



**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Structure from Motion

Turntable Based Reconstruction

**Thomas Rootwelt**

Master of Science in Computer Science

Submission date: July 2014

Supervisor: Theoharis Theoharis, IDI

Co-supervisor: Igor Barros Barbosa, IDI

Norwegian University of Science and Technology  
Department of Computer and Information Science



### **Abstract**

In this paper I explore the possibility of using a camera, a turntable, and a computer in place of a 3D-scanner in order to acquire usable 3D models of small objects. By using a basic Structure From Motion approach, I will show that it can easily be adapted to produce decent 3D models, with little to no user input, as long as the objects are sufficiently textured.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Task . . . . .	1
1.2	Structure From Motion . . . . .	2
1.3	Related Work . . . . .	4
<b>2</b>	<b>Description</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Camera model and calibration . . . . .	5
2.2.1	Theory . . . . .	6
2.2.2	Code . . . . .	7
2.3	Feature detection and matching . . . . .	8
2.3.1	Theory . . . . .	8
2.3.2	Implementation . . . . .	11
2.4	Epipolar geometry . . . . .	13
2.4.1	Theory . . . . .	13
2.4.2	Implementation . . . . .	14
2.5	Triangulation . . . . .	15
2.5.1	Implementation . . . . .	15
2.6	Bundle adjustment . . . . .	16
2.6.1	Theory . . . . .	16



2.6.2	Implementation . . . . .	16
2.7	Specializations for Turntable approach . . . . .	16
2.8	Post reconstruction steps . . . . .	21
<b>3</b>	<b>Results</b>	<b>22</b>
3.1	Refining and combining the solutions . . . . .	22
3.1.1	SFM output . . . . .	22
3.1.2	Refinement . . . . .	25
3.1.3	Combining solutions . . . . .	26
3.2	Assessing the solution . . . . .	29
3.2.1	Visual assesement . . . . .	29
3.2.2	Mathematical assesement . . . . .	32
<b>4</b>	<b>Further Work</b>	<b>34</b>
4.1	Scale . . . . .	34
4.2	Isolation . . . . .	35
4.3	Combination . . . . .	35
4.4	Refinement . . . . .	36
<b>5</b>	<b>Conclusion</b>	<b>37</b>
	<b>Bibliography</b>	<b>39</b>

# List of Figures

1.1	Setup of camera rig and turntable . . . . .	2
2.1	Camera model showing the camera center ( $C$ ), the camera plane, the principal point ( $PP$ ) and how a point ( $P$ ) is projected on the camera plane ( $P'$ ). It also shows the focal length ( $F$ ) . . . . .	6
2.2	Calibration using a known object, a chessboard, where the inner corners are tracked	7
2.3	Example images used for calibration . . . . .	8
2.4	2-directional filter . . . . .	9
2.5	FAST feature detection . . . . .	10
2.6	FREAK descriptor sample pattern . . . . .	11
2.7	Detected keypoints . . . . .	12
2.8	Matches between two frames . . . . .	12
2.9	Epipolar geometry of two cameras showing both camera centers $C1$ and $C2$ , a point $P$ and projections $P1$ and $P2$ , as well as the epipoles $E1$ and $E2$ . The line between epipoles and projected points is the epipolar line. . . . .	13
2.10	Initial matches . . . . .	17
2.11	Matches after removing those that do not move (those outside the turntable) . .	17
2.12	Matches that have moved, and that make epipolar sense . . . . .	17
2.13	Cameras and item will have this configuration . . . . .	18
2.14	Brute force matching . . . . .	19

---

2.15	Closest neighbors only matching . . . . .	20
2.16	N-Closest neighbor matching . . . . .	21
3.1	Item 1 images . . . . .	22
3.2	Cameras are represented by lines in the direction of view . . . . .	23
3.3	Cameras from a different angle . . . . .	23
3.4	Item 1 Side 1 . . . . .	24
3.5	Item 1 Side 2 . . . . .	24
3.6	Item 1 Side 1 Segmented . . . . .	25
3.7	Item 1 Side 2 Segmented . . . . .	26
3.8	Item 1, composed from both sides . . . . .	27
3.9	Item 1, ground truth provided by 3D-scanner . . . . .	27
3.10	Item 2 images . . . . .	28
3.11	Item 2 reconstruction . . . . .	28
3.12	Item 2 ground truth . . . . .	29
3.13	Cross section of one surface of Item 1 . . . . .	30
3.14	Cross section ground truth . . . . .	30
3.15	The polished side of Item 2, with few points besides the edges. . . . .	31
3.16	Polished side from 3D-scanner . . . . .	31
3.17	Item 1, reconstruction and ground truth combined . . . . .	32
3.18	Item 2, reconstruction and ground truth combined, note the inaccurate reconstruction of the bottom left corner . . . . .	33
4.1	Reconstruction showing a known distance . . . . .	35

# Chapter 1

## Introduction

### 1.1 Task

The purpose of this project is to explore the possibility of using Structure From Motion (SFM) as a replacement for 3D-scanners. By using sufficiently advanced software and relatively cheap cameras, it is hoped that expensive 3D-scanners can be avoided. I will look into how feasible this is, and attempt to discern whether or not it provides an adequate solution. For testing I use video provided by Presious (<http://presious.eu/>), a European archeological program by IDI, and corresponding 3D models as ground truth.

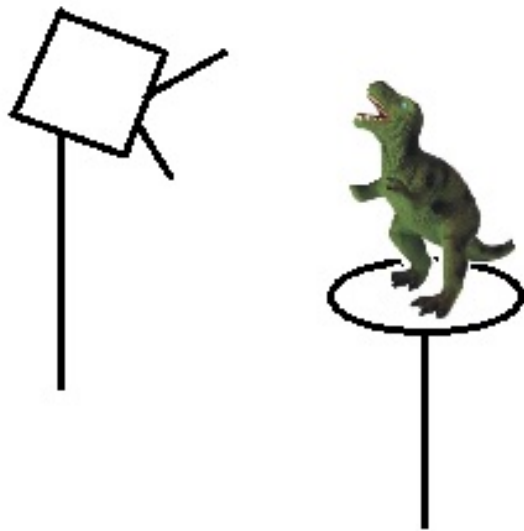


Figure 1.1: Setup of camera rig and turntable

## 1.2 Structure From Motion

Structure From Motion is a technique used to recreate a 3D structure from a series of 2D images taken by a sensor, such as a camera moving along a path. The algorithm takes as input a sequence of images and outputs a series of 3D points, called a point cloud. It is a specific case of a problem class called multiple view geometry. There are specialized versions created specifically for turntables based reconstruction, but this paper focuses on adapting a general SFM approach for the task, which normally follows a common sequence of steps.

### 1. Camera calibration:

Real cameras have imperfections, such as distortions caused by the lens, non-square pixels and, more importantly, they have a (varying) focal length. Cameras need to be calibrated as part of SFM, and this is usually done before the reconstruction itself starts.

### 2. Identify points of interest:

SFM outputs a series of points in 3D. These points need to be located in the 2D images

before their 3D position can be found. The first step is to identify points that are easily recognised. These points are usually "corners" or other features that can be pinpointed in two dimensions.

### 3. Track points:

Once points of interest have been found in two different images, they need to be matched. That means identifying which point in each image is actually the same 3D point. In structure from motion one can assume that the camera only moves a little bit, so that each point is will lie close to its match in the next image in the sequence.

### 4. Calculate camera movement:

Based on the movement of the tracked points from one image to another, the relative movement and rotation of the camera can be calculated, given enough point correspondences. This is encapsulated in something called the fundamental matrix  $F$ , or in the case of fully calibrated cameras, the essential matrix  $E$ .

### 5. Triangulate points:

Once the relative position of the two cameras is known, the 3D points can be triangulated. The basic principle is drawing a line from each camera center through the 2D image points (on a plane in 3D). These lines will intersect in the 3D point.

### 6. Bundle adjust:

Using two and two images is one way to get 3D information, but in structure from motion we have video. Using information from multiple frames to get a more accurate estimation of the 3D points is called bundle adjustment.

Unfortunately, in the real world the data would contain noise. Each of the outlined steps is performed using techniques that attempts to minimize error given the noisy data. Most SFM algorithms make some assumptions as well, such as the scene remaining static, the camera intrinsic parameters staying the same ( e.i. not zooming ), or it might make assumptions on the movement of the camera. An important part of SFM is therefore error checks and robust algorithms.

## 1.3 Related Work

Structure from motion is a large field, and there exists a lot of relevant work [8]. Zisserman's *Multiple view geometry* [6] is considered a sort of "bible" in the field. Turntable-based approaches [5] have also been explored, and I have used information on OpenCV [2]. The algorithm described in this paper is based closely on Chapter 4 of *Mastering OpenCV with Practical Computer Vision Projects* [4].

# Chapter 2

## Description

### 2.1 Introduction

This part describes how the code works, and why I have chosen to implement it the way I have. It also describes the underlying mathematical principles used. For simplicity, I assume all images are grayscale. Even though I solved a special case of structure from motion (turntable), I was recommended to implement a standard SFM algorithm. This would then be configured to work with the turntable based images. It was also decided early on that I should use the OpenCV library. I learned last year, however, that working with OpenCV presents some challenges. When implementing a SFM program for last years project, I had hoped to create a rough but working prototype. Instead, I got an implementation that didn't work at all. As a consequence, this year I decided to follow an existing approach [4] [2] more closely, to avoid the more obvious pitfalls of OpenCV.

