acknowledge my colleagues for every subject studied together and for their support during those years.

In addition, I would like to thank my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside university. Without them I would not have made it through my degree! Especially Emanuela Santoro for always supporting and motivating me in moving forward.

Finally, I would like to thank my parents and my sister for their presence through the time. You are always there for me.

1. Introduction

1.1 3D Reconstruction

Vision is perhaps the most important sense that man possesses. It allows us to infer the three-dimensional world, to recognize and locate the objects in a scene, to perceive the rapid changes in the environment, etc. The capabilities of biological systems are formidable: the eye collects a band of electromagnetic radiation bounced off different surfaces and from different light sources and the brain processes this information forming the picture of the scene as we perceive it.

Computer Vision is the discipline that studies how to enable computers to understand and interpret visual information present in images or videos. In Computer Vision 3D reconstruction is the process of capturing the shape and appearance of real objects from a multimedia source. The research of 3D reconstruction has always been a difficult goal. Using 3D reconstruction one can determine any object's 3D profile, as well as knowing the 3D coordinate of any point on the profile.

Recovering the lost dimension from 2D space has been investigated for a long time and achieved with different approaches divided as passive and active methods. The main difference is that active methods interfere with the considered object to reconstruct by applying mechanic or radiometric struments in order to acquire the depth map, e.g. structured light, laser range finder and other active sensing techniques. While passive methods analyze the objects' structure through image understanding. In our case we are approaching a passive method technique.

If the object is sufficiently simple then two images are enough for modeling. If the complexity of the object increases for perfect reconstruction a larger number of images is needed. The 3D reconstruction of an objects is a generally scientific problem and core technology of a wide variety of fields.

1.2 Most used techniques in 3D Reconstruction

The first generation of methods approached the problem from the geometric perspective; they focused on understanding and formalizing, mathematically, the 3D to 2D projection process in order to find a solution to the ill-posed inverse problem. Effective solutions typically require multiple images, captured using accurately calibrated cameras. Stereo-based techniques require matching features across images captured from slightly different viewing angles. They use the triangulation principle to recover the 3D coordinates of the image pixels. There are various algorithms like SIFT or NURBS proposed for reconstruction of the object from 2D images. More specifically SIFT is used to establish a correspondence between feature points of the multiple images. In the feature detection there may be some inaccurate matching. These mismatching pairs of feature points are also called the outliers. An outlier removal and model estimation method called the random sample consensus (RANSAC) is used to remove the outliers and to estimate the two-view geometry using the inlier feature points.

The Non-Uniform Rational B-Spline (NURBS) represents a mathematical form, that addresses the problem of analytical shapes, e.g. lines, conics etc., as well as the free-form shapes. It is shape invariant under affine and perspective projections and it is the generalized form of B-spline and Bezier curve.

Another method uses the idea of direct voxelisation of 2D images to 3D. This method is based on the simple fact that if 2D images can be represented as the collection of tiny squares called pixels then the 3D models can be expressed as the collection of tiny cubes and we call them as voxel i.e. volume pixel. In these voxel-based approaches, it is critical to robustly and accurately extract the silhouettes.

Other approaches are the surface-based that aim to estimate a surface representation of the visual hull rather than a volume representation. The 3D surface of an object is reconstructed by analyzing the geometric relationship between the silhouette boundaries and the visual hull surface. These methods directly estimate a mesh representation of the 3D surface by computing the intersection of the cone surfaces associated with the silhouette edges.

Otherwise, in the medicine environment the Inverse Radon Transform has been an important and active research topic. The context of tomographic reconstruction is similar to the volume-based approach in the sense that both are back-projection techniques.

But all shape-from-silhouette methods require accurately segmented 2D silhouettes. These methods may not be practical or feasible in many situations because there is a need to have a well-calibrated camera. Also, they are computationally expensive, time consuming and require vast resources.

Instead of using binary silhouettes, photo-consistency approaches consider photometric information which compute sets of photo-consistent voxels called Photohull. This method assumes a Lambertian surface and constant illumination, which means a valid point on the scene surface appears with the same colour over all visible images.
The second generation of 3D reconstruction methods tried to leverage this prior knowledge by formulating the 3D reconstruction problem as a recognition problem. The avenue of deep learning techniques, and more importantly, the increasing availability of large training data sets, have led to a new generation of methods that are able to recover the 3D geometry and structure of objects from one or multiple RGB images without the complex camera calibration process. Despite being recent, these methods have demonstrated exciting and promising results on various tasks related to computer vision and graphics.

For the work proposed in this paper, the approach used it a fusion of Multiview Stereovision and Images-based reconstruction.

1.2.1 Multiview Stereovision

The goal of multiview 3D reconstruction is to infer geometrical structure of a scene captured by a collection of images. Usually the camera position and internal parameters are assumed to be known or they can be estimated from the set of images. By using multiple images, 3D information can be (partially) recovered by solving a pixel-wise correspondence problem. Since automatic correspondence estimation is usually ambiguous and incomplete further knowledge (prior knowledge) about the object is necessary. A typical prior is assume that the object surface is smooth.

Over the last few years, a number of high-quality algorithms have been developed, and the state of the art is improving rapidly. Unfortunately, the lack of benchmark datasets makes it difficult to quantitatively compare the performance of these algorithms and to therefore focus research on the most needed areas of development.

Structure From Motion (SFM) is generally used to structure images. This means it estimates photo location, its orientation, Camera parameters. Multi view stereo (MVS) takes this location and orientation etc information from SFM and make a 3D dense point cloud. So in order to make a 3D model from set of images, we have to first do SFM and then MVS.

The major MVS algorithm consists of two steps: (i) estimating the 3D points on the basis of a photo-consistency measure and visibility model using a local image matching method and (ii) reconstructing a 3D model from estimated 3D point clouds. The accuracy, robustness and computational cost of MVS algorithms depend on the performance of the image matching method, which is the most important factor in MVS algorithms.

Using a multiview stereovision system, images can be taken:

- by using multiple cameras at fixed angles
- by using single camera and varying the angles.

For our scope, we treated the input images as a sequence of pair stereo views with a single camera. Each image is matched with all other images allowing a more detailed reconstruction, since every difference between pairs is considered during the creation of the 3D points clusters.

1.2.2 Image-based techniques

Image-based 3D reconstruction do not interfere with the object desired to reconstruct; they are only based through image understanding. Generally, a sensor (e.g. an image sensor in a camera) to measure the radiance reflected or emitted by the object's surface infer its 3D structure, so the input to the method is a set of digital images or video which are studied with Image Processing mechanisms. The most common techniques are:

- Monocular cues methods, which consists of construct the 3D objects' shape using a single viewpoint but one or multiple images. It makes use of 2D characteristics (e.g. Silhouettes, shading and texture) to measure 3D shape, and that's why it is also named Shape-From-X, where X can be silhouettes, shading, texture etc. It is a simple and quick approach, and only one appropriate digital image is needed thus only one camera is adequate.
- Shape-from-shading uses Lambertian reflectance based on shade information in the image to restore and reconstruct the object surface with the depth of normal.
- Photometric Stereo examinates different images taken in different lighting conditions to solve the depth information.
- Shape-from-texture consider an object with smooth surface covered by replicated texture units, and its projection from 3D to 2D causes distortion and perspective. Distortion and perspective measured in 2D images provide the hint for inversely solving depth of normal information of the object surface.
- Binocular Stereo Vision obtains the 3D geometric data from multiple images basing on the research of human visual system. The results are presented in form of depth maps. Images of an object acquired by two cameras simultaneously in different viewing angles, or by one single camera at different time in different viewing angles, are used to restore its 3D shape and its 3D profile and location. This method requires two identical cameras with parallel optical axis to observe one same object, acquiring two images from different points of view. In terms of trigonometry relations, depth information can be calculated from disparity. It also is well developed and stably contributes to favorable 3D reconstruction, leading to a better performance when compared to other 3D construction systems. Unfortunately, it is computationally intensive, besides it performs rather poorly when baseline distance is large.

The method used in the thesis is similar to the Binocular Stereo Vision approach. The 3D information is given by the definition of a depth map based on the images' features. The images are taken with one camera from different angles but in the algorithm they are considered taken from a stereo camera, so they are treated in pairs because it is easier to define a correspondance between the objects since the camera calibration is missing.

1.3 Objectives

Three-dimensional reconstruction of scene can be viewed as is the reproduction of a depth-map. Thus, as an input we have a sequence of images, which their output is distances of each point of the target object from a reference point. There are many ways to reconstruct an object. Starting from the classical triangulation scheme in which two or more images of the same static object are acquired and correspondence between points is formed (epipolar coordinates). Other more sophisticated methods use variation of light patterns falling on the object (structured light), feature tracking. Methods from reconstruction based on a single viewing plane can be seen in structure from motion and structure from shading. Three-dimensional objects reconstruction finds numerous applications in the real world through many fields and human activities. Reconstruction allows us to gain insight into qualitative features of the object which cannot be deduced from a single plane of sight, such as volume and the object relative position to others in the scene. The importance of three-dimensional reconstruction comes from the fact computer algorithms cannot make significant inferences regarding our three-dimensional world without input coming from all spatial dimensions. Commonly, we fill this gap in the algorithm ability using tricks and manipulation based on our understanding of the space as humans, rather than the computer actually seeing in 3D.

