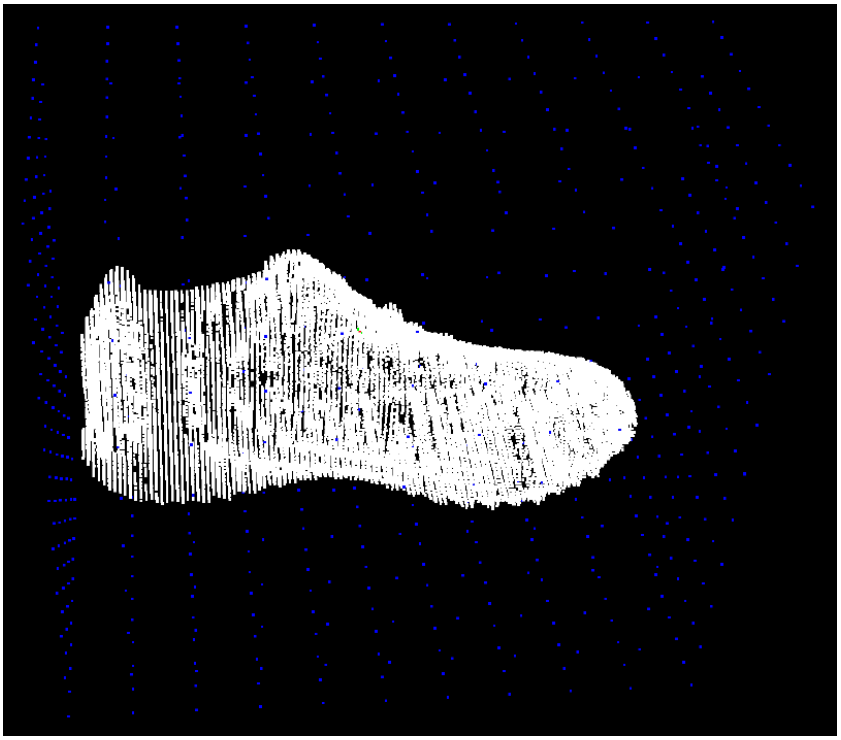
CHALMERS



Passive High Fidelity 3D-Reconstruction

Master of Science Thesis in the Program Computer Science

PER JAMOT JOHANSSON

ADAM SÄLLERGÅRD

Department of Computer Science & Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2013
Master's Thesis 2013

Passive High Fidelity 3D-Reconstruction

PER JAMOT JOHANSSON
ADAM SÄLLERGÅRD

Examiner: ULF ASSARSSON

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 10 00

Cover: Visualization of a point cloud of a shoe that was 3D-reconstructed in the thesis work.

Department of Computer Science and Engineering
Gothenburg, Sweden May 2013

Abstract

In this thesis, different methods to achieve high-fidelity passive 3D reconstruction are evaluated and discussed together with the requirements and importance of the image quality and studio settings. The different steps to go from images to a mesh are explained, and the aim is to present a solution which is as automatic as possible to minimize the need of human interaction. Therefore, also preprocessing steps such as masking is discussed.

Most focus has been given to the difficult task of pixel matching and its limitations, and how constraints, for example, ordering and smoothness, can help the matching.

Acknowledgments

We are thankful to Christoffer Nilsson that let us borrow his camera for this thesis work. To Ulf Assarsson, Erik Sintorn, and Viktor Kämpe for giving guidance and support. To Jonathan Gustafsson for helping with the implementation and research but also proofreading the thesis. To Robin Ytterlid and to Per Anders Johansson for helping with the proofreading of the report.

Contents

1 Introduction

The fields for using the internet and computers are developing rapidly. For example, selling products, making movies, and making games. One of the downsides of buying products on the internet, compared to buying them in stores, is the lack of interaction with the products. One of the ideas with this work is to change that by making a very accurate 3D model of the product. A 3D model allows for better interaction by being able to rotate and zoom in to the product.

Making a digital copy of an object by hand requires much time and skill and it is not feasible for large collections of products. Instead of digitizing the product manually, it can be scanned and reconstructed from ordinary images of the product. The scanning allows a faster and cheaper way to digitize products and can easily be made available for the customers online.

In movies or in games, skilled artists can create sculptures of, for example, buildings or characters. If these sculptures are scanned, small details such as cracks and wrinkles are included without any extra effort.

A further development of 3D reconstruction is motion capture. For an animation studio, there are several steps to produce animated material. These steps include the creation of the 3D models, animating, and rendering. The two most time consuming steps are modeling and animating. "It takes weeks to do a few seconds' worth of animation." (Takahashi, 2013) To aid the animators with the animation, motion capture can capture a performance of an actor and the animator receives a very accurate animation without any interaction.

This project focuses on facial 3D reconstruction of humanoid characters without any other tools than cameras. No human interaction will be required in the capturing stage. With this technology, highly accurate 3D models can be made with a low cost for equipment and few personnel hours.

This project should be interesting for anyone who desires 3D reconstruction, since most of the concepts are general and not bound to a single method.

1.1 Purpose

In this thesis, the aim is to scan human faces by successfully extracting the geometry with high fidelity, using no other aid than a set of photographs shot from known positions, and to study the restrictions of the 3D reconstruction. These restrictions are compared between different solutions, and some implementation details are explained and discussed.

1.2 Limitations

The project focuses on implementing 3D reconstruction, although it is important that the chosen algorithms are compatible with motion capture and retargeting for future development. There are two different types of algorithms to do 3D reconstruction: passive and active solutions. Active solutions means that aids are used to affect and modify the scene to simplify the reconstruction. Two examples of active systems are

markers (Sigal, 2012) and structured lighting (Casey & Hassebrook, 2013). Many active solutions, such as the structured lighting, can only capture from one angle at once because they affect the scene. Passive solutions, however, do not modify the surroundings and can be run in parallel.

To achieve a high detail reconstruction and motion capture for the object, it needs to be shot from all angles in parallel. This, in combination with the goal to have as little human interaction as possible, has made the active solutions less attractive (Ballan & Cortelazzo, 2008), and therefore, only passive solutions will be studied in depth. Another reason why passive solutions are studied is that they work well for both animations and to record textures without any artificial overlay. Due to the many benefits from using cameras that other solutions lack, this project focuses on 3D-reconstruction using cameras.

1.3 Problem

- What data is needed to perform 3D reconstruction and how can it be collected?
- Ordinary images are in 2D. How is the third dimension extracted from these images?
- What are the requirements for the cameras and the environment to perform 3D reconstruction?
- Is there anything that cannot be reconstructed, and why?
- Is there a single best solution for all applications? If not, what are the drawbacks and advantages of the different solutions?
- What quality can be achieved?