### 2.2 Camera model and calibration

In order to use images as a representation of the real world, we need a precise, mathematical model of how each image corresponds to the scene. To do this, I use the pinhole camera model and a distortion vector, and calibrate it for the camera used in the reconstruction.



## 2.2.1 Theory

### Model

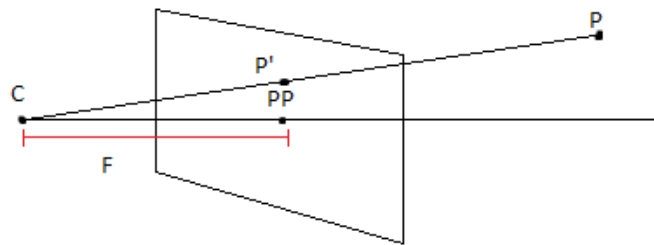


Figure 2.1: Camera model showing the camera center ( $C$ ), the camera plane, the principal point ( $PP$ ) and how a point ( $P$ ) is projected on the camera plane ( $P'$ ). It also shows the focal length ( $F$ )

The camera model encapsulates both the extrinsic and intrinsic parameters of the camera. The extrinsic parameters consist of the position and pose of the camera in world coordinates (a total of 6 parameters), while the intrinsic parameters consist of focal length, pixel dimensions, skew, and the location of the principal point (5 parameters). The extrinsic parameters change as the camera moves, but the intrinsic parameters only change when we zoom the camera (changing the focal length). If we keep the camera at a constant zoom we can then find values for the intrinsic parameters of a given camera. Having a calibrated camera will move our reconstruction from projective to metric space.

### Camera calibration

If we had a single image of a scene where we knew the 3D and 2D coordinates of at least 6 points, we could calibrate the camera (assuming a non-degenerate solution). This is because each 2D point provides 2 values and we have 11 parameters to solve for. However, we rarely know the 3D coordinates of the points in the image, so we need another approach.

One attractive solution is to automatically calibrate the camera as we reconstruct the scene. This does not require prior knowledge of the camera. However, according to Zisserman 19.9 [6], if the camera moves around a single axis (as is the case with turntable reconstruction), we will not get a metric solution, but a special case of a projective one.

Another approach is to use a known object to calibrate the camera. This approach uses multiple shots of a known object to robustly estimate the intrinsic parameters of the camera.

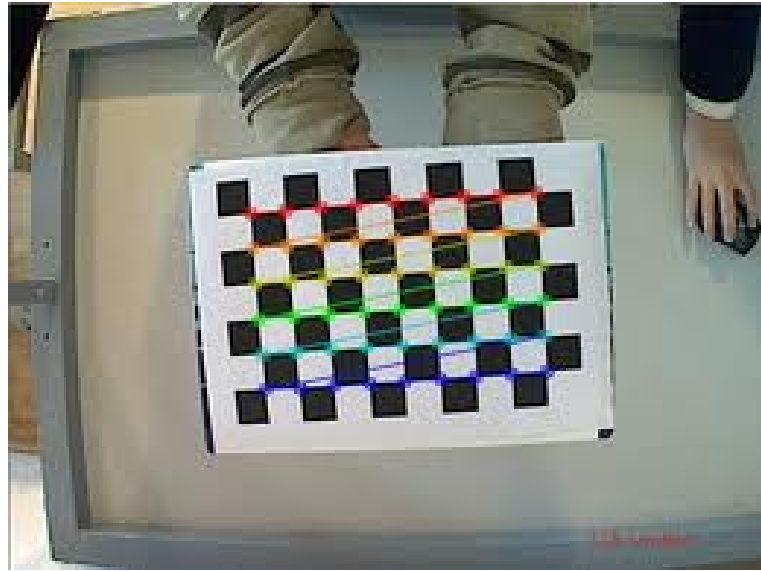


Figure 2.2: Calibration using a known object, a chessboard, where the inner corners are tracked

### 2.2.2 Code

Since auto-calibration is hard given the movement of the camera, I decided to calibrate using a known object. I used OpenCV's built in solver for a chessboard pattern, contained in a separate program. This calibration program outputs an xml file that is read by the SFM program. Just provide a number of images (preferably more than 10) of a chessboard and edit the input xml-file and run the calibration program. This step can be done by any camera calibration technique, and if no input is given to the SFM algorithm, moc-up values will be used. This might be sufficient to give decent results if a bundle adjuster that improves the internal camera matrix is used.

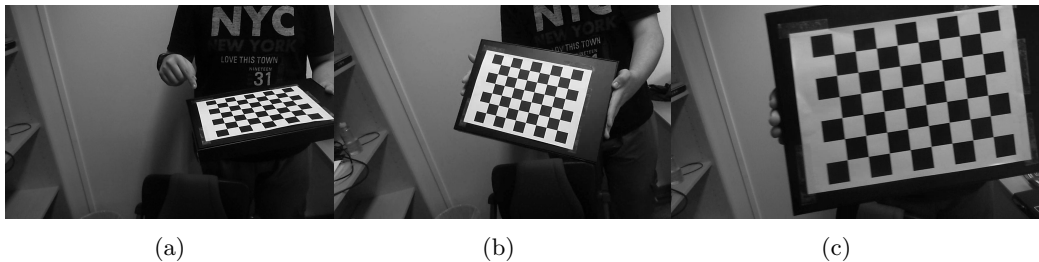


Figure 2.3: Example images used for calibration

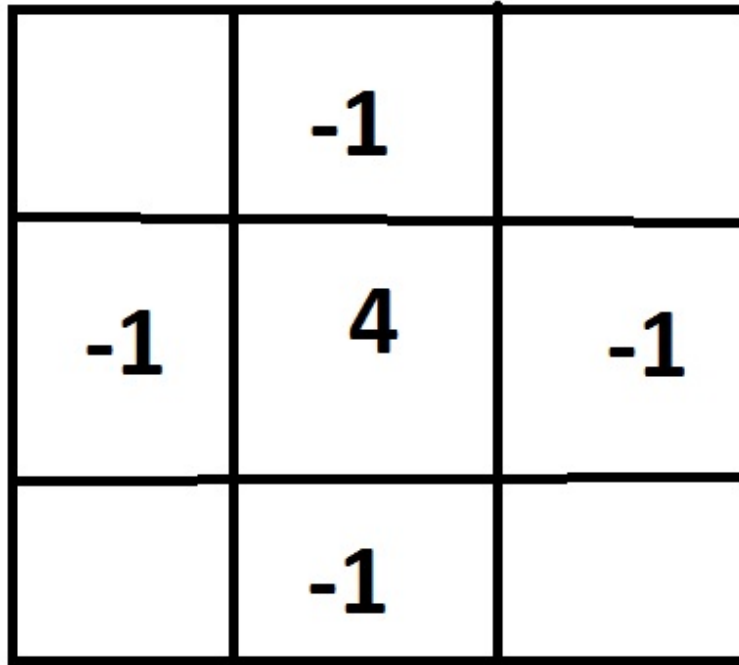
## 2.3 Feature detection and matching

The first operation that should be done on the images, once they have been loaded, is to find suitable points to work with, and calculate descriptors for each point. We can then work with points and camera positions, instead of images.

### 2.3.1 Theory

#### Feature Detection

Since we work with points, we need a way to detect them in an image. Not all pixels in an image can be used as points, as we want something we can find in another frame as well. Imagine keeping track of all points on a white piece of paper, differentiating between all pixels would be close to impossible. However, finding the corners is relatively easy, and it turns out almost every "point" in an image can be thought of as a corner. To understand what defines a point, let me first define a line. A line can be thought of as a sudden shift in intensity, and can be found by applying finding the derivative in all directions, for example by using a  $(-1, 1)$  filter. A corner can be defined as simply a line, or gradient, in two directions.



	-1	
-1	4	-1
	-1	

Figure 2.4: 2-directional filter

This filter is just an example, and would make a bad feature detector. The first order derivatives would for example also detect gradients. More advanced methods find points with higher accuracy and find more of them more efficiently. One of these method is called FAST [9] [10].

## **FAST**

FAST stands for Features from Accelerated Segment Test, and is a feature detector that fuses different methods to find corners. It uses a circle of 16 pixels around a candidate corner, and if a certain continuous section of this (the size of this section is normally determined by machine learning) is either brighter or darker than the center by a certain threshold, the pixel is considered a corner. This test is constructed in such a way that non-corners can be rejected quickly using a high speed test. Machine learning is an important part of FAST, used to tweak the parameters and speed up the detection.

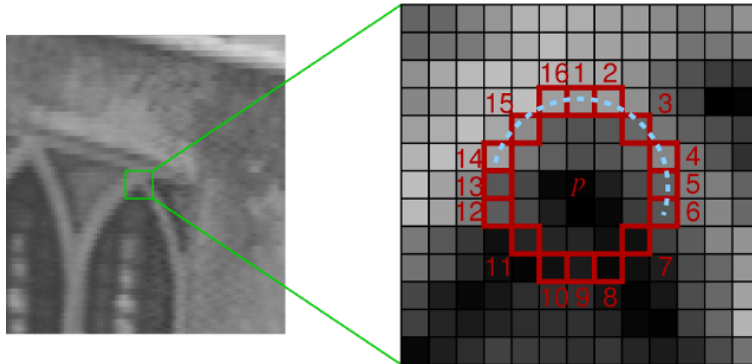


Figure 2.5: FAST feature detection

### Feature Matching

Once we have a set of points, we want a way to match them to points in other images. The first step is calculating a descriptor for each point. The descriptor should be something that does not change much from frame to frame. An example of a descriptor could be the intensity of the neighbouring pixels. The assumption is that this doesn't change much as the camera moves. Points can also be matched by spatial locality, in which case the descriptor would contain information about where the point was found in the image. This works better in SFM as each point does not move much from frame to frame. To match two images, we simply compare all the descriptors in one image all the descriptors in the other. The most similar descriptors will be matched.