In this thesis we use principles of Image Processing to describe the theoretically transformation of 2D points present in an image to a 3D world coordinate which can be interpreted and represented from a generical 3D software. Since the introduced problem is still in the stage of advanced studies in the world of computer science, the main focus is not to obtain a final product that automatically convert a sequence of images in input into a complete 3D model, but to propose a prototyping tool which combine two known techniques in this field to see which benefits we can gain in the reconstruction process. The most important scope of the project is to propose a way to reconstruct 3D data from multiple images without the use of camera calibration. With camera calibration, the feature extraction process that allows to match correspondance between images is easy and well-working. But for common applications (supposing the use of a phone camera for example) we usually can't run a calibration. The challenge was to make up for this lack by finding a way to reconstruct the geometry of real space by performing a pure analysis by comparing the images and 2D information.

In the first part of the thesis, we deepen the importance of feature detection to describe the content in the images. In the second part we use the extracted features to model a depth map which contains the 3D information about our images and we use it to establish the objects' shape. In the last part we analyze how to deal with the created 3D points. They are structured into cluster called Point Clouds and their position in the reconstructed 3D space respect the point of view is fixed through geometry algorithms. Finally a surface definition process is applied and the final model can be used in computer graphic softwares or for 3D printing.

1.4 Related Disclipines

It is very difficult to draw up an exhaustive list of all the fields to which the Computer Vision can be applied, because the topic is vast, multidisciplinary and constantly expanding: new applications are born continuously. For this we will pause to describe only the aspects and disciplines strictly related to what concerns our work.

Image Processing. Image processing is the discipline that deals with rework images using mathematical operations and any shape processing. For our purposes it is a discipline that differs from Computer Vision in what concerns the properties of the image. Since many algorithms for Computer Vision require image processing as a preliminary step, it is easy to superimpose the two disciplines. Particular areas of application of Image Processing include: image enhancement (calculates an image with a better quality than to the original), image compression (compact representation of the image, used for transmission), image restoration (eliminates known effects due to degradation), feature extraction (identification of characteristic elements of the image such as contours, or structured areas).

Photogrammetry. Photogrammetry is the science of making measurements from photographs.

The input to photogrammetry is photographs, and the output is typically a map, a drawing, a measurement, or a 3D model of some real-world object or scene. Many of the maps we use today are created with photogrammetry and photographs taken from aircraft.

Structure from Motion (SfM). Structure from Motion (SFM) is to determine the spatial and geometric relationship of the target through the movement of the camera, which is a common method of 3D reconstruction. It only needs an ordinary RGB camera, so the cost is lower, and the environment is less restricted, and it can be used indoors and outdoors. However, SFM requires complex theories and algorithms to support it, and it needs to be improved in accuracy and speed, so there are not many mature commercial applications.

1.5 Applications

3D Reconstruction is widely used nowadays in Computer Vision for many applications such as:

Robot navigation. The goal for an autonomous robot is to be able to construct (or use) a map or floor plan and to localize itself and its recharging bases or beacons in it. A detailed understanding of these environments is required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot. Artificial intelligence and computer vision share other topics such as pattern recognition and learning techniques. Consequently, computer vision is sometimes seen as a part of the artificial intelligence field or the computer science field in general.

Object recognition and scene understanding. Given one or (typically) more images of a scene, or a video, scene reconstruction aims at computing a 3D model of the scene. In the simplest case the model can be a set of 3D points. More sophisticated methods produce a complete 3D surface model. The advent of 3D imaging not requiring motion or scanning, and related processing algorithms is enabling rapid advances in this field. Grid-based 3D sensing can be used to acquire 3D images from multiple angles. Algorithms are now available to stitch multiple 3D images together into point clouds and 3D models.

3D modeling and animation. Due to Multimedia application and demand of realistic animation in movie/game development requires advanced techniques for character generation and dynamic movement of these characters to create realistic scene. These applications need the construction of 3D model of character specifically face and it is followed by animation. Models can be obtained with 3D reconstruction while processing video is used during movie developement with motion techniques.

Industrial control. A second application area in computer vision is in industry, sometimes called machine vision, where information is extracted for the purpose of supporting a manufacturing process. One example is quality control where details or final products are being automatically inspected in order to find defects. Another example is measurement of position and orientation of details to be picked up by a robot arm. Machine vision is also heavily used in agricultural process to remove undesirable food stuff from bulk material, a process called optical sorting.

Medical diagnosis. One of the most prominent application fields is medical computer vision, or medical image processing, characterized by the extraction of information from image data to diagnose a patient. An example of this is detection of tumours, arteriosclerosis or other malign changes; measurements of organ dimensions, blood flow, etc. are another example. It also supports medical research by providing new information: *e.g.*, about the structure of the brain, or about the quality of medical treatments. Applications of computer vision in the medical area also includes enhancement of images interpreted by humans—ultrasonic images or X-ray images for example—to reduce the influence of noise.

Archeology and Arts. characterized by contrasting needs for protect and make available rare finds or unrepeatable works of art.

1.6 Python, OpenCV & Open3D

The project has been developed with Python 3.6.10. Python is a programming language very used nowadays for its ability to convert ideas into code simply. It is currently the most prevalent, mature, and well-supported among programming languages in the area of machine learning, that is why many developers use Python for CV. Implementing CV through Python allows developers to automate tasks that involve visualization. Being an interpreted and object-based language allows programmers to benefit of a simplified approach with complex scenarios. Libraries like OpenCV are written in C++ and make Python have slower runtime as it will still call C/C++ libraries.

This means you will have the development advantage from Python while you can have performance optimization from C++. Python is free, unlike MATLAB, which also specializes in data analysis, exploration, visualization, etc. and allows you to deploy your work for free on sites. Also, Python has micro frameworks that are just as functional as their larger counterparts. This means it has a bigger community. There are a lot of blog posts and online resources regarding Python + OpenCV, so you can always get help most of the time trying to fix a problem.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. The library is used extensively in companies, research groups and by governmental bodies. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

Open3D is an open-source library that supports rapid development of software that deals with 3D data. The Open3D frontend exposes a set of carefully selected data structures and algorithms in both C++ and Python. The backend is highly optimized and is set up for parallelization. Open3D was developed from a clean slate with a small and carefully considered set of dependencies. It can be set up on different platforms and compiled from source with minimal effort. Open3D has been used in a number of published research projects and is actively deployed in the cloud.

Since the work is defined as a prototyping tool, to run it and test it Python interpeter and the libraries mentioned above are needed.

1.7 Planning and monitoring

Before starting the development of the proposed prototyping tool some research about which kind of 3D reconstruction model was better has been done. In particular, the problem was to find a way to recreate the three-dimensional shape of an object having avaiable a set of pictures given by the professor without the possibility to implement a camera calibration. The first approach to the study was to understand how the process normally works, so tests with camera calibration and objects with an easy geometrical form were taken in exam.

More specifically, the first tests have been done using the software Autodesk Maya, in which thanks to a plane with a chessboard texture it was possible to determine the intrinsic and extrinsic camera parameters and reconstruct a primitive solide like a cube. The reconstruction was not properly good enough so the second step was try to utilize a sensor camera with camera calibration. The problem got here was the presence of noise due to the natural light reflected by the objects around the subject to reconstruct. After a lot of research on Internet about open source projects and after consulting my Computer Vision professor in Catania I understood the problem could be solved with the use of SfM (Structure from Motion). Benefiting of what has been studied during the classes, an approach with Features Detection and Disparity Map has been preferred respect the Optical Flow. Implementing ORB and calculating the fundamental matrix was still not enough to have a precise Depth Map in order to apply a triangulation to reconstruct the 3D points. Homography and rectification of the images have been studied before passing them to the function which creates the Depth Map obtaining a better representation of the profundity in the real space. Testing the Depth Map with different parameters was another long step during the process.

To have more details about the missing information in the images another loop to compare one image with all the others was added. The last step was to create the cluster of 3D points. The first try was to use the Perspective transformation matrix given by the OpenCV documentation to apply the standard method of triangulation. Since each point in an image corresponds to a line in 3D space, all points on the line in 3D are projected to the point in the image. If a pair of corresponding points in multiple images can be found it means they are the projection of a common 3D point. So this method is an easy consequence of the work done before with epipolar geometry and fundamental matrix. But since it requires the camera calibration in Python, testing it gave strange forms result.

Starting using Open3D library and applying the back-projection using the inverted intrinsic camera matrix, finally showed the expected output, but the computation was too slow and complex. After fixing some parameters like the field of view and debugging the project with less but selected images the 3D shape pretty similar to the examined object was accomplished. The last step was to try to fix the geometry of the 3D points with some Open3D algorithms and find a computation to transform the combined points into a surface. Three methods have been tested:

- **Surface reconstruction based on the euclidean distance between points**: it was too computationally honerous and the 3D model had too many unfixed problems in the geometry;
- **Surface reconstruction using voxels**: the fastest method but also the less accurate;
- **Surface recontructions using Poisson algorithm:** the right compromise. The only problem with that method is that usually the shape is encircled with a not useful surface. Also here the parameters have been tested to represent the best 3D shape possible.