2 Method

The following steps have been included to complete this thesis work:

- Relevant research papers were studied to gather information and knowledge in the field.
- To get practical knowledge, selected papers were implemented in parallel with the literature study, and the implementation was improved as an iterative process when new knowledge was gathered.
- Several cameras and environments were tested to see the minimum requirements and limitations of the equipment.
- When the limitations were identified, a solution was developed to minimize the effect of the limitations, including:
    - creating a more sophisticated disparity map algorithm (see Section 3.3).
    - transforming the disparity map to a point cloud.
    - transforming the point clouds to a common world coordinate system, e.g. merging them.
    - create mesh of the point cloud.
    - multi-camera calibration.

3 Previous works

The science of 3D reconstruction has been researched for many years (Szeliski, 1992), and there is yet no perfect solution to the problem. Several approaches have been investigated including laser scanning, infrared projection scanning, ordinary RGB-image scanning, and active solution scanning. The different solutions all have their advantages and disadvantages. There are three steps that most solutions for image 3D reconstruction include, namely: calibration, feature matching, and meshing (Ansoldi, 2005).

The first step of 3D reconstruction is calibration (Section 3.1) where the relative positions of the cameras are calculated, because they are required to calculate a depth of a point in the image. With the positions of the cameras, given by the calibration, a visual hull (see Section 3.2) can be calculated. The visual hull is used to constrain the volume that the object can occupy. The constrained volume simplifies the pixel matching (described in Section 3.3).

In the pixel matching step, a depth is calculated for all pixels in every image and a depth map is produced for each image. The depth maps are transposed to 3D points in a world coordinate system, and the set of points is called a point cloud (see Section 3.4).

In Section 3.5, methods to create a mesh of the points in the point cloud are described, and in Section 4.4, a method to color the mesh by the original images is presented.

3.1 Calibration

To be able to calculate a depth from generic 2D images, the following are required: at least two images and their relative position and rotation, called extrinsic parameters (Camera Calibration and 3D Reconstruction, 2009), and the cameras internal focal lengths and principal points, called intrinsic parameters (Camera Calibration With OpenCV, 2011). The step of calculating the intrinsic and extrinsic parameters is called calibration (Duraiswami).

The first step of calibration is to find the calibration object (see Section 3.1.1 for some different calibration objects and patterns) by using an algorithm such as active contour (described in Section 3.1.2). Points that are arranged in a known pattern are masked using a blob detection algorithm, such as Laplacian of Gaussian (see Section 3.1.3). The blobs are extracted by a blob extraction algorithm such as Connected Component Labeling (see Section 3.1.4). For each camera, a Euclidean coordinate system is created and the extracted blobs are reprojected (see Section 3.4.1) to that coordinate system. The points from the different coordinate systems are then matched against each other, using for example RANSAC (see Section 3.1.5), to obtain their relative positions. To improve the accuracy of the calibration, an iterative algorithm is used to tune the output from the RANSAC algorithm (described in Section 3.1.6).

3.1.1 Calibration patterns

To be certain that good feature points can be found easily and robustly for the calibration, images of a known pattern can be used (Faugeras & Luong & Maybank, 1992). Some examples of patterns are a chessboard (Camera calibration with square chessboard, 2013), a laser pointer that are moved around (Svoboda & Martinec & Pajdla, 2005), and a ball with dots (Beeler et al., 2010).

3.1.1.1 Chessboard

The chessboard calibration technique (Camera calibration with square chessboard, 2013) is used when calibrating a system of two cameras, also called a stereo pair. A chessboard pattern is moved around and photographed from different views and distances with both cameras in the stereo pair. The points that are tracked are the intersections of the black and white squares on the chessboard. If the sizes of the chessboard and its squares are known, the 3D position of the points can be unambiguously determined for each view. The points from each view are then processed by an algorithm such as RANSAC (see Section 3.1.5) to get the relative position of the cameras.

3.1.1.2 Laser pointer

Svoboda et al. (Svoboda & Martinec & Pajdla, 2005) uses a laser pen that can be seen from every angle and move the laser pen around in a studio while shooting several images a second in parallel with all the cameras. The benefit of this algorithm is that it is easy to mask out the laser pen points and that it is easy to match them between the images, since there is only one point in each image. But no estimate can be done on the 3D coordinates of these points for each view independently, because there is no way to know the distance from the camera for a single point. This requires that many points are collected from each camera, to be able to compute the relative positions of the cameras.

3.1.1.3 Ball with dots

Beeler et al. (Beeler et al., 2010) suggests using a spherical ball with dots as can be seen in Figure 35. Using a ball as calibration object has benefits, for example, that the ball can be seen from any angle, which makes it an attractive solution for multi-camera setups. Since each dot lies on the surface of the ball, their 3D coordinates can be estimated with high certainty. The calibration method requires only one photograph of the ball. This means that the ball does not need to be moved, and no manual work is needed.

One problem for this algorithm is that the accuracy of the calibration decays with the distance from the ball’s surface (Beeler et al., 2010).

3.1.1.4 No known pattern

An alternative to have a known pattern is to find feature points in a generic image with the SIFT algorithm (Lowe, 1999). The problem is that it can be difficult if there are few distinguishable features, and there is a higher risk of mismatches of the feature points between the images. The advantages of not using images of a known pattern, is that no additional images than the images of the object, to be reconstructed, are needed, and a single camera can be used to take all the pictures.

3.1.2 Active Contour

The active contour algorithm, also called snake (Kass & Witkin & Terzopoulos, 1988), is a robust algorithm to find a shape in an image. It is for example used to find the ball in the image when using the Ball-with-dots pattern (described in Section 3.1.1.3). There are other simpler algorithms that can do this, for example a Hough transform (Forsyth & Ponce, 2002), but if the sphere is not perfectly spherical, the Hough transform will not perform as well as the active contour algorithm.

Active contour works in such way that the sought object lies where the sum of the gray scaled pixel color values in the contour of the object, minus the sum of the gray scaled pixel colors right outside of that contour, is as large as possible. The active contour equation that describes a circle is shown in equation (0).

$$\underset{\mathbf{p}_c, r_c}{\text{maximize}} \quad \frac{1}{r_c - \delta_r} \oint_{C(\mathbf{p}_c, r_c - \delta_r)} I(\mathbf{p}) d\mathbf{p} - \frac{1}{r_c + \delta_r} \oint_{C(\mathbf{p}_c, r_c + \delta_r)} I(\mathbf{p}) d\mathbf{p} \tag{0}$$

The aim is to maximize the contour integral for the two unknown values, p_c (circle center) and r_c (circle radius). The first part of the equation is the sum of all gray scaled colors on a contour lying r_c-δ_r from the center of the sphere, i.e. inside the contour of the sphere, and the second part is the sum of all gray scaled colors in a contour lying r_c+δ_r from the center of the sphere, i.e. right outside the contour of the sphere. In Figure 1, the two contours on a circle in continuous space are shown.