### FREAK

FREAK [1] (Fast Retina Keypoint), is a relatively new method for extracting keypoint descriptors. The descriptor is a Local Binary Pattern, meaning that it is a vector of bits describing a relationship between pairs of regions, in this case, the sign of the integral between two regions. FREAK, unlike earlier methods, provide finer resolution closer to the corner than further away. This is similar to the human eye, and this is where the method got its name. This provides a highly efficient yet easy to compute descriptor.

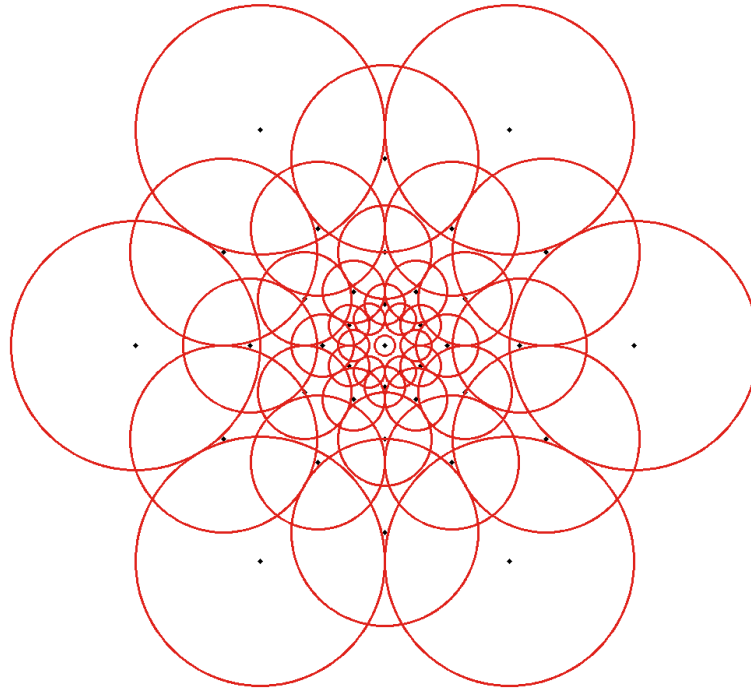


Figure 2.6: FREAK descriptor sample pattern

### 2.3.2 Implementation

I have tested with various feature matchers and descriptor extractors, attempting to find as many points as possible. Because my algorithm does not have any serious time constraints, I use those OpenCV feature detectors and descriptor extractors that yield better and more points, at the expense of time. I ended up settling on a "PyramidFAST" detector with "FREAK" extractor, but this can be replaced with other rich feature matchers, or optical flow matchers. "Pyramid" means the detector works on a downscaled version of the image, and works its way up.



Figure 2.7: Detected keypoints

Up next is matching points between two images. I use a brute force matcher that will match each point in the first image to the best match in the second. Obviously, this might result in multiple matchings to some of the points in the second image. I remove these multiple matchings using only the first match found. Another option that seems to give better results, at the expense of time, is matching from image 1 to image 2, and vice versa, and only keep matches that agree with both. I now have a series of probable matches, but not all of them will make epipolar sense (see 2.4). On large images, this is an expensive calculation, and involving the GPU might yield huge performance improvements.

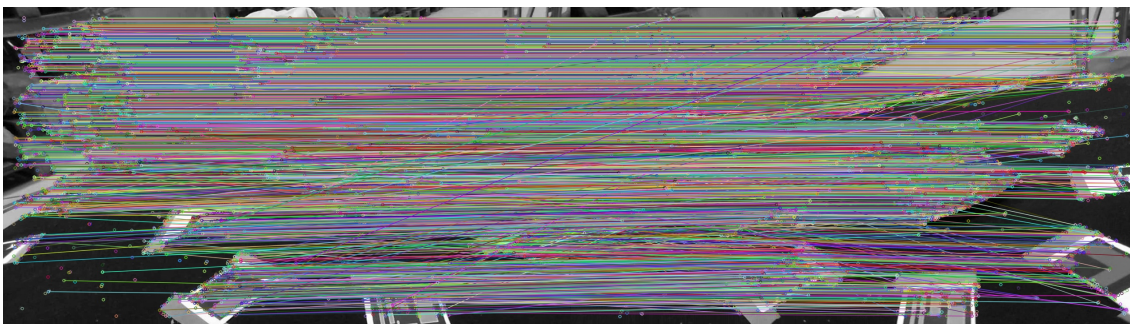


Figure 2.8: Matches between two frames

## 2.4 Epipolar geometry

In order to recreate a scene from images, we need to know where the cameras are. This can be calculated using matching image points and is described by the epipolar geometry of the two cameras. Importantly, the epipolar geometry does not depend on the scene, and we only need 7 image point pairs to compute it.

### 2.4.1 Theory

Epipolar geometry defines the relationship between two cameras and a scene. In structure from motion, we don't have two cameras, but each frame of video can be considered a separate camera as long as the scene remains static. Note that epipolar geometry applies to two cameras (although it can be extended to three or four), and we will add more cameras using a different technique. Therefore, the discussion here will be limited to two cameras, a set of point matches, and an unknown scene.

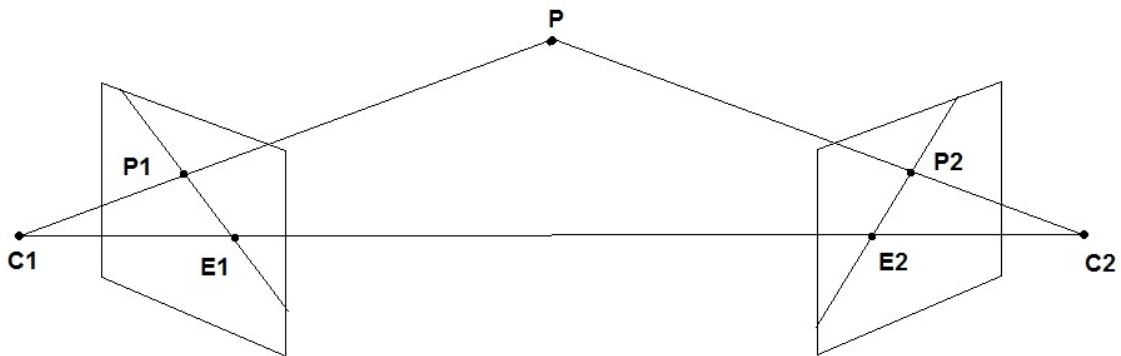


Figure 2.9: Epipolar geometry of two cameras showing both camera centers  $C1$  and  $C2$ , a point  $P$  and projections  $P1$  and  $P2$ , as well as the epipoles  $E1$  and  $E2$ . The line between epipoles and projected points is the epipolar line.

To describe epipolar geometry, we introduce the Fundamental matrix ( $F$ ). In essence, the Fundamental matrix relates points in one image to points in the other by the following equation.

$$X^T F X' = 0$$

The Fundamental matrix can be explained by considering 2.9. If we draw a line from one camera center, and extend it through a 2D point on the image plane, that line represents all possible locations of the corresponding 3D point in the scene. The image of this line on the other image plane is called an epipolar line. More precisely, this line is the one between the



epipole and the 2D point in this image. The epipole is the "image" of one camera center as seen from the other camera.

$$\forall x x F = l'$$

$$\forall x' x' F^T = l$$

What is important to us, is that once we find  $F$ , we can use it to find the relative position of the two cameras. So what do we need to calculate it?

$F$  is a 4x4 matrix with 7 degrees of freedom in the general case, and each pair of points will give rise to one linear equation, meaning we need (at least) 7 point pairs to calculate a solution, given that points are not in a degenerate configuration.

Once we have  $F$ , we can calculate the Essential matrix  $E$  (Z9.6). The essential matrix is basically  $F$ , but normalized using the intrinsic camera matrix  $K$ .

$$E = K^T F K$$

Once we have  $E$ , we can use Single Value Decomposition to extract the relative rotation and translation of the cameras. Note that  $E$  only has 5 degrees of freedom, while we should have 6 for rotation and translation. This is because the translation is on an arbitrary scale. Intuitively, this is because scale is hard to extract without knowledge of the scene. Also, we have a four-fold ambiguity, meaning we get 4 solutions, where one is correct.

If we already have a set of 3D points, then we can estimate the position of subsequent cameras in a different way. By re-projecting 3D points and comparing them to their corresponding 2D points on a new image plane, the camera position will be the one where the error is minimized. This can be thought of as the same process as walking around your house with a picture to find where it was taken.

## 2.4.2 Implementation

As a final step in feature matching between each image pair, I calculate the Fundamental matrix using a robust method (RANSAC). Only matches that agree with the Fundamental matrix are included for reconstruction. Feature matching is done with between all image pairs if we do not assume anything about the order of the images. This is obviously an  $O(n!)$  algorithm, meaning it rises very quickly when the number of images goes up. This is fine if relatively few images are used, but wildly impractical in other situations. In 2.7 I look at modifications to fit the turntable approach.

## 2.5 Triangulation

Once we have two, or more, camera positions (potential or otherwise), we can triangulate the 3D points. In theory, we just extend the line from each camera center through each matching image point in their respective frame. The lines will intersect in the 3D point. In practice the lines won't intersect, and we try to find the position that minimizes the reprojection error for all cameras.