The last part of the project was attempting to improve its quality, precision and computation complexity.

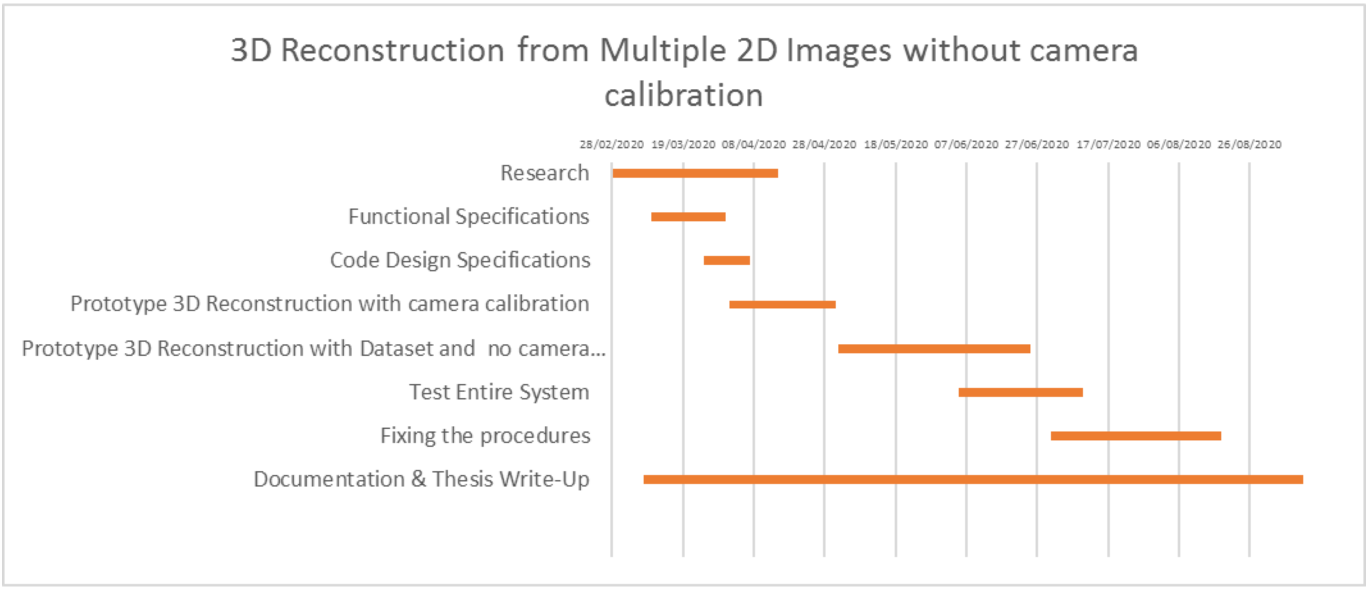

Fig. 1 Gantt chart about thesis planning and developing

1.8 Summary of the project

The algorithm proposed is a mixture of image-based modeling techniques and Multiview stereovision. Two images are considered taken with a stereovision system and compared using the ORB feature extraction. A group of 60 images about a flower that hasn't bloomed taken with a Samsung Galaxy S9 phone has been the testing dataset for the project. The images have been ordered manually to simulate in a better way the use of a stereovision system.

The first process is consider a pair of images, convert them in gray scale and apply the Kuwahara Filter to reduce the possible noise. ORB is implemented to determinate the feature points of the two images. Cycle per cycle one image is compared with all the others. The extraction of feature points allows us to match a correspondance between the image pair and so calculate the Fundamental Matrix. Any outliers points are removed and then Homography of both images is accomplished using the obtained Fundamental Matrix and RANSAAC algorithm to deaden miscalculation of the camera calibration absence.

Epipolines geometry is fitted with Homography to achieve a sort of rectified images that will be used during the developement of right and left views Depth Map. Those are combined into one Depth Map, and the result is normalized and inverted before applying Canny edge detector to remove the background points.

Using the K matrix of intrinsic parameters of the known camera, a back-projection gives us the final camera coordinates. Those are converted into a Point Cloud, that is a set of 3D coordinates and they are stored in the disk memory. Point Clouds are then retrieved to be fused in a generic Point Cloud containing all the 3D information of the different views. Since having different views means having scattered points, a geometry point-to-point ICP algorithm reorders their angolation and distribution. The last step is the creation of a triangle mesh within Poisson surface reconstruction formula. The mesh in addition is reduced and cleaned from all degenerate, duplicated triangles, verteces and from non-manifold edges.

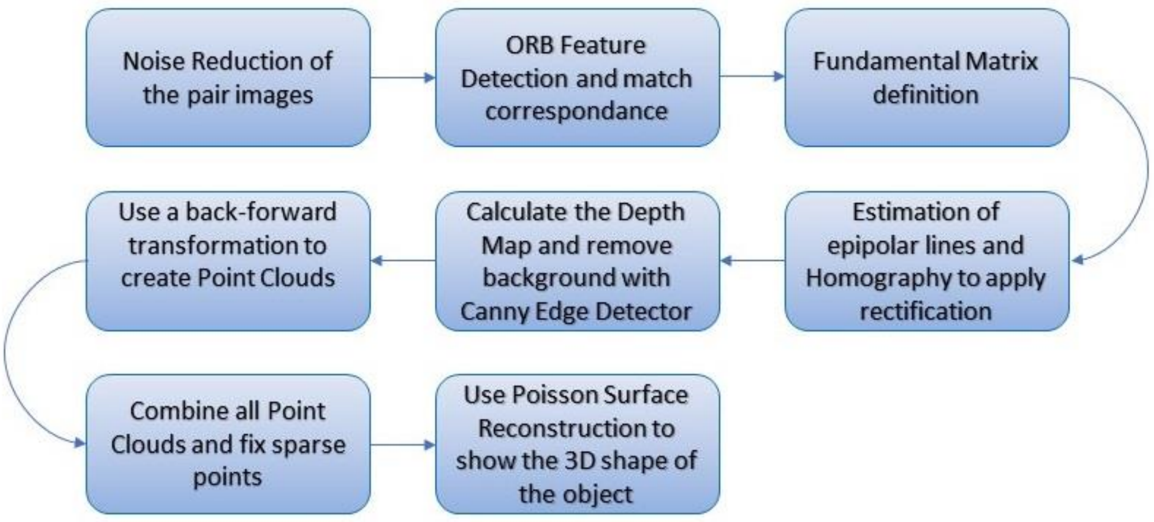
Noise Reduction of the pair images
ORB Feature Detection and match correspondance
Fundamental Matrix definition
Estimation of epipolar lines and Homography to apply rectification
Calculate the Depth Map and remove background with Canny Edge Detector
Use a back-forward transformation to create Point Clouds
Combine all Point Clouds and fix sparse points
Use Poisson Surface Reconstruction to show the 3D shape of the object

Fig.2 Workflow overview of the proposed tool

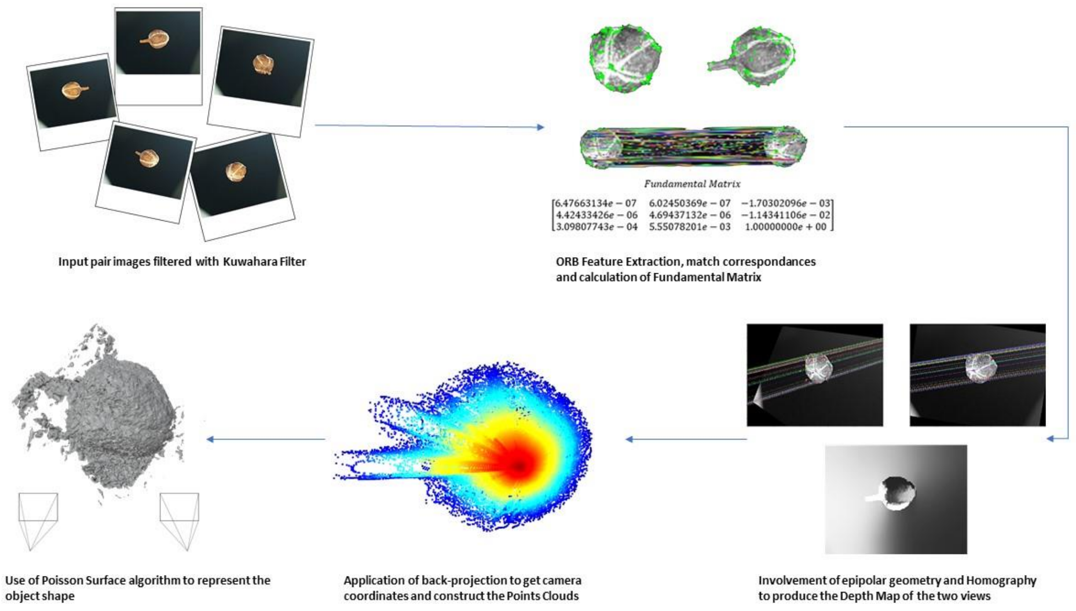
Input pair images filtered with Kuwahara Filter
Fundamental Matrix
$\begin{bmatrix} 6.47663134e-07 & 6.02450369e-07 & -1.70302096e-03 \\ 4.42433426e-06 & 4.69437132e-06 & -1.14341106e-02 \\ 3.09807743e-04 & 5.55078201e-03 & 1.00000000e+00 \end{bmatrix}$
ORB Feature Extraction, match correspondances and calculation of Fundamental Matrix
Involvement of epipolar geometry and Homography to produce the Depth Map of the two views
Application of back-projection to get camera coordinates and construct the Points Clouds
Use of Poisson Surface algorithm to represent the object shape