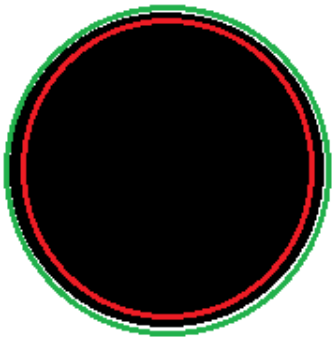


Figure 1: *A ball, masked with active contour. The green circle is the second term in equation 0, and the red circle is the first term.*

See Figure 2 of an example result when using the active contour on an image where a pre-processing step has been done before the active contour to increase the gray-scaled colors of the pixels inside the sphere.

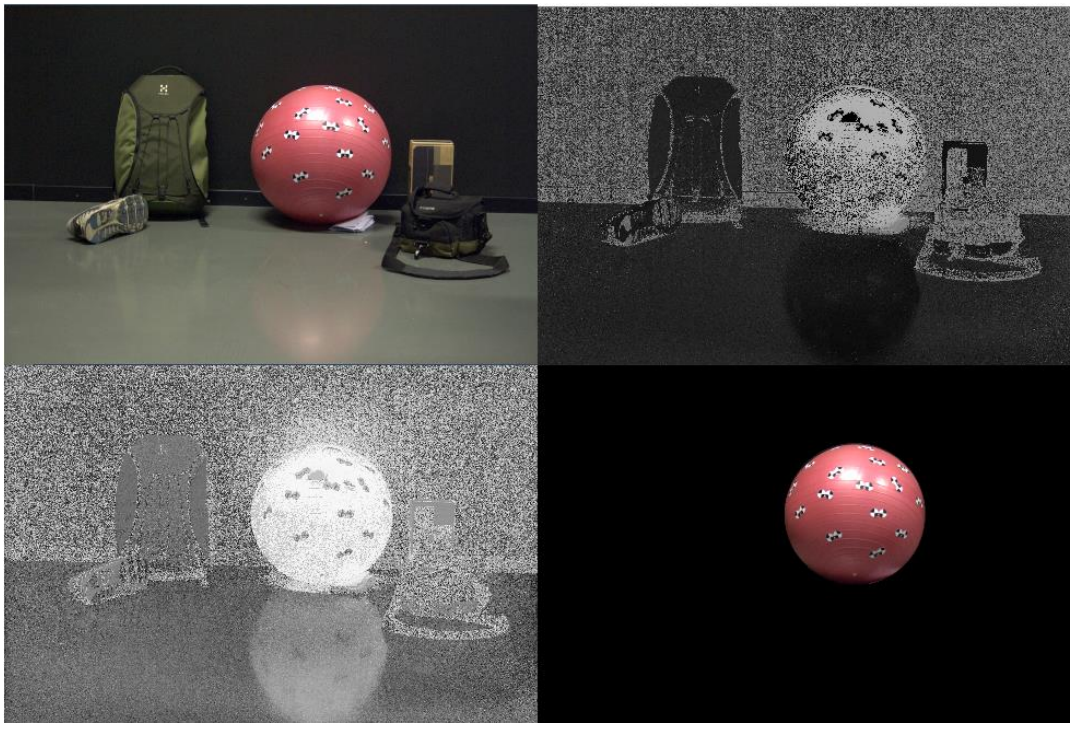


Figure 2: *The input image at the top left corner, the hue of that image at the top right, an enhanced hue image at the bottom left, and the masked out ball by using active contour algorithm at the bottom right.*

3.1.3 Blob detection

In many calibration patterns, small circles are used to find known points in the world (see Figure 35). These circles have to be found in the images somehow (Xiao, 2010), and this Section will describe a common technique to find these circles, from now on called blobs.

A commonly used technique to find blobs in an image is to first use a Gaussian filter

to smooth the image in order to reduce noise, followed by applying a Laplacian filter that outputs the second spatial derivative (i.e., identifies rapid color changes). This filter is often called Laplacian of the Gaussian (LoG) (Fisher & Perkins & Walker & Wolfart, 2003).

Before the LoG algorithm can be computed, a pre-processing step to this algorithm is to use the hue of the image and then increase the color intensity of pixels, depending on the difference between their hue and some reference hue.

For example, if yellow dots are to be found, yellow has a hue of 60, the closer hue to 60 a certain pixel has the higher intensity it will get after this pre-processing step (see Figure 3).