### 2.5.1 Implementation

I start by getting the baseline triangulation. I select a pair of cameras with a high number of matches that can not be described by a simple homography (to avoid degenerate cases). Then I calculate the fundamental matrix (F) based on the matches, and subsequently the essential matrix (E) from F and the camera matrix K. This matrix should be acquired beforehand using a calibration program such as Matlab or OpenCV. The essential matrix is then decomposed into a rotation matrix and a translation vector. However, since both the rotation matrix and translation vector can be decomposed in two different ways, I need to test 4 different combinations to find the right one. The right decomposition will be the one where the triangulated points are in front of both cameras, and the reprojection error is not too big. I triangulate points for all ambiguities until I find the right one. All 3D points are then set as the initial point cloud, the first camera is set as the center (0,0,0), facing positive z direction, and the second is set to the decomposed translation vector and rotation matrix.

I now have two cameras, a defined coordinate system with scale, and a point cloud. All cloud points have a list over which 2D points that represent them in each camera. This is important for bundle adjustment later. Now I go through all remaining cameras, adding them one by one, always selecting the one that has the most matches already in the cloud. The first step is to estimate the pose of the camera. I need a set of cloud points that correspond to a set of 2D points in the camera, and use this information to estimate the pose. Now I triangulate points from this camera to all cameras already added. New cloud points are added, while existing ones are updated with information of which cameras can see them.

## 2.6 Bundle adjustment

The final step of most SFM approaches is bundle adjustment. It involves using all cameras and all points to calculate a better solution.

### 2.6.1 Theory

Bundle adjustment refers to the bundle of light rays from each point to the camera centers, adjusting them so the mean reprojection error is as small as possible, represented by a large number of non-linear equations. These equations relate all points and cameras to each other. Huge computational benefits can be achieved by observing that each point does not affect the others, and that the cameras do not affect each other. This sparse structure allows huge datasets to be solved relatively quickly. One popular implementation is the Levenberg-Marquardt algorithm.

### 2.6.2 Implementation

Now I have initial estimates for camera positions and points, and I can run bundle adjust to refine the solution. Bundle adjust may be thought of as the workhorse of my implementation. The other parts of the reconstruction, except perhaps the feature matching, are mainly there to set up a decent initial guess. I therefore run the bundle adjuster after adding each camera. There are several great implementations of bundle adjust available, I chose "Multicore Bundle Adjustment" [3] for the possibility of using the GPU, but this can easily be swapped for "SSBA" [11], "SBA" [7], or the OpenCV implementation. A drawback of "Multicore Bundle Adjustment" is that it will not improve the internal camera matrix.

## 2.7 Specializations for Turntable approach

The implementation so far describes a basic SFM approach, and this section describes the changes made to adapt it to Turntable SFM.

One very important step in turntable feature matching is removing points that are part of the background. When filming the turntable, part of the image might include the background, and this will stand still while the turntable turns. If points in the background are included in the calculation of the F matrix or triangulation we will get poor or useless results. I therefore filter matches that have not moved between images before calculation of F.



Figure 2.10: Initial matches

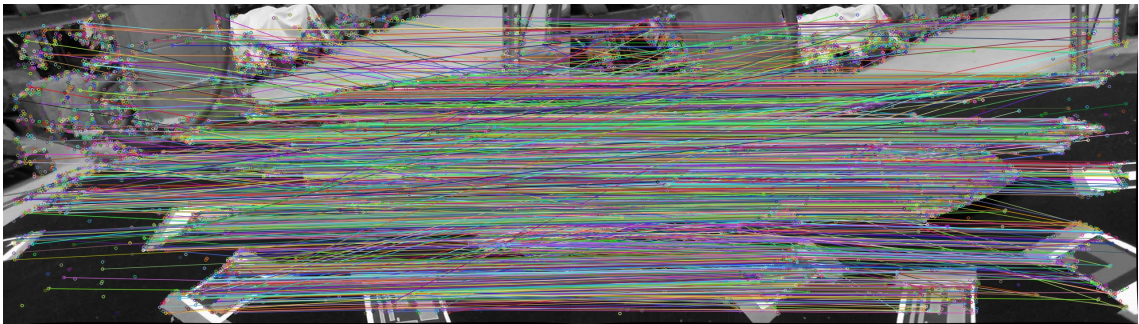


Figure 2.11: Matches after removing those that do not move (those outside the turntable)

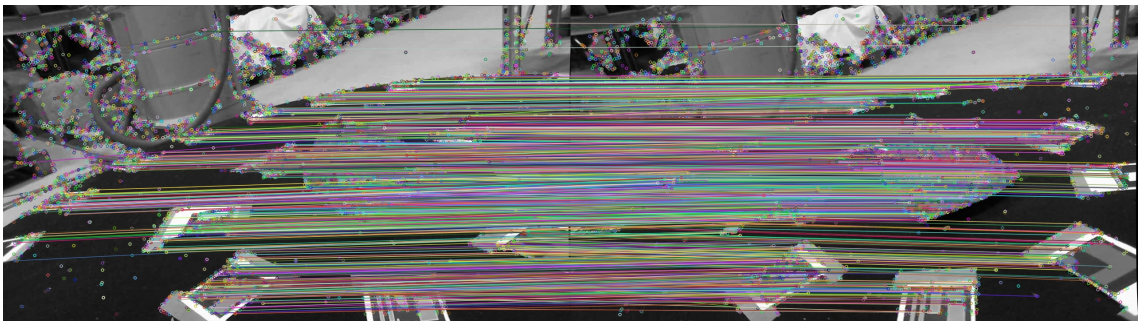


Figure 2.12: Matches that have moved, and that make epipolar sense

When choosing which cameras to match with each other, we have a few different options. We assume a camera and item configuration as below.

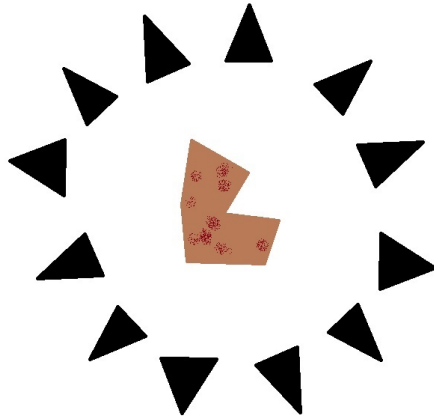


Figure 2.13: Cameras and item will have this configuration

The standard approach does not assume any order to the images, and uses a brute force matching that is very expensive (this an  $O(n^2)$  algorithm) and results may be poor. Matches between images from opposite sides will not yield any useful results and will most likely only add false points.

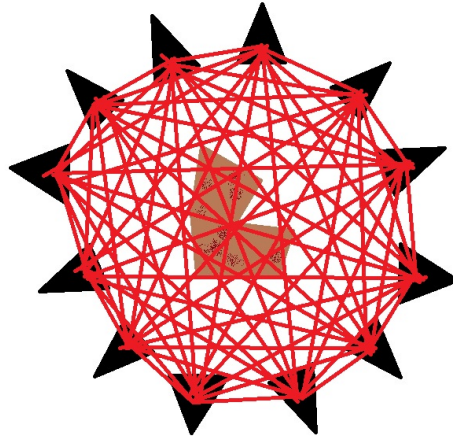


Figure 2.14: Brute force matching

On the other hand, we could just match each image to its immediate neighbors. This would be a lot quicker, but would not use information from more than two cameras for each point, limiting accuracy and adding fewer points. Note that the first and last cameras are not matched to each other, as we do not assume anything about the pattern of images.

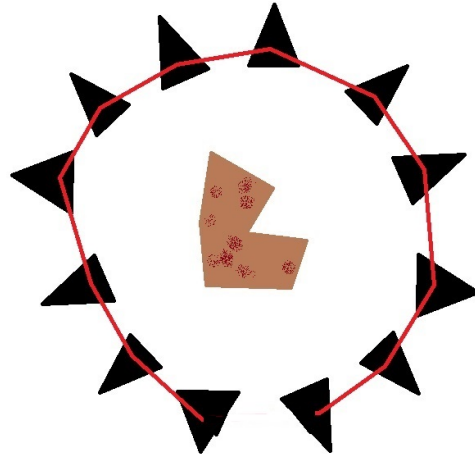


Figure 2.15: Closest neighbors only matching

I ended up using a hybrid solution. I select an image sequence that consist of one full rotation and around 60 images. I set the last image to be the predecessor of the first and match each image with  $N$  of its closest neighbors either through manual matching (matching A to B, B to C, A to C and so on) or "chaining" matches (matching A to B, B to C, and match A to C by taking the matches in common between AB and BC). I found that both of these gave better results than pure Brute Force, and chained matching has a better run time, (especially with large images). This approach has some drawbacks, though, such as the reconstruction failing if I have a sequence of bad images, which would cause the reconstruction to stop.

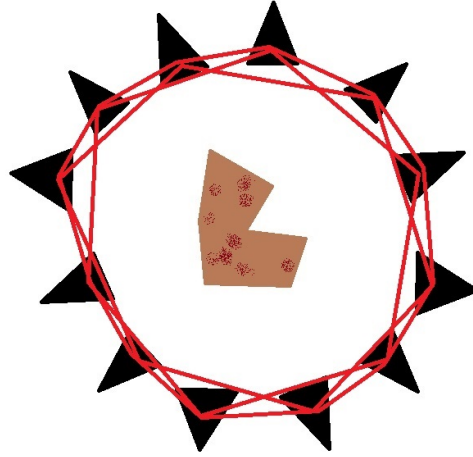


Figure 2.16: N-Closest neighbor matching

Once the reconstruction is complete, I take steps to improve my solution by removing points behind every camera and far from the centroid.