Fig.3 Visual rappresention of the structure of the 3D Reconstruction problem

2. Feature Detection

2.1 Noise Reduction using Kuwahara Filter

The first step of the algorithm regards the noise reduction of the images. Since images were obtained with a pinhole camera and we are not supposed to have its calibration, it is important to reduce the noise those devices normally have (for example the radial distortion). The dimension of the photos is also reduced to speed up the reconstruction process and because it is not required in our case of study an high level of details. **Kuwahara filter** is used for noise reduction, it is a non-linear smoothing filter with the particularity that it preserves edges. First of all the images are turned into a grey scale, then a square window of size *2a+1* is centered around an *xy* point in the image. The square is subdived into four smaller regions:

$$Qi(x,y) = \begin{cases} [x, x+a] \times [y, y+a] & if\ i = 1 \\ [x-a, x] \times [y, y+a] & if\ i = 2 \\ [x-a, x] \times [y-a, y] & if\ i = 3 \\ [x, x+a] \times [y-a, y] & if\ i = 4 \end{cases}$$

Where × is the cartesian product. Pixels above borders between two regions belong to both of them so there is a slight overlap between subregions.

The arithmetic mean $m_i(x,y)$ and standard deviation $\sigma_i(x,y)$ of the four regions centered around a pixel *(x,y)* are established to determine the value of the central pixel. The output of the Kuwahara filter for any point *(x,y)* is then given by:

$$\phi(x,y) = m_i(x,y) \text{ where } i = argmin_j \sigma_j(x,y)$$

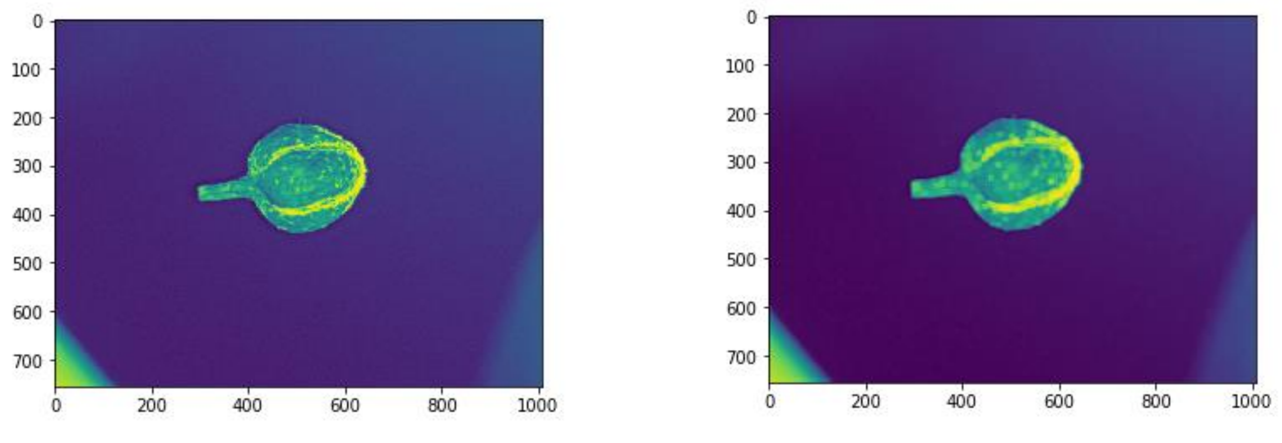

Fig. 4-5 The effect of using a Kuwahara filter on an original image (left image) using a window with sizes of 11 pixels

This implies the focal pixel will take the mean estimation of the region that is generally homogenous. The area of the pixel related to an edge assumes an extraordinary job in figuring out which district will have the greater standard deviation. Since the filter considers the homogeneity of the regions it is guaranteed the preserving of edges. The use of the mean, on the other hand, creates the blurring effect. Kuwahara filter uses a sliding window approach to access every pixel in the image. The size of the window is chosen in advance depending on the desired level of blur. Bigger windows are used for abstract images whereas small windows produce images that retain their detail. In the algorithm a square of dimension 5 is used.

2.2 ORB (Oriented FAST and Rotated BRIEF)

In the project two images are considered taken with a stereovision system, one image is compared with all the others using the ORB feature extraction cycle per cycle. This will entail a long-time computation but will give us more information about the 3D information of the object.

ORB uses FAST keypoint detector and BRIEF descriptor with many modifications to increase the performance. First it use **FAST (Features from Accelerated Segments Test)** to find keypoints by considering pixel brightness around a given area. A pixel is defined as keypoint if at least 8 pixels have higher brightness than it in 16 pixels intensities marked in a circle but the same result can be achieved by comparing 4 equidistant pixels on the circle speeding up the keypoints finding process by 4 times. Keypoints provide us the locations where the pixel intensities are varying. What we get is prominent corners of an object from which we can identify an object from opposed to any other object in the image. However, FAST doesn't consider the orientation. It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation.

More specifically: designated a pixel *p* in an array FAST compares *p* brightness to the surrounding 16 pixels present in a small circle around *p*. After the comparison, the circle's pixel are divided in three classes (lighter than *p*, darker than *p* or similar to *p*). If more than 8 pixels are darker or brighter than *p* than it is selected as a keypoint. However, FAST features doesn't include orientation component and multiscaling. For that reason ORB algorithm uses a multiscale image pyramid, that is a multiscale representation of a single image, that consist of sequences of images all of which are versions of the image at different resolutions. Each level in the pyramid contains the downsampled version of the image than the previous level. Once ORB has created a pyramid it uses the FAST algorithm to detect keypoints in the image. By detecting keypoints at each level ORB is partial scale invariant.

For descriptors, ORB use **BRIEF (Binary Robust Independent Elementary Features)** descriptors. BRIEF convert all keypoints into a binary feature vector, that is a vector that only contains 1 and 0 which is 128–512 bits string. A smoothing of the image using a Gaussian kernel is applied to prevent descriptor from being sensitive to high-frequency noise. Than a random pair of pixels is selected in a defined neighborhood around a keypoint (patch), which is a square of some pixel width and height. The first pixel in the random pair is drawn from a Gaussian distribution centered around the keypoint with a stranded deviation. The second pixel in the random pair is drawn from a Gaussian distribution centered around the first pixel with a standard deviation by two. Now if the first pixel is brighter than the second, it assigns the value of 1 to corresponding bit else 0. For a 128-bit vector, brief repeat this process for a keypoint. The problem is that it performs poorly with rotation.

So ORB uses **rBRIEF(Rotation-aware BRIEF):**

Consider a smoothed image patch, *p*. A binary test τ where $\tau(p; x, y)$ is defined as:

$$\tau(p; x, y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases}$$

where *p(x)* is the intensity of *p* at a point *x*.The feature is defined as a vector of *n* binary tests:

$$f(n) = \sum_{I<i<n} 2^{i-1} \tau(p; x_i \, y_i)$$

The matching performance of BRIEF falls off sharply for in-plane rotation of more than a few degrees. for any feature set of *n* binary tests at location (x_i, y_i), define a *2×n* matrix, *S* which contains the coordinates of these pixels:

$$S = \begin{pmatrix} x1, \dots & x_n \\ y1, \dots & y_n \end{pmatrix}$$

Then using the orientation of patch ϴ, its rotation matrix is found and rotates the *S* to get rotated version S_θ:

$$S_\theta = R_\theta S$$

Now, the steered BRIEF operator becomes:

$$g_n(p, \theta) = f_n(p) | (x_i, y_i) \in S_\theta$$

ORB then discretize the angle to increments of 12 degrees, and construct a lookup table of precomputed BRIEF patterns. As long as the keypoint orientation is consistent across views, the correct set of points will be used to compute its descriptor.

In the code the ORB function is implemented within its default parameters.