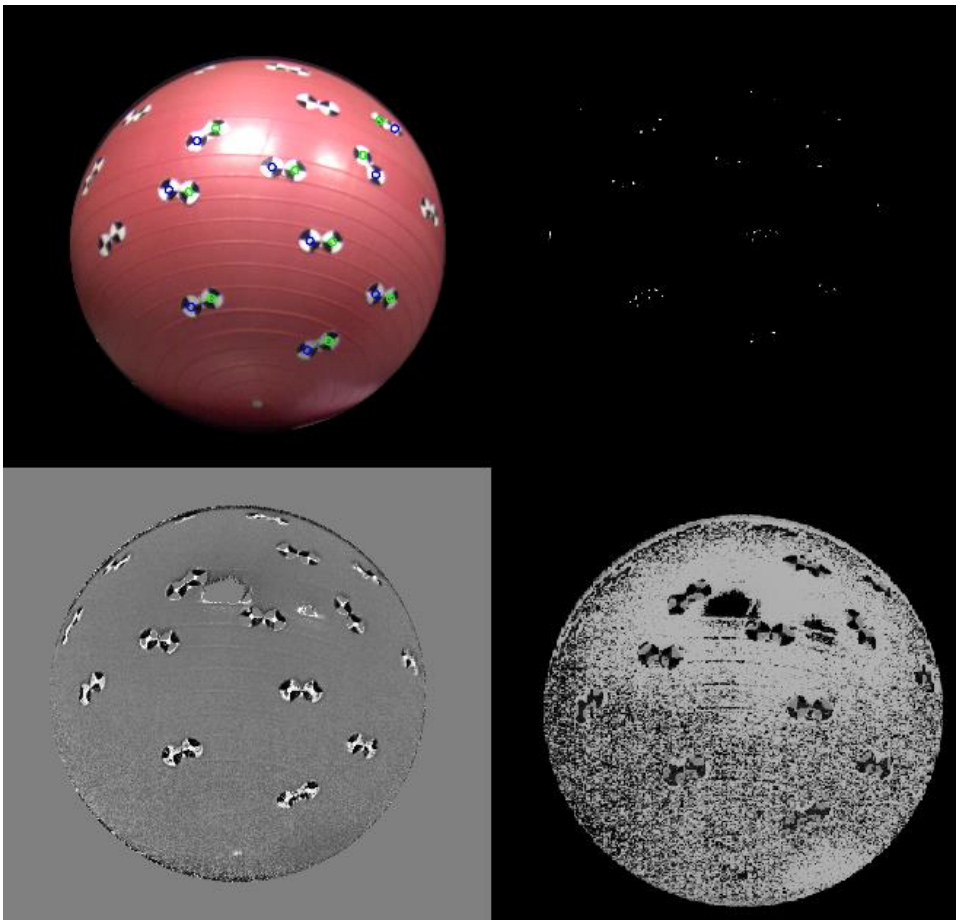


Figure 3: *The bottom right ball is the plain hue colors, and the bottom left is the pre-processing step which is the input to the LoG algorithm. The top right is the output from the LoG function and the top left has highlighted the fiducials that was identified from output from the LoG function.*

The LoG algorithm first uses a Gaussian kernel to blur the image to remove noise, of which the next kernel, the Laplacian, is very sensitive.

Now follows a small example. The purpose of the example is to demonstrate how to find a blob of a certain size by using a Laplace kernel.

Example 1:

A pixel matrix A with 6x6 pixels (see Figure 4) is convoluted with a 3x3 Laplace kernel B (see Figure 5).

0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

Figure 4: *An image with binary grey scaled colors.*

1	1	1
1	-8	1
1	1	1

Figure 5: *A 3x3 Laplace kernel*

In this example, there are two blobs in A, one made up by 3x3 pixels and one by 1 pixel. The result from the convolution can be seen in Figure 6, and the result after a threshold of -6 has been applied to Figure 6 can be seen in Figure 7.

1	1	1	0	0	0
1	-8	1	0	0	0
1	1	2	2	3	2
0	0	0	-5	-3	-5
0	0	0	-3	0	-3
0	0	0	-5	-3	-5

Figure 6: *The image from* Figure 4 *after it has been convoluted with the Laplace kernel in Figure 5*

0	0	0	0	0	0
0	-8	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Figure 7: *The same image as in Figure 6 after a threshold of -6 has been applied.*

The next example demonstrates how to, instead, find the larger 3x3 pixel blob by using a different Laplace kernel.

Example 2:

A pixel matrix A with 9x9 pixels (see Figure 8) is convoluted with a 5x5 Laplace kernel (see Figure 9). This time, there is only one blob in A (the 3x3 blob), and the 1-pixel blob is noise. That is, it has a similar color to a blob but is not an actual blob that should be found.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figure 8: *The same image as in Figure 4 but extended with zeros to make it possible to convolve all the ones with a 5x5 kernel.*

2	2	2	2	2
2	-3	-3	-3	2
2	-3	-8	-3	2
2	-3	-3	-3	2
2	2	2	2	2

Figure 9:*5x5 Laplace kernel.*

The result after the convolution is displayed in Figure 11. Figure 10 illustrates Figure 10 after a threshold of -15 has been applied.

2	2	2	2	2	0	0	0	0
2	-3	-3	-3	2	0	0	0	0
2	-3	-6	1	8	6	6	4	2
2	-3	1	0	4	-3	2	3	4
2	2	8	4	-5	-17	-7	2	6
0	0	6	-3	-17	-32	-17	-3	6
0	0	6	2	-7	-17	-7	2	6
0	0	4	3	2	-3	2	3	4
0	0	2	4	6	6	6	4	2

Figure 10: *The image in Figure 8 after a convolution with the Laplace kernel in Figure 9. Notice how the pixel on row 3, column 3, has a lower value (-6) than the pixel on row 5, column 5, which has a value -5.*

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	-17	0	0	0
0	0	0	0	-17	-32	-17	0	0
0	0	0	0	0	-17	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figure 11: *The image from Figure 10 after a threshold of -15 has been used. Notice that only five pixels were detected in the blob.*

The problem with the Laplace kernel is that it is very sensitive to noise.

In example 2, the 3x3 blob was found, but the 1-pixel sized blob also gets a rather large negative value, and with more noise, there could be a case where the noise gets a higher negative value than the actual blob. The next example will demonstrate how to minimize the effect of this problem by applying a Gaussian kernel before the Laplace kernel.

Example 3:

The original pixel matrix A (see Figure 8) is first convolved with the Gaussian kernel in Figure 12 and produces the result in Figure 13. The matrix in Figure 13 is then convolved with the Laplace kernel in Figure 9 with results shown in Figure 14.

1	2	1
2	4	2
1	2	1

Figure 12: *3x3 Gaussian kernel.*

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	2	1	0	0	0	0	0	0
0	0	2	4	2	0	0	0	0	0	0
0	0	1	2	2	3	4	3	1	0	0
0	0	0	0	3	9	12	9	3	0	0
0	0	0	0	4	12	16	12	4	0	0
0	0	0	0	3	9	12	9	3	0	0
0	0	0	0	1	3	4	3	1	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure 13: *The image from Figure 8 after it has been convoluted with the Gaussian Kernel in Figure 12. Notice how the colors of the pixels in the 3x3 window now have larger values than the 1 pixel noise.*

2	6	8	8	8	6	2	0	0	0	0
6	13	9	4	9	13	6	0	0	0	0
8	9	-16	-30	-2	31	32	22	16	8	2
8	4	-30	-41	6	52	54	48	44	27	8
8	9	-2	6	25	10	-20	1	43	44	16
6	13	31	52	10	-110	-190	-123	1	48	22
2	6	32	54	-20	-190	-290	-196	-28	46	24
0	0	22	48	1	-123	-196	-123	1	48	22
0	0	16	44	43	1	-28	1	43	44	16
0	0	8	27	44	48	46	48	44	27	8
0	0	2	8	16	22	24	22	18	8	2

Figure 14: *The result after the Convolution between the images in Figure 13 and the Laplace kernel in Figure 9.*

Now the 1-pixel blob has a value of -41, and every pixel in the 3x3 blob has a value below -110. This is considerably better than without the Gaussian blur (see Example 2).