## 2.8 Post reconstruction steps

Once the reconstruction is done for both side of the object, we need to take a few steps in order to complete the 3D object so that it can be compared to the ground truth or used in other applications. I use CloudCompare ( <http://www.danielgm.net/cc/> ) for this.

Firstly, the reconstructed object needs to be separated from the turntable itself, and the noise. This can be done by simply segmenting the reconstruction visually. Secondly, both sides of the object need to be merged into a single object. By manually selecting 3 pairs of corresponding points, the clouds can be aligned, and an error minimizing scheme (registering) can be used to closely match the clouds. We now have a complete 3D object.



## Chapter 3

# Results

The results can be considered in two different ways. Firstly, the 3D models can be considered on their own, as in how useful are they in various applications. Secondly, how do the results compare to those of 3D-scanners, and are there any valid reasons to use SFM when a 3D-scanner is unavailable. Using a laptop with a webcam is cheaper and simpler than using a 3D-scanner, and a laptop is more easily available.

### 3.1 Refining and combining the solutions

#### 3.1.1 SFM output

Lets start with Item 1, and the steps leading from a partial reconstruction to a full one.

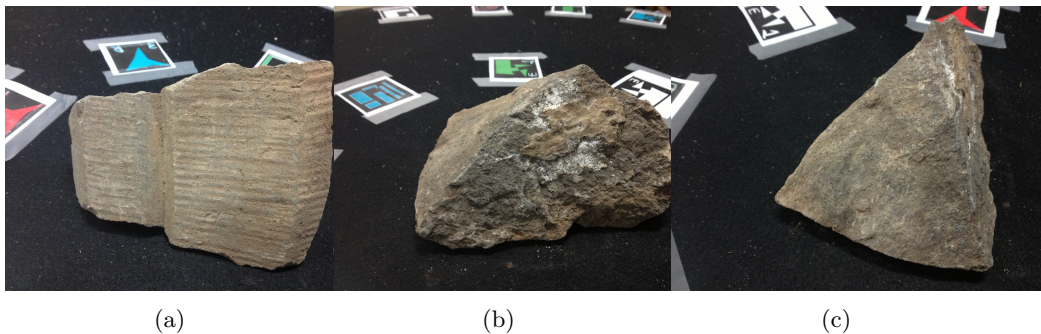


Figure 3.1: Item 1 images

In order to verify the reconstruction, and to help with debuffing, it's useful to be able to see not just the points, but also the cameras. To do this I use a homemade renderer that displays the points, as well as the cameras and the direction they are facing (the direction vector is slightly bugged).

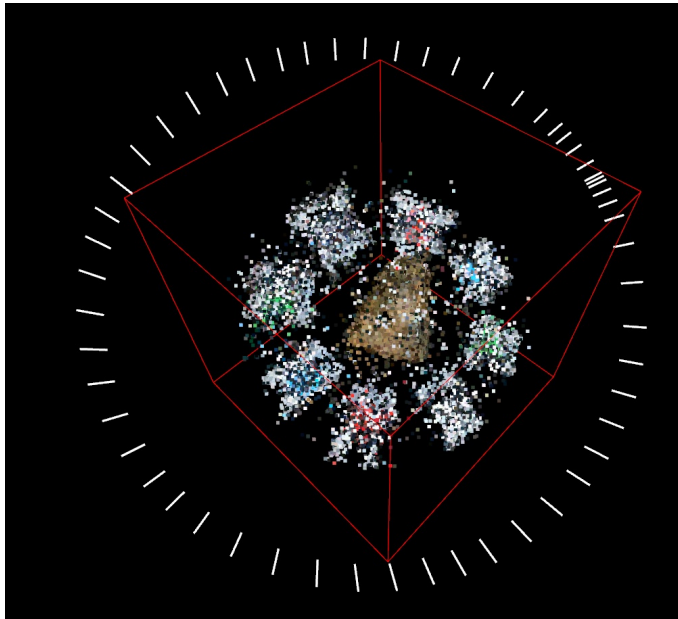


Figure 3.2: Cameras are represented by lines in the direction of view

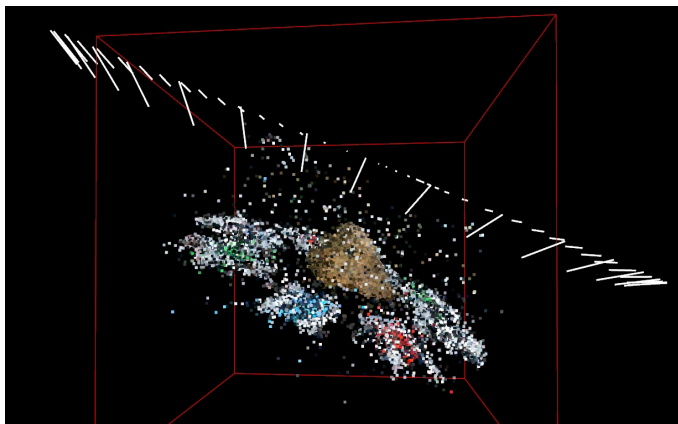


Figure 3.3: Cameras from a different angle

As previously stated, we need to combine reconstructions of both sides of an object. I call the two reconstructions Side 1 and Side 2. The following renderings and operations are performed in CloudCompare.

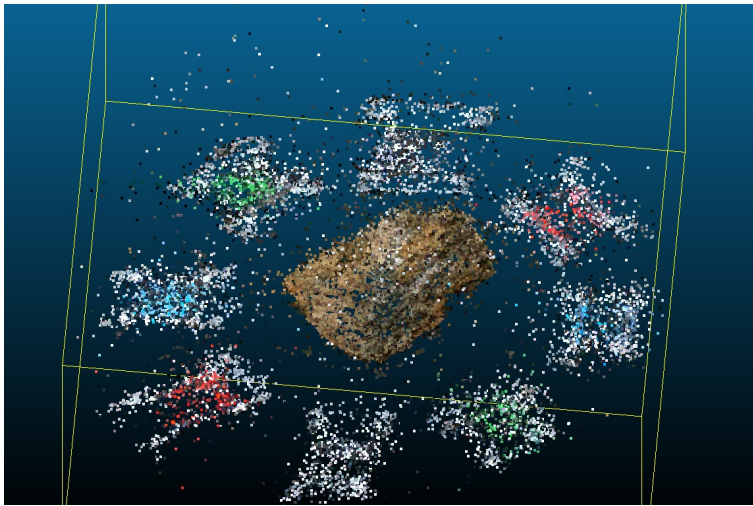


Figure 3.4: Item 1 Side 1

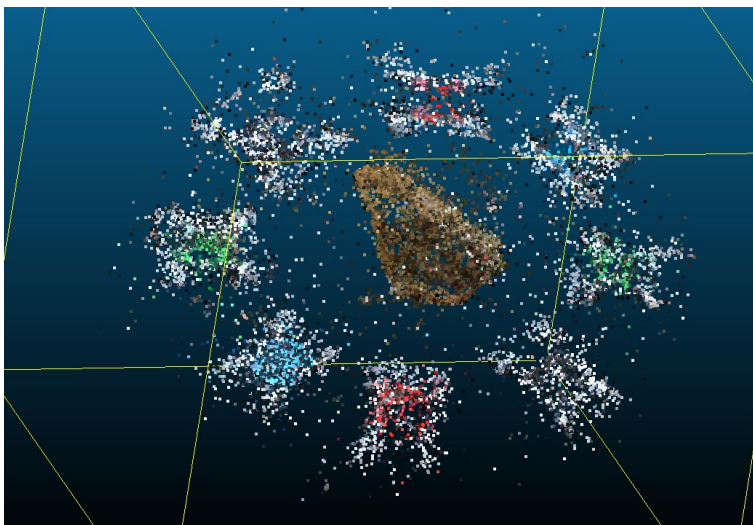


Figure 3.5: Item 1 Side 2

### 3.1.2 Refinement

There are two components to these reconstructions, the central object and the turntable itself. We are only interested in the central object, but before we do this, we can use our knowledge of the specific figures on the turntable to calibrate the scale of the reconstruction. This is not currently implemented in my solution, and the turntable is removed without fixing a scale. This means that the reconstruction is in an arbitrary scale.