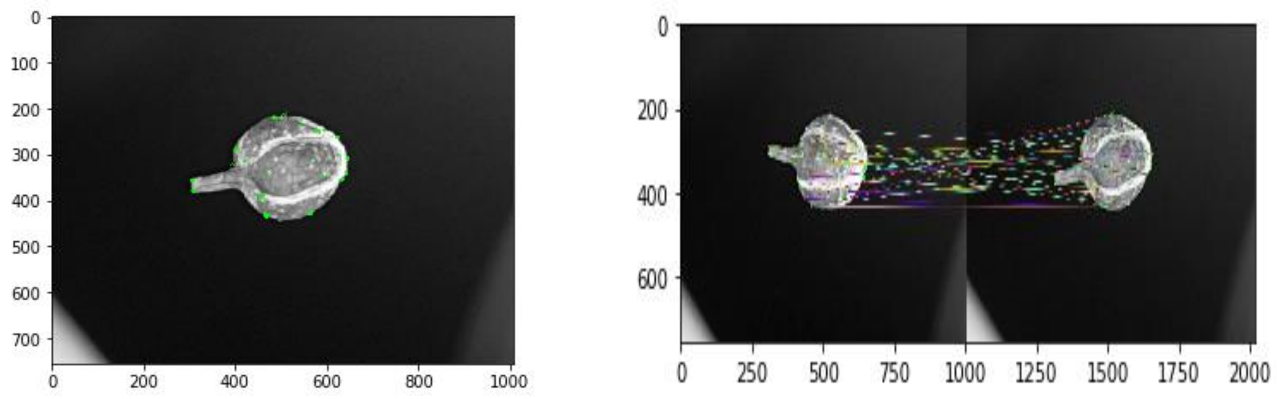

Fig. 6-7 On the left are shown feature points extracted with ORB. On the right the correspondance between stereo pair images

2.3 Epipolar Geometry and Fundamental Matrix

The **epipolar geometry** is the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the cameras' internal parameters and relative pose. The geometric entities involved in epipolar geometry are:

- The **epipole** is the point of intersection of the line joining the camera centres (the baseline) with the image plane. Equivalently, the epipole is the image in one view of the camera centre of the other view. It is also the vanishing point of the baseline (translation) direction.
- An **epipolar plane** is a plane containing the baseline. There is a one-parameter family (a pencil) of epipolar planes.
- An **epipolar line** is the intersection of an epipolar plane with the image plane. All epipolar lines intersect at the epipole. An epipolar plane intersects the left and right image planes in epipolar lines, and defines the correspondence between the lines.

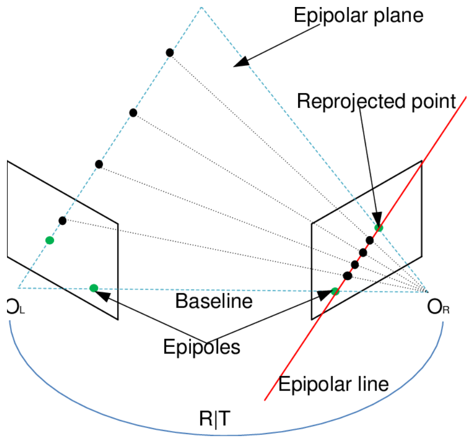

Fig. 8 Example of Epipolar Geometry considering two images

The **fundamental matrix** is the algebraic representation of epipolar geometry. It is a 3×3 matrix which relates corresponding points in stereo images. In epipolar geometry, with homogeneous image coordinates, x and x', of corresponding points in a stereo image pair, Fx describes a line (an epipolar line) on which the corresponding point x' on the other image must lie. That means, for all pairs of corresponding points holds:

$$x'^T F x = 0$$

The fundamental matrix is a relationship between any two images of the same scene that constrains where the projection of points from the scene can occur in both images. Given the projection of a scene point into one of the images the corresponding point in the other image is constrained to a line, helping the search, and allowing for the detection of wrong correspondences. The relation between corresponding image points, which the fundamental matrix represents, is referred to as epipolar constraint, matching constraint, discrete matching constraint, or incidence relation.

Being of rank two and determined only up to scale, the fundamental matrix can be estimated given at least seven point correspondences. Its seven parameters represent the only geometric information about cameras that can be obtained through point correspondences alone.

The properties of the fundamental matrix are then elucidated, both for general motion of the camera between the views, and for several commonly occurring special motions. It is shown that the cameras can be retrieved from F up to a projective transformation of 3-space. If the camera internal calibration is known, the Euclidean motion of the cameras between views may be computed from the fundamental matrix up to a finite number of ambiguities. Moreover, it can be computed from correspondences of imaged scene points alone, without requiring knowledge of the cameras' internal parameters or relative pose.

In our case, reconstruct the fundamental matrix without camera calibration may give us an high level of error. To avoid this, one method called RanSaC (Random Sample Consensus) is used to properly fit the model. For RanSaC, 8–12 sample correspondences are randomly chosen to calculate fit a fundamental matrix to. Then the number of inlier correspondences are counted for this randomly sampled fundamental matrix. This process is repeated thousands of times and the fundamental matrix with the most inliers will be returned as the chosen model.

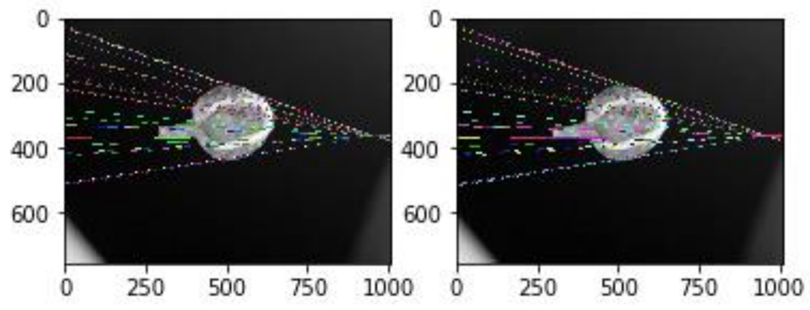

Fig. 9 Epipolar lines obtained using the Fundamental Matrix of stereo pair images

2.4 Homography and Rectification with no Camera Calibration

In the field of computer vision, any two images of the same planar surface in space are related by a homography (assuming a pinhole camera model that is also our case of study). This has many practical applications, such as image rectification, image registration, or computation of camera motion—rotation and translation—between two images.

Image stereo-rectification is the process by which two images of the same solid scene undergo homographic transforms, so that their corresponding epipolar lines coincide and become parallel to the x-axis of image. A pair of stereo-rectified images is helpful for dense stereo matching algorithms. It restricts the search domain for each match to a line parallel to the x-axis. Due to the redundant degrees of freedom, the solution to stereo-rectification is not unique and actually can lead to undesirable distortions or be stuck in a local minimum of the distortion function.

A **Homography** is a transformation (a 3×3 matrix) that maps the points in one image to the corresponding points in the other image.

Now since a homography is a 3×3 matrix we can write it as

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Let us consider the first set of corresponding points (x_1, y_1) in the first image and (x_2, y_2) in the second image. Then, the Homography H maps them in the following way

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

it heavily depends on the epipolar geometry. Therefore, if the camera lenses have a significant distortion, it would be better to correct it before computing the fundamental matrix and calling this function.

H maps points x and x' given all the points $X's$ (world points corresponding to x and x') lie on the same plane. Let us assume projection matrices for two views are:

$$P = (I|0) \text{ and } P' = (A|a)$$

Composing of rotation and translation matrices and plane π is defined by

$$\pi^T X = 0 \text{ with } \pi = (V^T, 1)^T$$

then using perspective geometry and since X lies on the plane π, we can say that

$$sx = PX = [I|0]X \text{ and } s'x' = P'X = [A|a]X \text{ and } [v\ 1]X = 0$$

where *s* is the scale factor denoting the ambiguity when we convert from image space to world space due to inherent loss of 3D information. Lets assume rotation matrix for the camera is Identity and translation is zero. Using the equations above we can easily conclude that:

$$s'x' = sHx \quad cx' = Hx \qquad \text{where } c = s'/s \text{ and } H = A - av^T$$

Thus, homography can be calculated using relative rotation and translation between two cameras. But it is often the case that we don't have the relative pose between two views (or to put in a different way we don't always have two calibrated cameras) and still we are required to calculate the mapping between first image and second image for the planar points in the world. Before going into how to calculate homography using point correspondences, lets briefly discuss some of the properties of a homography matrix. Homography relates points in first view to points in the second view and since there are no constraints in either views it is a full rank (=3) matrix. Also, homography is defined upto a scale (c in above equation) i.e. it can be changed by a non zero constant without any affect on projective transformation. Thus, homography has 8 degree of freedom even though it contains 9 elements (3x3 matrix).

Lets assume $x = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ and $x' = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ in homogeneous coordinates

and $H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$

$$c\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

By eliminating *c* we can formulate the above equation in the form

$$Ah = 0$$

Where $A = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{pmatrix}$

And $h = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^T$

When there are more than 4 points it would be an over-determined case and hence we have to use least square solution to find *h.*

Hartley and Zisserman proposed a normalization step before calculating *h* more stable in presence of noise and to prevent divergence from correct result. The idea is to calculate transformation matrix *T* and *T′* which transforms *x* and *x′* such that centroid of these points are coordinate origin and fixed average distance from origin. We can obtain the original homography matrix using backward formulas. Homography may fail and give us distorted rectifications. In that case the algorithm will still proceed just considering the correct points for the creation of the Depth Map.