3.1.4 Component labeling

In 3D scanning, it is often not wanted to reconstruct the background in the image, and therefore, it has to be masked away before the pixel matching. It is also essential to have a good masking of the object for the visual hull algorithm (see Section 3.2). It is often hard to find a good heuristic to tell if a pixel is part of the foreground or background. Therefore, the heuristic will assign some pixels to the foreground that should have been in the background and vice versa. To remove the noise, component labeling (Stefano & Bulgarelli, 1999) is used to identify connected components in the

image. The largest connected component in the image is then chosen.

The component labeling algorithm uses two passes. In the first pass, each pixel is tested by a given heuristic to tell if it is part of any component. If the pixel is said to be a part of some component, then the pixel gets assigned a label number. The labels are used to tell the components apart. The assigned label of the pixel depends on its neighboring pixels' labels. All different combinations of the labels of the neighboring pixels can be simplified into three cases:

Case 1: The neighboring pixels have no labels.

Case 2: Either, one of the neighboring pixels has a label, or all neighboring pixels with a label have the same label.

Case 3: Two or more of the neighboring pixels have different labels.

In case 1, a new label is assigned to the pixel, and in case 2, the pixel gets assigned the same label as its labeled neighboring pixels. In case 3, the pixel gets assigned the label with the lowest number, and the connection between the labels is recorded in all the involved labels connected lists.

When all pixels that are a part of a component have a label, the second pass is started. In this pass, each pixel is assigned the lowest label in the labels connected list, and when the second pass is finished, all components have a unique label.

There are two common ways to define if the components are connected, and they are called 4- and 8-connected. In the 4-connected version of the algorithm, only two neighboring pixels' labels need to be checked, because the other two neighboring pixels will not have been given a label yet. For example, if the image is traversed from the top left corner, then the neighboring pixels to the right and below have not yet been traversed and thus have no label. For the 8-connected version, the three neighboring pixels' labels that need to be checked are the left, top and the diagonal to the top left.

In Figure 15, there is an example of how the labeling and the connected list could look after the first pass. Each black pixel has a number printed in it to show the label it got assigned after the first pass. The heuristic to tell if a pixel is part of any component is: if the pixel is black, then it is a part of some component, and if the pixel is white, it is not.

Note that the component in a shape of a rhombus has many different labels assigned after the first pass. In the second pass, all labels will be changed to the lowest label in the connected list. In the example, all pixels in the rhombus will be labeled one in the second pass.

Figure 15: *Here is an image with two components (a rhombus and a U) and the belonging list of connected labels. The number in each pixel is the label assigned to the pixel in the first pass. The image is traversed from top left corner.*

3.1.5 RANSAC

In the calibration of the cameras, 3D points are calculated from every view. The points from the different views need to be matched together to get the positions of the cameras. A difficulty is that some points may not be visible in both views, and therefore, these points need to be identified and excluded from the matching.

RANSAC (RANdom SAmple Consensus) (Fischler & Bolles, 1981) is an algorithm that aims to set the parameters of a given model to match a data set of samples. The algorithm also tries to handle incorrect samples in the data set by assigning each point to be an inlier or an outlier. Outliers are points that are included in the data set but are invalid. Therefore, only the inliers affect how the parameters of the model are set. The model could, for example, be the equation of the line. The parameters to be set would be the derivative and the offset in the y-direction. When calibrating, a rotation in 3D is the parameter to be set, and the data set of the other view to be matched is the model.

At first, as few samples as possible, that can describe the model, are randomly chosen from the data set. The chosen samples are assigned to be inliers and all other samples in the data set get assigned to be outliers. The parameters for the model are calculated, and then for each outlier sample, an error is calculated describing how good it matches the given model. If the error is low enough, the sample is changed to be an inlier. When all samples are tested, the number of inliers in the model, combined with the sum of the errors of the inliers, becomes the total error for this iteration. In each iteration, new random samples are chosen from the data set. The iteration with the lowest error is the best match. A maximum number of iterations is often set to make the algorithm faster.

3.1.6 Calibration refinement

The intrinsic and extrinsic parameters after RANSAC (see Section 3.1.5) are rough approximations that have to be refined to be good enough for further usage. The known variables are the projected points in the images, and each projected point gives two known variables: its x and, y position.

The unknown variables are the 3D coordinates of all the points used in the calibration, but also the extrinsic and intrinsic parameters. Each 3D point has three unknowns: its

x, y, and z coordinate.

The unknown extrinsic parameters are four rotation parameters, and three translations. The unknown intrinsic are the focal length, the principal point, and possibly five distortion parameters that can describe the tangential and radial distortion.

This adds up to 16 unknown variables for every camera and three unknown variables per 3D point.

Consider a system with five cameras and 15 3D points, i.e. 125 unknown variables. When a 3D point is visible in an image, it is called a projection. A 3D point might be visible in more than one image and therefore will produce more than one projection.

Since every projection adds two known variables, and for an equation system to be solvable, there has to be at least the same amount of known and unknown variables. Therefore, each point has to create at least two projections to be useful. This means that, for the example with 125 unknown variables, 63 projections have to be found in the five images. It is, therefore, important with a high overlap between the images.

The minimization problem is to find the 3D coordinates, extrinsic, and intrinsic variables, so that the projections of those 3D coordinates differentiate as little as possible to the known projections in the images. A common minimization algorithm often used when calibrating is called Sparse Bundle Adjustment (SBA) (Triggs & McLauchlan & Hartley & Fitzgibbon, 1999).

3.2 Visual Hull

Visual hull is a fast and robust technique for 3D reconstruction that uses the contours of objects to find the surface (Vogiatzis, 2005). The technique requires calibrated cameras. Additionally, the technique only works for objects that have silhouettes against the masked background. A disadvantage is that it is impossible to reconstruct any concavities that cannot be seen in the silhouette of any camera. An example of such a concavity can be seen between the feet in the right image of Figure 16.

Figure 16: *(Hernandez & Schmitt, 2003) The left image is the visual hull, and the right image is the complete reconstruction of the statue. Some artifacts are visible in the left image due to lack of silhouettes in those regions.*

Hernandez et al. (Hernandez & Schmitt, 2003) suggests that a visual hull can be used as an input estimation to the pixel matching step and then, for each pixel, a ray has to be created and intersected against the visual hull.

The benefit of using this algorithm is also that no perspective distortion affects the result. Perspective distortion appears due to the perspective projection (Grimm & Goldman, 2010) that happens when the image is projected down to a point, similar to how our eyes function. Another projection type, known in Computer Graphics, is called Orthogonal projections (Inder, 2003), where the image is projected down to an area (see Figure 17).