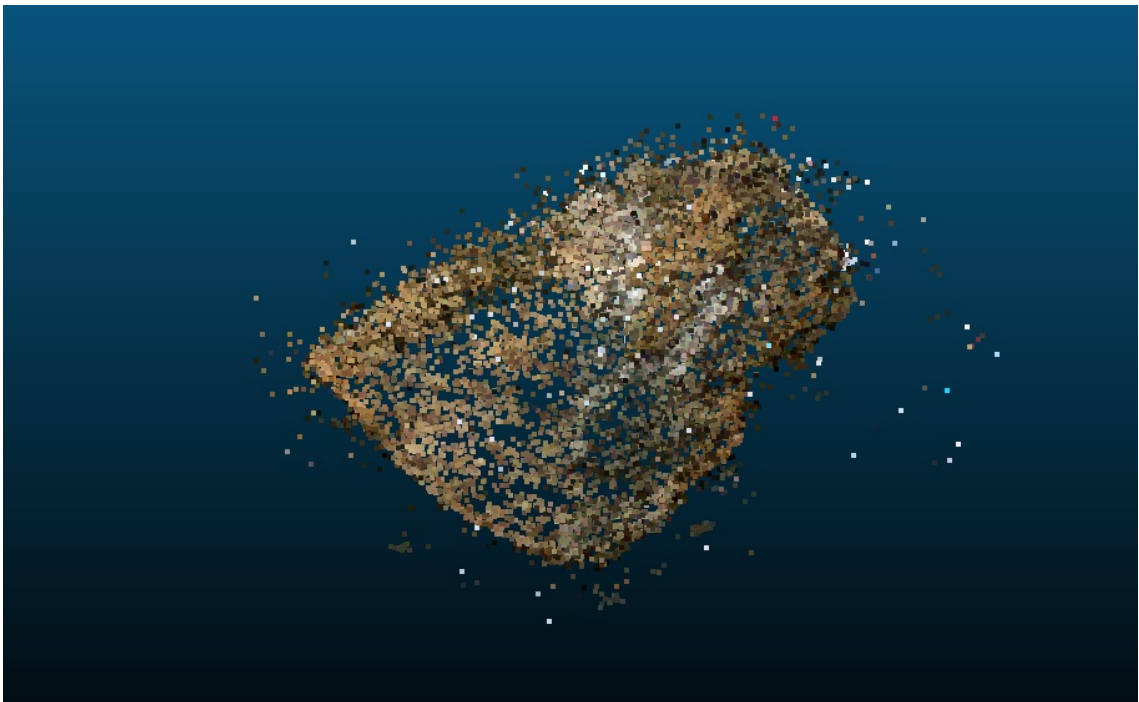


Figure 3.6: Item 1 Side 1 Segmented

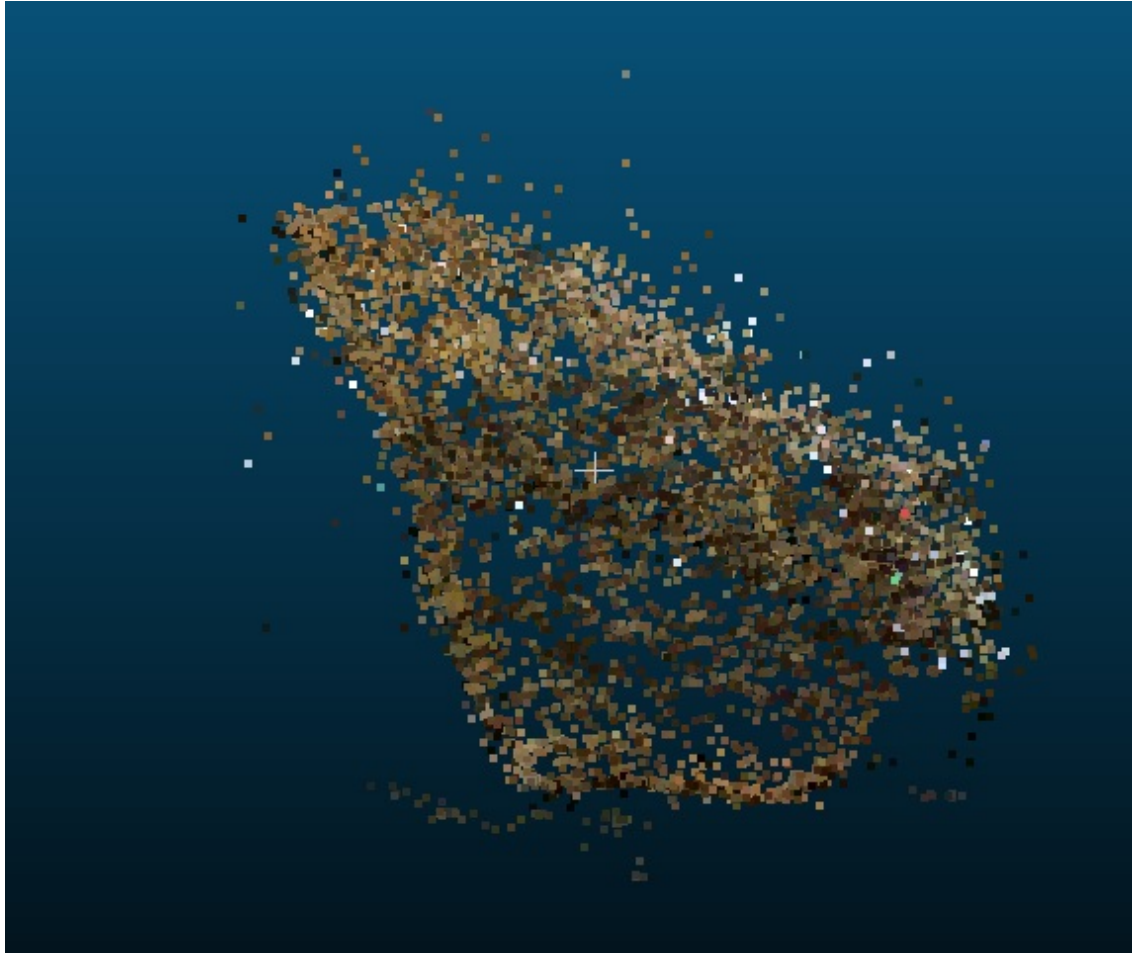


Figure 3.7: Item 1 Side 2 Segmented

### 3.1.3 Combining solutions

Most of the points we can see now are either part of the item or noise, so the next step should be to remove as much noise as possible. Using a tool to remove statistical outliers, or manually removing them is done before combining the two sides. In order to combine them, we need to specify a rotation (and scaling). In CloudCompare, we can do this by specifying 3 corresponding points, and CloudCompare will roughly aligning the point clouds. Then we fine tune the result using registraion (minimizing the mean square error). The result is a fully reconstructed 3D object. Item 2 has been created in the same way as Item 1.