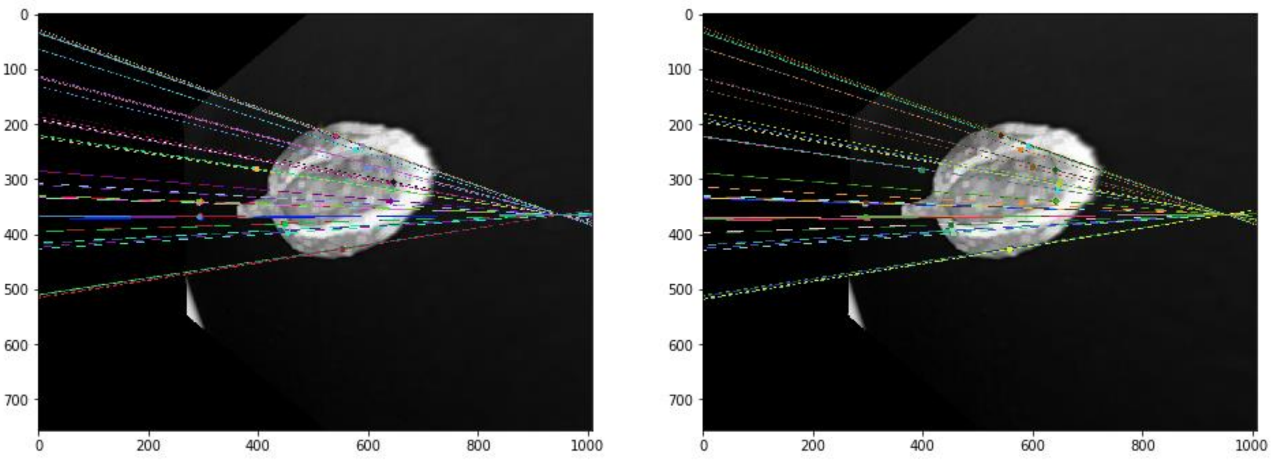

Fig. 10 Example of rectification using the Homography of both stereo images

3. Depth Map

3.1 StereoSGBM

The problem of estimating depth is an on-going research and has well progressed over the years. Many techniques have been developed and the most successful methods come from determining depth using stereo vision.

A **depth map** is an image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint. The term is related to and may be analogous to **depth buffer**, **Z-buffer**, **Z-buffering** and **Z-depth**.The "Z" in these latter terms relates to a convention that the central axis of view of a camera is in the direction of the camera's Z axis, and not to the absolute Z axis of a scene. The depth of a point in real world space can be calculated through the use of mathematical formulas and functions that incorporate the principles of epipolar geometry.

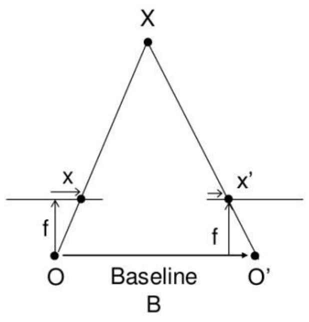

Fig. 11 Representation of how the depth X is established from the two views

First we should note the optical centers of both our cameras at *O* and *O'* in addition to the point in 3D space whose depth we are trying to find; point *X*. Length *f* is the focal length of our cameras. Now *x* and *x'* are the points on the 2D image plane where point *X* will appear in each image respectively. The idea here is that the equivalent triangles as denoted *Ofx* and *O'fx'* can be compared to each other in order to extract the depth *Z* of point *X*. The equation that allows us to accomplish this is shown below.

$$disparity = x - x' = \frac{Bf}{Z}$$

To describe the equation in as plain terms as possible, it tells us that the distance *Z* of point *X* away from our cameras' optical centers is inversely proportional to the difference in distance of points *x* and *x'* (the image points of our object as they appear on the 2D plane).

In summary Open CV is being used to find matched points in each of our images, greatly aided and expedited by the principles of an epipolar constraint. Subsequently it proceeds to calculate the disparity between the matched points therefore giving us the depth of the object in question. This process can be repeated for every pixel in our stereo images offering our concluding solution to 3D reconstruction.

The **StereoSGBM** class implements the modified **H. Hirschmuller algorithm**. The algorithm matches blocks, not individual pixels. The parameters used have being tested to reach the best quality of the depth image. In our case here is the code used in the project:

```
matcher = cv2.StereoSGBM_create(
        minDisparity=3,
        numDisparities=24,
        blockSize=18,
        P1=8 * 3 * window_size ** 2,
        P2=32 * 3 * window_size ** 2,
        disp12MaxDiff=1,
        uniquenessRatio=30,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
    )
```

A computation of the "left" and "right" images is done with the StereoSGBM before applying a **Weighted Least Squares Filter** according to the use of left-right-consistency-based confidence to refine the results in half-occlusions and uniform areas. Mutual information cost function is not implemented. Instead, a simpler Birchfield-Tomasi sub-pixel metric from is used. Some pre- and post- processing steps are included, for example: pre-filtering (Sobel) and post-filtering (uniqueness check, quadratic interpolation and speckle filtering).

The result image is then normalized and passed to the Edge detection function.

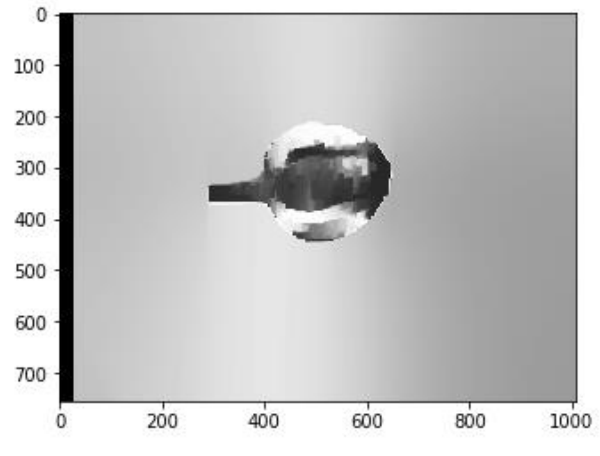

Fig. 12 Depth map result of the use of StereoSGBM combining two views

3.2 Edge Detection

The depth map result includes information of the depth pixels of the whole images, so its use to create the 3D Point Clouds lead us to have background points we don't need for the reconstruction. In order to remove those points an Edge Detection function is used to recognize the structure of the object and use this part for applying the back-reprojection and find the 3D points from the pixels.

More particularly, the **Canny Detector** is used. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle(\theta) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions. After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. This stage also removes small pixels noises on the assumption that edges are long lines.

After Edge Detection, countours are determined and sorted. An empty black mask is created filled with white polygon on it corresponding to largest contour. The mask will be smoothed and bluered with Gaussian filter before being applied to create the black/white image of the object with a blend.

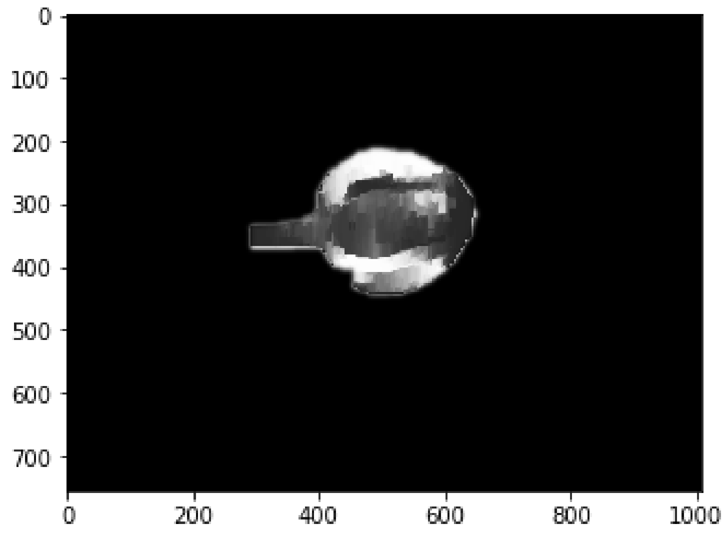

Fig. 13 Application of masking a black background image with the result of Canny Edge detector on the Depth Map

4. Point Clouds

4.1 Intrinsic Camera Parameters

First and foremost, understanding the geometrical model of the camera projection serves as the core idea. What we are ultimately interested in is the depth, parameter Z. Here, we consider the simplest pinhole camera model with no skew or distortion factor.

3D points are mapped to the image plane *(u, v) = f(X, Y, Z)*. The complete mathematical model that describes this transformation can be written as *p = K(R|t) * P*.

$$s\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & u_0\\0 & f_y & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

where

- p, (u′, v′, z′) the projected point on image plane in pixel coordinate
- K is the camera intrinsics matrix
- (R|t) is the extrinsic parameters describing the relative transformation of the point in the world frame to the camera frame.
- P, (X, Y, Z, 1) represents the 3D point expressed in Euclidean coordinate system
- aspect ratio scaling, s: controls how pixels are scaled in the x and y direction as focal length changes

The matrix K is responsible for projecting 3D points to image plane. To do that, the following quantities must be defined as:

- focal length *(fx, fy)*: Measure the position of the image plane to the camera centre.
- principal point *(u0, v0)*: The optical centre of the image plane.
- skew factor: The misalignment from a square pixel if the image plane axes are not perpendicular. In our example, this is set to zero.

The most common way of solving all the parameters is using the checkerboard method. Where several 2D-3D correspondences are obtained through matching and solving the unknown parameters by means of PnP, Direct Linear Transform or RANSAC to improve robustness. With all the unknowns determine, we can finally proceed to recover the 3D points (X, Y, Z) by applying the inverse.