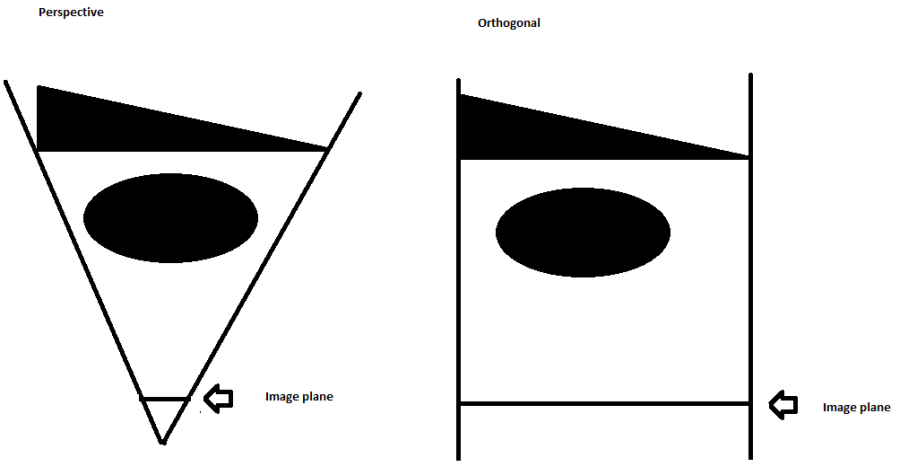


Figure 17: *The Figure above displays the difference between orthogonal and perspective projections. The orthogonal projection projects to a plane while perspective projection projects to a point.*

In orthogonal projections, the objects keep their size, no matter how close or far they are from the lens. That is what is wanted in the feature-matching step (see Section 3.3).

In perspective projections, the object's size in the image depends of how far away it is from the viewpoint, and if these sizes differ notably between two images, there is a risk that their match will not be found.

3.3 Feature Matching

To reconstruct an object, the depth has to be collected, and the main solution that is evaluated in this thesis is called pixel matching. The technique states that if two cameras' intrinsic and extrinsic parameters are known and if the pixels, describing the same point of the object, are found in both images, then the 3D coordinate of that point is also known. Since the calibration step (described in Section 3.1) returns both the intrinsic and extrinsic parameters, the task is now to match pixels between images.

One of the most common technique is called block matching (Chen, 2012), which means taking a block around the pixel whose depth is to be calculated and match that block against blocks of possible matches in the other images. The block matching returns an estimate of how well the blocks match, which is the probability that the two blocks represent the same surface of the object.

The most common template matching algorithm used, to match the two blocks, is called Normalized Cross Correlation (NCC) (Hii & Hann & Chase & Van Houten, 2006), and the mathematical function can be seen in Equation 1, where A and B are two blocks of some size.

$$corr = \frac{\sum_i \sum_j (A_{ij} - \bar{A})(B_{ij} - \bar{B})}{\sqrt{(\sum_i \sum_j (A_{ij} - \bar{A})^2)(\sum_i \sum_j (B_{ij} - \bar{B})^2)}} \quad (1)$$

From the matching of a pixel, a depth or a disparity can be computed. The disparity of a pixel defines how many pixels the corresponding point has moved between two images. For example, if a pixel is on column 10 in one image and its match is on column 15 in another image, then that pixel has a disparity of 5.

3.3.1 Challenges

Even if the images are perfect for reconstruction, e.g., no noise and exactly the same color balance, matching pixels between the images would be challenging due to some physical properties, explained in the Sections 3.3.1.1 to 3.3.1.4.

3.3.1.1 Texture Repetition

Texture repetition is the problem that occurs when there are repetitive patterns in the image, and there is therefore a large risk for a mismatch.

Figure 18: *Picture displaying texture repetition. The lines and squares are repeated on the surface of the pillow.*

In Figure 18, an image with repetitions can be seen. A numerical example to illustrate the problem in depth is shown in example 4.

Example 4:

Assume two images I and J, represented by two matrices with gray scaled pixel colors (see Figure 19). The block size is 3x3 pixels and the block $A_{i,}$ in image I, is highlighted in Figure 20. A_i is to be matched in the image J, where two blocks B_j and C_j are matches (highlighted in Figure 21). B_j and C_j both are perfect matches, and there is no way to tell which of them that is the correct match. This phenomenon can happen when there are texture repetitions.

1 2 3 4 5 6 7	2 3 4 5 6 1 1 1
0 1 1 2 3 4 0	1 1 2 3 4 1 2 3
0 1 2 3 2 4 0	1 2 3 2 4 2 3 2
0 1 1 2 3 4 0	1 1 2 3 4 1 2 3
0 1 1 2 3 3 0	1 1 2 3 3 1 2 3

Figure 19: *Two images with 5x7 pixels.*

1 2 3 4 5 6 7
0 1 **1 2 3** 4 0
0 1 **2 3 2** 4 0
0 1 **1 2 3** 4 0
0 1 1 2 3 3 0

Figure 20: *The 3x3 block highlighted are to be matched against the blocks in the right matrix in Figure 19.*

2 3 4 5 6 1 1 1
1 **1 2 3** 4 **1 2 3**
1 **2 3 2** 4 **2 3 2**
1 **1 2 3** 4 **1 2 3**
1 1 2 3 3 1 2 3

Figure 21: *The two blocks that matched perfectly against the highlighted block in Figure 20 are highlighted in this image.*

A solution to texture repetition (explained in example 4) is to increase the size of the blocks used (see example 5).

Example 5:

Here are the same images I and J as in example 4, but the block size is 5x5 pixels. The block A_i (highlighted in Figure 22) is matched in image J. One match is found in image J, called B_j (highlighted in Figure 23).

1 **2 3 4 5 6** 7
0 **1 1 2 3 4** 0
0 **1 2 3 2 4** 0
0 **1 1 2 3 4** 0
0 **1 1 2 3 3** 0

Figure 22: *A 5x5 block surrounding the same pixel as the block in Figure 20 with the exception of the block size.*

1 1 2 3 4 1 2 3
1 2 3 2 4 2 3 2
1 1 2 3 4 1 2 3
1 1 2 3 3 1 2 3

Figure 23 *The correct match to the block highlighted in Figure 22 is highlighted in this image. Notice how only one block matches well this time, instead of two as in Figure 21.*

The essence of example 4 and 5 is that larger block sizes can neutralize the problem of texture repetition, but there are other disadvantages of having larger block sizes, such as less accuracy and longer computation times. Ideally, it would be good to first have a large block to get a rough estimate of where the pixel match, and then decrease the block size and limit the possible matches to around the previous estimation.