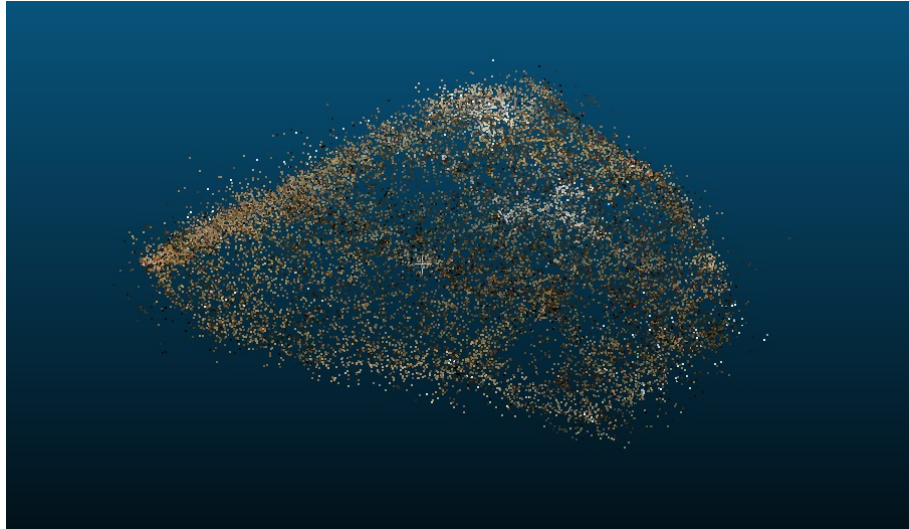


Figure 3.8: Item 1, composed from both sides

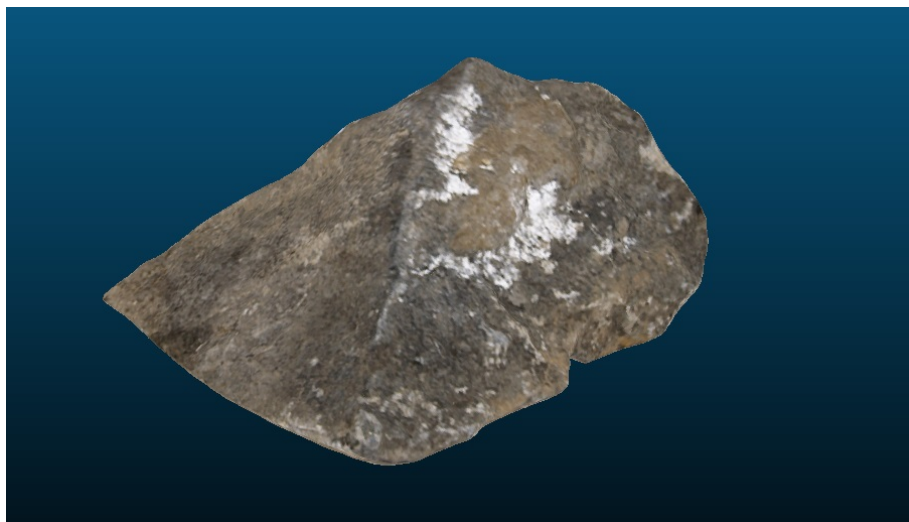


Figure 3.9: Item 1, ground truth provided by 3D-scanner



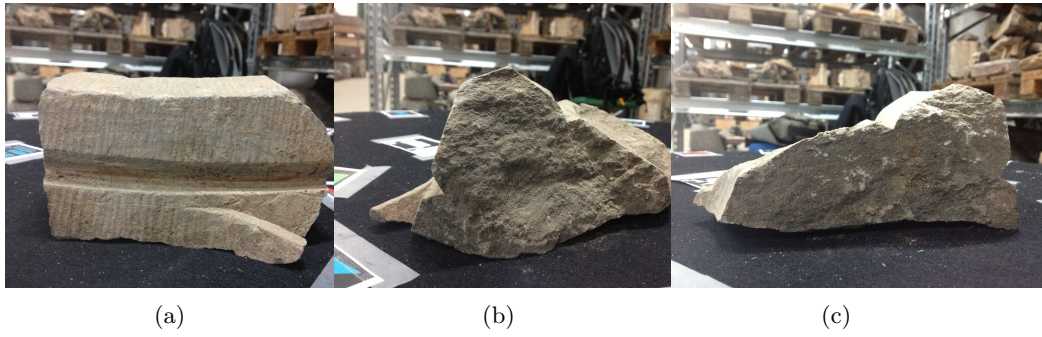


Figure 3.10: Item 2 images

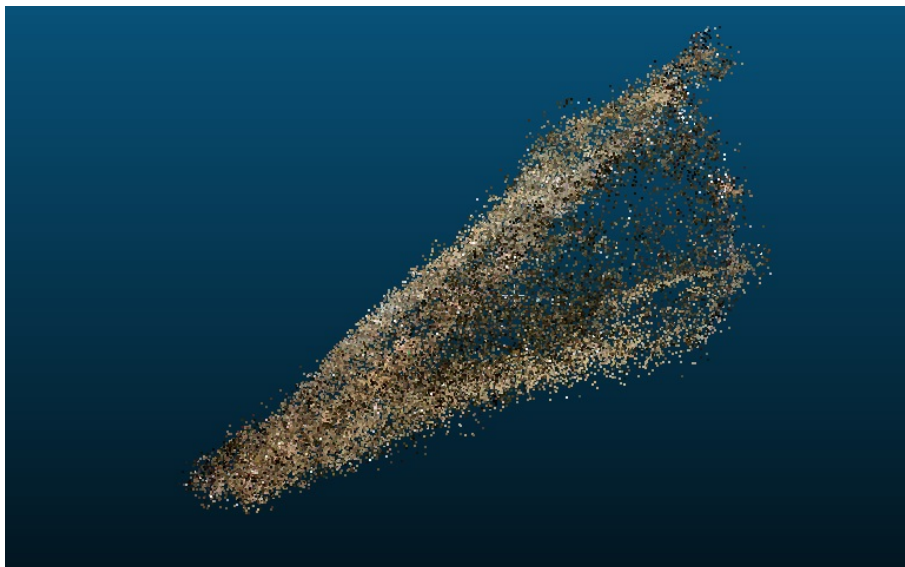


Figure 3.11: Item 2 reconstruction

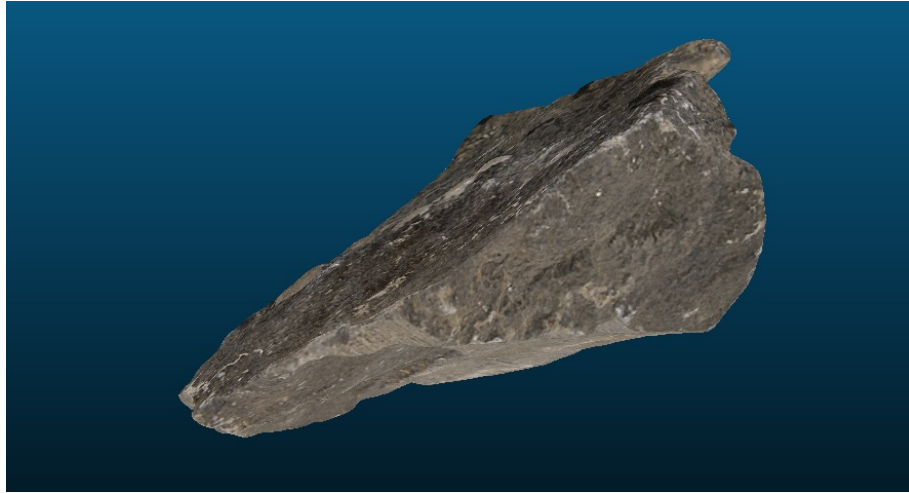


Figure 3.12: Item 2 ground truth

## 3.2 Assessing the solution

### 3.2.1 Visual assesement

The fully reconstructed objects are easier to appreciate when viewed in a 3D renderer, and we can discern a few things visually. Most importantly, the reconstructed objects are recognisable as the ones in the videos, and visibly similar to the ground truth provided by 3D-scanner. We can see that the different surfaces of the object are of the right shape and size. This should be enough for simple applications where high accuracy isn't important, such as in video games. Next we look at the objects in detail. When looking at a section of surface it is appearant



that the reconstruction will yield more of a cloud, with points distributed more densely closer to where the surface would be, while the 3D-scanner will provide a single, dense surface. This might make it harder to create a mesh or calculate normals.

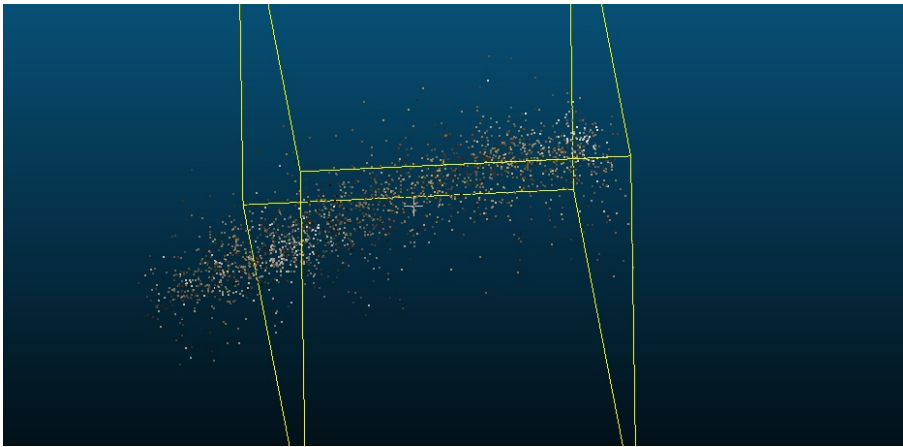


Figure 3.13: Cross section of one surface of Item 1

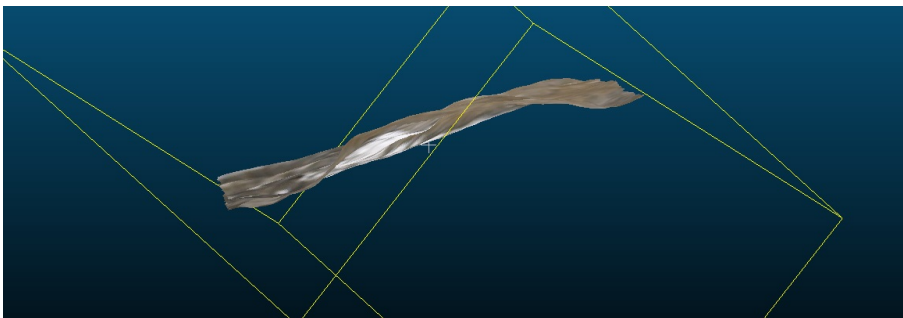


Figure 3.14: Cross section ground truth

Also, the points of the reconstruction are distributed unevenly. Because the algorithm tracks distinct points, highly texturized areas will give a dense point cloud, while polished surfaces will not provide points at all. In the image below, note that the surface (the "front" of Item 2) does not contain enough information to recreate the "groove" (seen in Figure 3.10a).

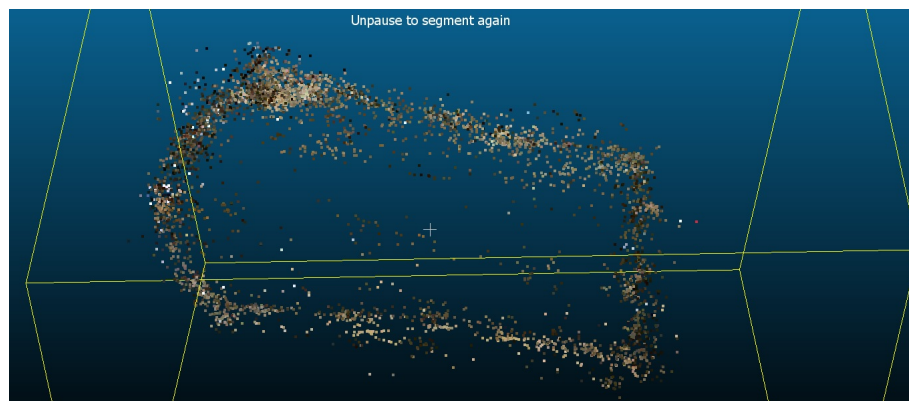


Figure 3.15: The polished side of Item 2, with few points besides the edges.



Figure 3.16: Polished side from 3D-scanner

### 3.2.2 Mathematical assesement

Finally, a direct comparison of the ground truth and the reconstructions using registration will provide a convinient and exact way of measuring error. The number of points is also a good measurement.