Consider the equation above. Suppose (X, Y, Z, 1) is in the camera coordinate frame. i.e. we do not need to consider the extrinsic matrix (R|t). Expanding the equation would give as

$$u = \frac{1}{s_x} f \frac{X}{Z} + O_x \qquad v = \frac{1}{s_t} f \frac{Y}{Z} + O_y$$

The 3D points can be recovered with Z given by the depth map and solving for X and Y. Since we couldn't determine the intrinsic parameters using checkerboard, we consider the focal lengths and optical centre for the pinhole camera model as follows. We just assume that the optical center can be approximated by the center of the image, while to discover the focal lenght the information needed is the imaging sensor height and width in pixels and the effective field of view in the vertical and horizontal direction. Camera manufacturer usually provides those. In our example, we will use +77.8 degrees both in the vertical and horizontal direction. We will set the scale factor to 1. We will also assume that radial distortion does not exist.

Now, the inverse is computed as follows:

- Obtain the intrinsic camera parameters, K
- Find the inverse of K
- Apply equation with Z as depth from a depth map.

Using Linear Algebra:

*cam_coords = K_inv @ pixel_coords * depth.flatten()*

The @ (at) operator is intended to be used for matrix multiplication.

The camera coordinates are reduced with a field of view (<=80 meters in our case), transposed and used as a parameter to create the Point Cloud using the geometry library of Open3D. The point clouds are flipped using a vector of the form:

(([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])

The Point Clouds are finally stored in the computer memory as .pcd file extension.

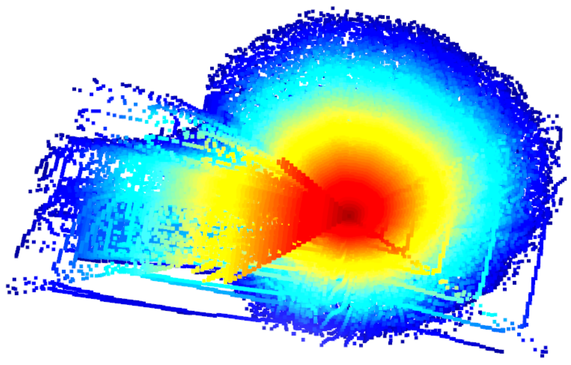
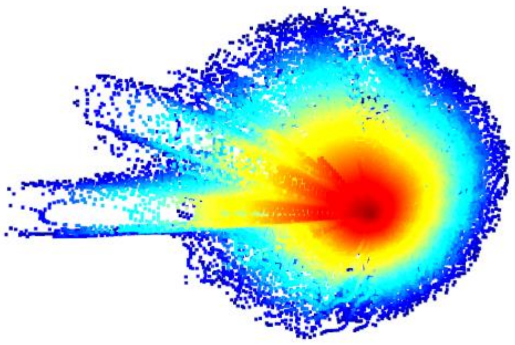

Fig. 14-15 Point clouds showed with Open3D after the computation of every image pair

4.2 Multiway Registration

The Point Clouds are downsampled and misaligned. **Multiway registration** is the process to align multiple pieces of geometry in a global space. Typically, the input is a set of geometries,in this case Point Clouds. The output is a set of rigid transformations , so that the transformed point clouds are aligned in the global space.

Open3D implements multiway registration via **pose graph optimization**. A pose graph has two key elements: nodes and edges. A node is a piece of geometry associated with a pose matrix which transforms it into the global space. The set of matrices are the unknown variables to be optimized. We set the global space to be the space of the first node. Thus the first matrix is the identity matrix.

The other pose matrices are initialized by accumulating transformation between neighboring nodes. The neighboring nodes usually have large overlap and can be registered with **Point-to-plane ICP (Iterative Closest Point)** registration algorithm, which possess a faster convergence comparing other methods. The input are two point clouds and an initial transformation that roughly aligns the source point cloud to the target point cloud. The output is a refined transformation that tightly aligns the two point clouds. The point-to-plane ICP algorithm uses the next objective function:

$$E(T) = \sum_{(p,q)\in K} ((p - Tq)n_p)^2$$

where n_p is the normal of point *p*.

A pose graph edge connects two nodes that overlap. Each edge contains a transformation matrix that aligns the source geometry to the target geometry. This pairwise registration problem is proved to be error-prone. False pairwise alignments can outnumber correctly aligned pairs. Thus, they partition pose graph edges into two classes. Odometry edges connect temporally close, neighboring nodes. A local registration algorithm such as ICP can reliably align them. Loop closure edges connect any non-neighboring nodes. The alignment is found by global registration and is less reliable. In Open3D, these two classes of edges are distinguished by the uncertain parameter in the initializer of PoseGraphEdge.

In addition to the transformation matrix, the user can set an information matrix for each edge. If used the loss on this pose graph edge approximates the RMSE of the corresponding sets between the two nodes, with a line process weight. Open3D uses function global_optimization to perform pose graph optimization. Two types of optimization methods can be chosen: **GaussNewton** or **LevenbergMarquardt**. The latter is recommended since it has better convergence property. The global optimization performs twice on the pose graph. The first pass optimizes poses for the original pose graph taking all edges into account and does its best to distinguish false alignments among uncertain edges. These false alignments have small line process weights, and they are pruned after the first pass. The second pass runs without them and produces a tight global alignment. In this example, all the edges are considered as true alignments, hence the second pass terminates immediately.

PointCloud in Open3D has convenient operator + that can merge two point clouds into single one. After merging, the points are uniformly resampled.

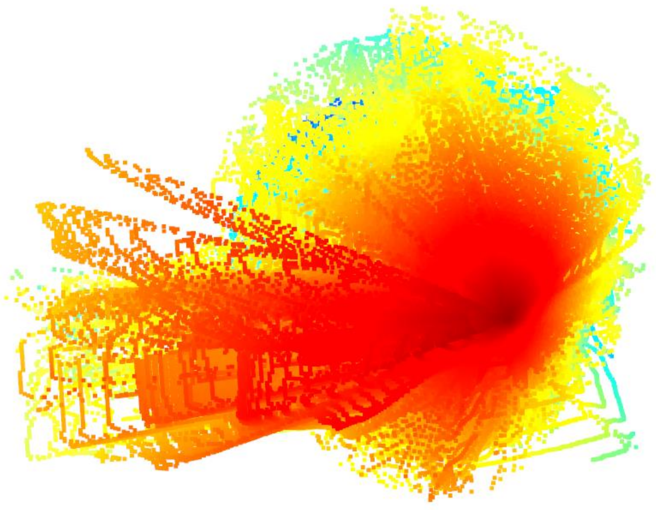

Fig. 16 Application of the Multiway Registration method. The image presents now more points in correspondance of the flower sterm and more sparse point in the center to describe the angulation and profundity

4.3 Triangle Mesh with Poisson Algorithm

Surface reconstruction is an inverse problem and it is performed using a distance function which assigns to each point in the space a signed distance to the surface. Loss of the geometry precision in areas with extreme curvature, i.e., corners, edges is one of the main issues encountered. Since we want to get a watertight and smooth surface we use the "**Poisson Surface Algorithm**".

At first we need oriented normals for all of the points included in the scanned point cloud. Assuming that the scanned points form the surface of the scanned object, the isosurface of the model is approximated from the normalfield that is in relation to the gradient of the indicator function describing the isosurface. This problem is considered as the solution of a poisson problem. At first the points with the oriented normals are taken and the gradient of the indicator function is approximated through the normalfield.

After that the indicator function, which is zero everywhere except near the surface, is derived from this gradient and the surface of the object can be extracted. For this extraction the "marching cubes algorithm" is used to build an octree data structure for the representation of the surface. The marching cubes algorithm devides the point cloud into a voxel grid by marching through the point cloud and analyzing which points define the isosurface of our object. By detecting which egdes of the voxel are intersected by the model's isosurface the algorithm creates the triangles of the mesh. There are several parameters available in Open3D that affect the result of the remeshing:

- Depth: tree-depth that is used for the reconstruction. default: 8
- SolverDivide: specifies depth at which a block Gauss-Seidel solver is used to solve Laplacian equation. default: 8
- IsoDivide: specifies the depth at which a block iso-surface extractor should be used to extract the iso-surface. default: 8
- SamplesPerNode: specifies the minimum number of sample points that should fall within an octree node as the octree construction is adapted to sampling density. For noise-free data: [1.0, 5.0], noisy data: [15.0, 20.0]
- Scale: ratio between the diameter of the cube used for reconstruction and the diameter of the samples' bounding cube. default: 1.25
- Offset: an hacked offset value. if set to 1 there is no offset.
- Confidence: enabling this flag tells the reconstructor to use the size of the normals as confidence information. When the flag is not enabled, all normals are normalized to have unit-length prior to reconstruction.

Especially the parameters depth and samples per node have a great influence on the generated mesh. The higher the value for the octree-depth is chosen the more detailed results you will get. This is because the deeper the marching cubes algorithm goes the finer gets the granularity of the voxelgrid. On the other hand with noisy data (like our scanned point clouds) we keep vertices in the generated mesh that are outliners but the algorithm doesn't detect them as such. So a low value (maybe between 5 and 7) provides a smoothing but you will lose detail.

The higher the depth-value the higher is the resulting amount of vertices of the generated mesh. The samples per node parameter defines how many points the marching cubes algorithm puts into one node of the resulting octree. A high value like 10 means that the algorithm takes 10 points and puts them into the node of the octree. If you have noisy data a high sample per node value provides a smoothing with loss of detail while a low value (between 1.0 and 5.0) keeps the detail level high. A high value reduces the resulting count of vertices while a low value remains them high.