However, it does not solve the computation time problem, but rather, it is increased. Beeler et al. (Beeler et al., 2010) suggests a hierarchical solution, where the images are down sampled to make a, so called, image pyramid while keeping the size of the blocks used for matching very low (3x3). First, the pixels are matched at the lowest resolution of the pyramid. This approximates using larger blocks at the original image, but instead of increasing the computation time, as with larger blocks, it might decrease. Second, the matches from the lowest resolution image are used as input to the higher resolution layer to limit the possible matches; hence, the computation time can decrease. In Figure 24, an example of such a depth-map pyramid can be seen.

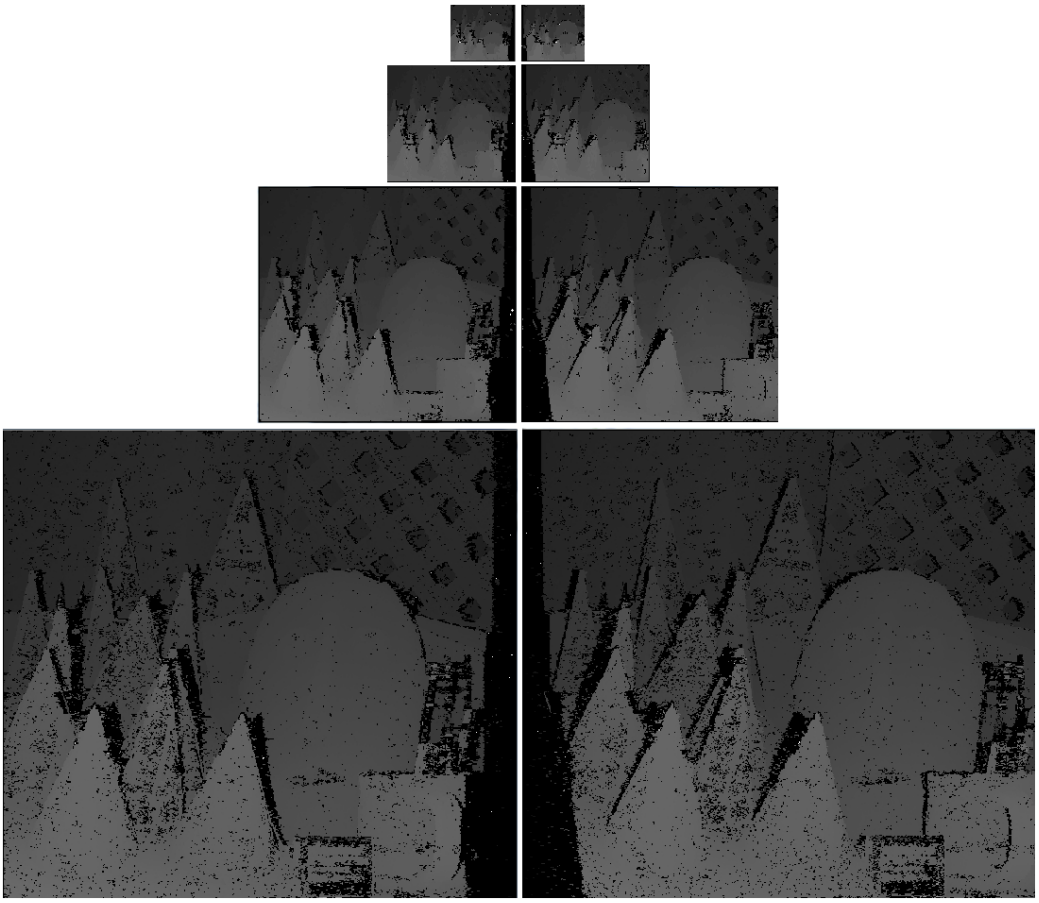


Figure 24: *A depth-map pyramid created using a data set from Middlebury (Scharstein, 2003). The pyramid has four levels and the detail is increased as the resolution is increased*

Figure 25: *The data set from Middlebury (Scharstein, 2003) that was used to create the disparity map pyramid above. The images might look similar, but they are shot from slightly different angles and positions.*

3.3.1.2 Lack of texture

The problem when there simply is not enough texture, so that no candidate depth can be found, is called lack of texture (see Figure 26). The white surface, marked with the red rectangle, in Figure 26, does not have any unique texture and therefore cannot be matched. The lack of texture problem implies a significant restriction to what objects that can be reconstructed with passive pixel matching.

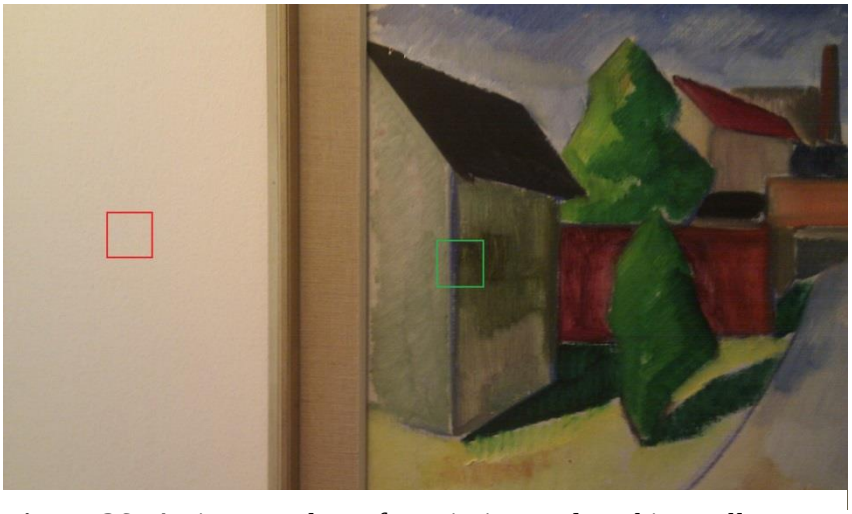


Figure 26: *An image taken of a painting and a white wall to demonstrate the problem with lack of texture. The red square lacks texture while the green one does not.*

A possible solution is to use projectors to project patterns on the object, such as the Kinect (Kinect for Windows). The Kinect uses an IR projector to project an invisible dot pattern and an IR camera to photograph the pattern to calculate the depth, depending on the distance between dots. A Kinect, however, is limited to the IR-projector resolution, and several Kinects can only operate sequentially.

Another solution is to use light to create different colors on the surface and therefore more texture (Hernandez & Schmitt, 2007), but this thesis focuses more towards other problems with feature matching.

3.3.1.3 Occlusion

Occlusion occurs when a point on the object-to-be-reconstructed is only visible in one of the two images that should be used for matching. An example when occlusion appears can be seen in Figure 27, where the red dot, in the left image, is behind a pillar in the right image due to the rotation between the images.