Item 1	
Reconstructed points	20 138
3D-scanner faces	910 976
RMS	1.24175

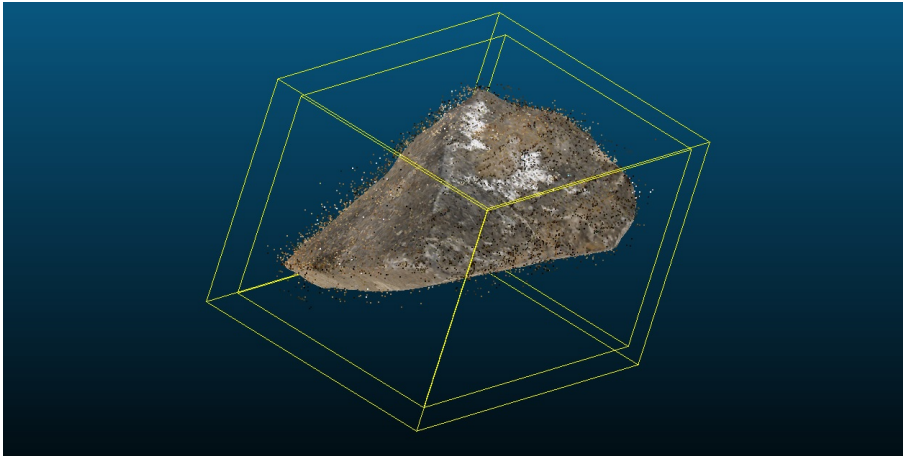


Figure 3.17: Item 1, reconstruction and ground truth combined

Item 2	
Reconstructed points	32 708
3D-scanner faces	2 503 330
RMS	5.19767

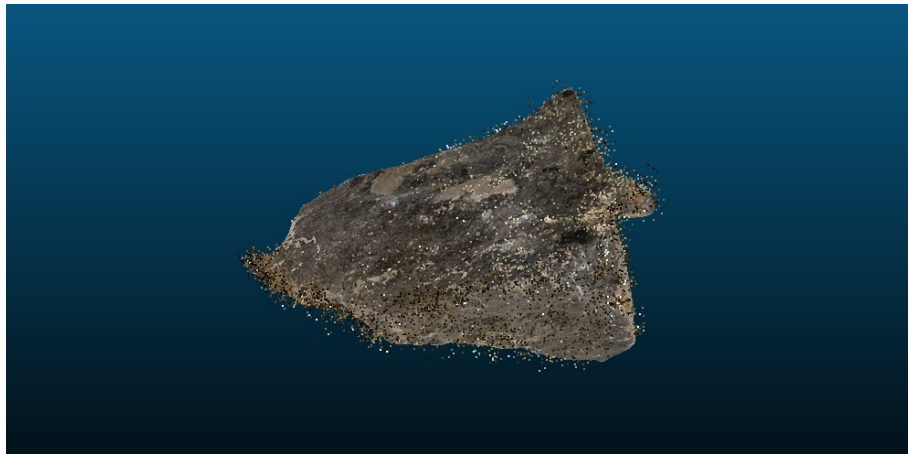


Figure 3.18: Item 2, reconstruction and ground truth combined, note the inaccurate reconstruction of the bottom left corner

## Chapter 4

# Further Work

This chapter describes work that remains to be done in order to make the SFM implementation more useful, and to remove or reduce the need for manual work.

### 4.1 Scale

The current reconstruction is without scale, but since we have information about the turntable, this is something we can set automatically. The easiest way to do this with the turntable we have is to select two points on the turntable and measure the distance between them. These points can for example be one of the corners of the colored figures on the turntable. When the reconstruction is complete, the scale can be adjusted to match the turntable.

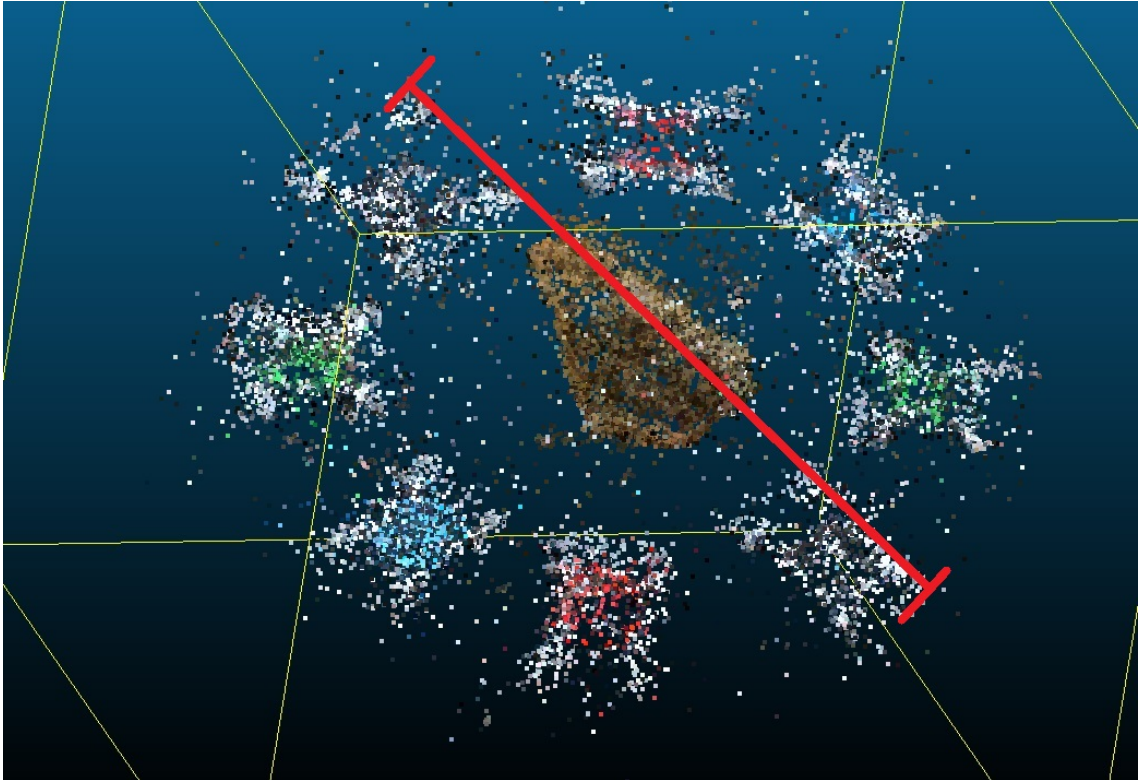


Figure 4.1: Reconstruction showing a known distance

## 4.2 Isolation

Once we have extracted the scale, we can remove the turntable from the reconstruction. This is currently done by using a statistical approach, removing points far from the centroid, which will likely lie inside the item. More advanced methods could involve identifying which points belong to the turntable, or using an adaptive statistical approach. This step should remove most of the turntable, while keeping most of the item. Noise and isolated points should also be removed (statistical outlier removal).

## 4.3 Combination

In order to automatically combine two (or more) sides of an item, we need to do two things. First, we roughly align the clouds. To do this, we could try to identify 3 easily recognisable features (corners, clusters of points) in each cloud, and using this to align them. Finally, we

can use registration to minimize the mean square error.

## 4.4 Refinement

As seen in Figure 3.13 and 3.15, the reconstruction has some flaws. As a final step, these should be refined. The target should be an even point cloud, where the surface is not "fuzzy". This will make it more like the clouds provided by 3D-scanners, and will make it easier for existing algorithms to use the point clouds. If the goal is, for example, to make texturized, polygonal models, then a uniform point cloud is easier to work with.

## Chapter 5

# Conclusion

In order to wrap up, lets review the original task, to answer whether a simple camera and software could potentially replace 3D-scanners, specifically in respect to Presious. I've shown that SFM will produce a resonable 3D item. As we can see from the results, the reconstructions only consist of about 1% of the points provided by a 3D-scanner. In addition, these points are located in a cloudy surface, instead of a sharply defined one. The reconstructions will also have weakspots and outright errors. It should be noted that all the results in this paper where created with a cheap webcam, where the items only covered a square of about 300x300 pixels. Using a better camera will increase the accuracy and number of points. In addition, there is a lot of room for improvement of the software. The weakest part of the algorithm at the moment, is reliability. Structure From Motion does consist of some guess work and estimation, and the reconstruction might fail entierly. It therefore requires some degree of human oversight.

Lastly, I want to point out what sets this approach apart from other turntable based techniques. Most importantly, unlike some other approaches [5], this one is based on a general SFM algorithm. This means we do not require strict circular motion. Results can be achieved without a turntable by simply moving the camera around an item. It also means that existing solutions and libraries can be adapted easily. In order to adapt the general approach, a few modifications have been made. The background has been filtered out in order to provide a "static" scene, and the motion of the camera is assumed to approximate a circle, or at least a closed loop. Furthermore, the resulting reconstruction has a predictable configuration, a flat, known turntable with the item at the center, inside the circle of cameras. This should make it possible to automate further.



While it is clear that SFM is not a replacement for 3D-scanners, it will produce useful results when none are available. Because of the steep price of 3D-scanners, and the wide availability of computers and cameras, turntable based Structure from Motion does have some utility.

# Bibliography

- [1] Raphal; Vandergheynst Pierre Alahi, Alexandre; Ortiz. Freak: Fast retina keypoint. 2012.
- [2] Gary Bradski and Ardiran Kaehler. *Learning OpenCV*. O'Reilly, 2008.
- [3] Brian Curless Changchang Wu, Sameer Agarwal and Steven M. Seitz. Multicore bundle adjustment. *CVPR*, 2011.
- [4] David Millan Escriva Khvedchenia Ievgen Neureen Mahmood Jason Saragih Daniel Lelis Baggio, Shervin Emani and Roy Shilkrot. *Mastering OpenCV with Practical Computer Vision Projects*. Packt, 2012.
- [5] Vincent Fremont and Ryad Chellali. Turntable-based 3d object reconstruction, 2004.
- [6] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.
- [7] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.
- [8] D.P. Robertson and Roberto Cipolla. Structure from motion. In *Practical Image Processing and Computer Vision*. Varga, 2009.
- [9] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. *IEEE International Conference on Computer Vision*, 2005.
- [10] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *European Conference on Computer Vision*, 2006.
- [11] Christopher Zach. Simple sparse bundle adjustment. *Computer Vision and Geometry Group*, 2011.