The Poisson Reconstruction is a bit more technical/mathematical. Its approach is known as an implicit meshing method, which I would describe as trying to "envelop" the data in a smooth cloth. Without going into too many details, we try to fit a watertight surface from the original point set by creating an entirely new point set representing an isosurface linked to the normals. There are several parameters available that affect the result of the meshing.

The reconstructor use linear interpolation to estimate the positions of iso-vertices. To get a clean result, it is often necessary to add a cropping step to clean unwanted artifacts highlighted as yellow from the left image. For this, we compute the initial bounding-box containing the raw point cloud, and we use it to filter all surfaces from the mesh outside the bounding-box. The mesh in addition is reduced and cleaned from all degenerate, duplicated triangles, verteces and from non-manifold edges.

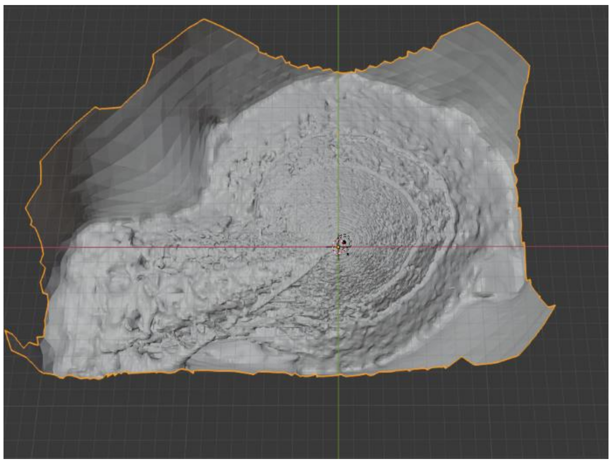
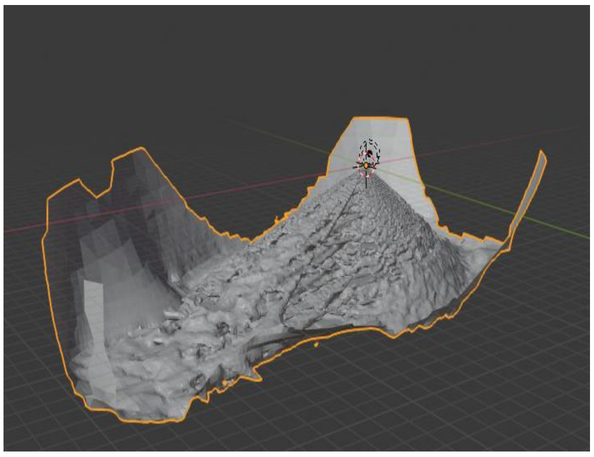

Fig. 17-18 Result of the 3D Reconstruction algorithm in Blender viewport. The borders around the object is one problem due to the Poisson function.

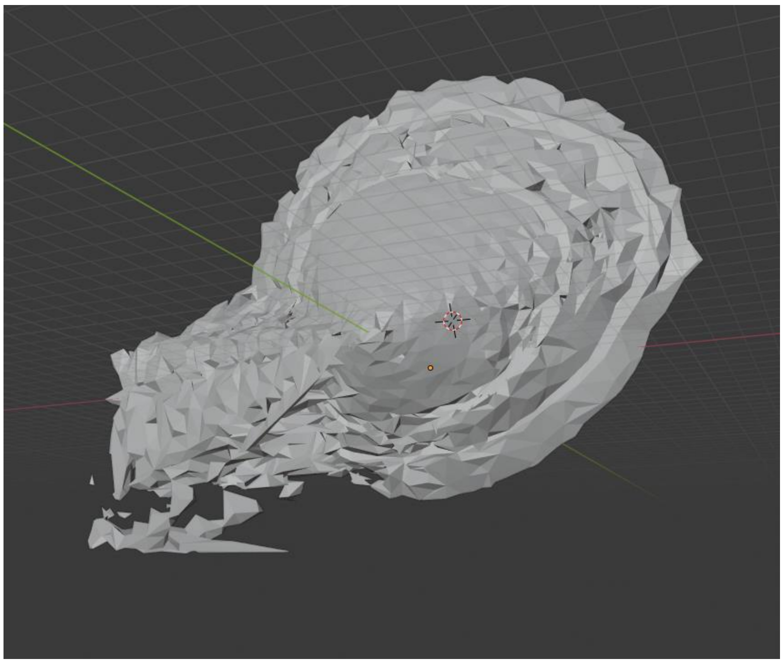

Fig. 19 3D Reconstructed object with clean borders and mirror modifier

5. Conclutions

Approaching a 3D Reconstruction with no camera calibration is complex and it represents a field of study nowadays in the ambit of Computer Vision. Comparing it with a standard approach based on camera calibration we can assert that the algorithm discussed in this paper is more complex computationally speaking, since it uses a lot of calculations to compensate the absence of camera calibration. Especially computing all those images between them to get more information about the 3D shape of the object and trying to rectify and fix them from their distorsion is very expensive.

The Kuwahara Filter seems to work nice and performs very well to initially reduce the noise of the images to apply the ORB feature detection. We can say that probably the use of SURF or SIFT algorithm could improve the speed of the algorithm but they will return a less detailed feature extraction and so the 3D model obtained from the process would have been less detailed.

The depth map calculation is very hard to improve and also different values should be tested considering different objects to reconstruct.

The back-projection using the approximated intrinsic camera parameters return a great number of 3D information which corresponds to the world data. The field of view value is the only thing that can speed up the computation. Greater values corresponds to more points but more points are expensive to manage. Reduce the value too much means an uncorresponded reconstruction.

The Multiway Registration process is the slower part of the algorithm since it envolves I/O operation to read and write the Point Cloud documents stored in the memory. Also here, basing on the object to reconstruct a different list of values for the downsampling has to be choosen.

The Poisson Algorithm is very fast and allows great reconstruction of the 3D shape, but the problem is that in some objects we would have some sort of contour to the mesh due to the mathematical principles behind the algorithm. Since the model can be used with computer graphic software like Maya, Blender etc. to apply some modifications, it can be fixed with automatic reshaping method present in those. For symmetric object applying a "mirror" effect will recreate the part of 3D data the algorithm can't read.

5.1 Future directions

The algorithm proposed is just a "skeleton" of what we can do using Python and libraries in Computer Vision, so a lot of improvements can be developed starting with the code:

- **Texture Mapping**: one different part to achieve in 3D Reconstruction with Multiview Stereo is to reconduce the pixel information in the 2D image to the correspondant 3D point in the model or in the point cloud. Open3D allows to do it but using their structure starting from images.

- **Algorithm Improvements**: the project proposed can surely be structured in a way to improve its speed. One idea could be considering a restricted group of specific images to reduce the workload.

- **Point Cloud**: the size of the point cloud used is around 10MB. Their dimension can be reduced considering less points in feature extraction and in the depth map or using algorithm involving geometry basics or clustering to represent a group of points with just one of them.

- **Segmentation**: to obtain a perfect 3D shape reconstruction of every part of the object, segmentation and machine learning can be used to establish of how many parts and which parts represent the object and proceed with a subdivision process to allow a reconstruction of single small parts to be then combined.

Bibliography and sitography

(1)

R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision* (2004)

(2)

D. Baggio and S. Emami *Mastering OpenCV 3* (2008)

(3)

J. Howse, P. Joshi, M. Beyeler *OpenCV: Computer Vision Projects with Python* (2016)

(4)

https://en.wikipedia.org/wiki/3D_reconstruction_from_multiple_images

(5)

https://en.wikipedia.org/wiki/Kuwahara_filter

(6)

https://it.mathworks.com/matlabcentral/fileexchange/8171-kuwahara-filter

(7)

https://globlib4u.wordpress.com/2013/03/01/structure-from-motion-and-3d-reconstruction-on-the-easy-in-opencv-2-3-w-code/

(8)

https://becominghuman.ai/stereo-3d-reconstruction-with-opencv-using-an-iphone-camera-part-i-c013907d1ab5

(9)

https://towardsdatascience.com/inverse-projection-transformation-c866ccedef1c

(10)

https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba

(11)

http://www.open3d.org/docs/release/

(12)

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_table_of_contents_calib3d/py_table_of_contents_calib3d.html

(13)

https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

(14)

https://shkspr.mobi/blog/2018/04/reconstructing-3d-models-from-the-last-jedi/

(15)

https://github.com/mapillary/OpenSfM

(16)

https://www.cc.gatech.edu/~hays/compvision/proj3/

(17)

https://realitybytes.blog/2018/05/10/computer-vision-perception-structure-from-motion/

(18)

https://towardsdatascience.com/inverse-projection-transformation-c866ccedef1c

(19)

http://www.maths.lth.se/media11/FMA270/2017/forelas6.pdf

(20)

https://lutpub.lut.fi/bitstream/handle/10024/124031/kandidaatintyo_rapo_lauri.pdf?sequence=2

(21)

https://cs.adelaide.edu.au/~wojtek/papers/iaif97Improved.pdf

(22)

https://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/CS231a-FinalReport-sgmccann.pdf