Figure 27: *The red dot in the left image is not visible in the right image due the rotation between the views of the temple. This is an example of occlusion.(Scharstein)*

When a point is to be matched to another image where it is not visible, there is no reason that it should find the right depth. Therefore, there is a very high risk of an outlier. There is no way to calculate the right depth of an occluded point. Hence, it is wanted to identify these points and calculate their depth from another image pair instead.

A solution that is discussed by both Beeler et al. (Beeler et al., 2010) and Campbell et al. (Campbell & Vogiatzis & Hernandez & Cipolla , 2008) is to remove the occluded points by using a smoothness constraint. The constraint forces a pixel to have a depth close to its neighboring pixels depth (see Section 3.3.2.2).

3.3.1.4 Distorted images due to slanted surfaces

The problem caused by slanted surfaces occurs when two cameras-to-be-matched views a surface from different angles, making the surface area differentiate a lot between the images (see Figure 28).

Figure 28: *Two photos taken at a bulletin board on a wall. The photos are taken from different angles to display how the bulletin board area differentiates.*

This problem is mainly solved by using a smoothness constraint (Wang & Zheng, 2008). But it can also partly be solved by obtaining the normal at the surface of a pixel. The block sizes can then be adjusted according to the normal to fit the other block.

3.3.2 Constraints

To identify miss matched pixels (due to the problems described in Section 3.3.1.1 to Section 3.3.1.4) Beeler et al. (Beeler et al., 2010) suggest applying three constraints, called ordering, smoothness, and uniqueness, when reconstructing a face.

3.3.2.1 Ordering

The ordering constraint takes advantage of the fact that a face consist of a connected surface, which means that the phenomenon that is shown in Figure 29 cannot happen.

The two images in Figure 29 are taken from different positions. Notice that in the top image, the part surrounded by a red rectangle is to the left of the part that is surrounded by a green rectangle. In the bottom image in Figure 29, the part surrounded by the green rectangle is to the right of the part surrounded by the red

rectangle.

Figure 29: *The above Figure displays a case where the ordering constraint does not hold. In the top image a part is highlighted with a green rectangle and another part is highlighted with a red rectangle. In the bottom image the two highlighted regions have switched places.*

Beeler et al. (Beeler et al., 2010) formulates the constraint given a pixel p as: "*computed disparity at p does not exceed the disparity of its right-neighbor pixel by more than one pixel*"

3.3.2.2 Smoothness

The smoothness constraint tries to enforce a high depth coherency for a pixel and its neighbors. Beeler et al. (Beeler et al., 2010) implements this constraint by: "*enforcing that more than half of all neighbors in a 3x3 neighborhood differ by a disparity less than one pixel*"

3.3.2.3 Uniqueness

Beeler et al. (Beeler et al., 2010) also have a constraint that says that given a pair of images, called i and j, and a pixel p in i that matches against a pixel q in j, the uniqueness constraint hold if and only if q matches back to pixel p. An extension to this constraint is that a uniqueness threshold of up to one disparity still passes the uniqueness constraint (see Figure 30 for an example).

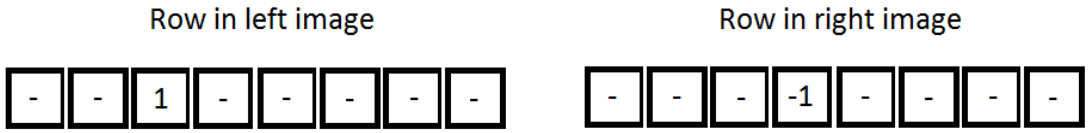


Figure 30: *Two pixel rows from different images and their disparity.*

If the above matrices are pixel rows in two rectified images and the numbers are the disparity for the pixels, then the third pixel in the left row has a disparity of 1, which corresponds to a match against the fourth pixel in the right row.

The fourth pixel in the right row, in return, has a disparity of -1, which says that it corresponds back to the third pixel in the left image and the uniqueness constraint is fulfilled without any problems.

3.4 Point clouds

A point cloud is a set of 3D points, and in 3D reconstruction, the points can be seen as samples of the surface of the object. Every pixel in the images is reprojected to a point in the world, with help of the camera parameters, and together, all points become a point cloud. The point cloud can be used to either describe the object by itself, or it could be processed further into a mesh.

3.4.1 Reprojection

Reprojection is the opposite of projection, which means that reprojection is to go from 2D to 3D. To be able to do this, the camera's intrinsic parameters has to be known. If the focal length and principal point is known for the camera, a ray can be created for each pixel (see Figure 31 for a one dimensional case).

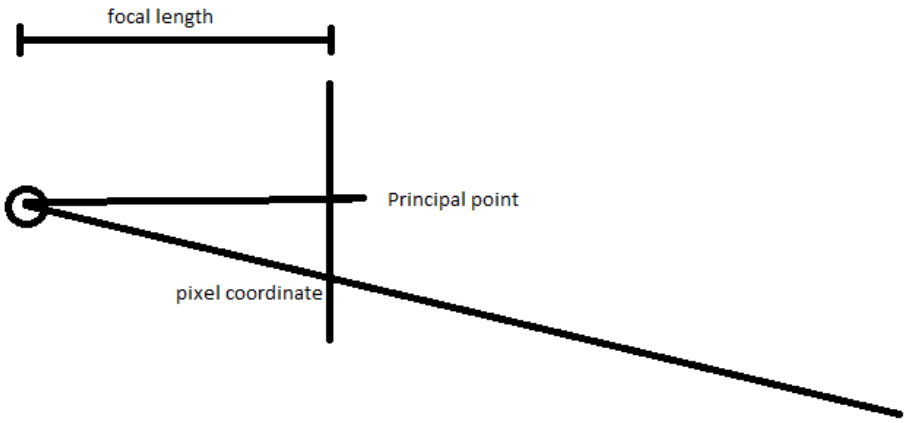


Figure 31: *The above image shows a simplified model of how a camera is constructed. The relevant camera parameters are the focal length and principal point. The focal length is the length between the focal point and the image plane. The principal point is the point that is intersected by a line going from the focal point and perpendicular through the image plane, usually in the center of the image.*

The output from the feature matching step is on what depth on the ray that the pixel lies in 3D, and the reprojection can be done.

In equation (3), $p = (p_x,p_y)$ is a pixel coordinate, $c = (c_x,c_y)$ is the principal point, and f is the focal length. (d_x,d_y,d_z) is the direction of a ray that starts in the focal point of the camera and intersects the pixel p on the image plane.

$$(x, y, z) = (px - cx, py - cy, f)$$
$$(dx, dy, dz) = \frac{(x, y, z)}{|(x, y, z)|} \qquad (3)$$

If it is known that pixel p is positioned t units from the focal point, then the 3D coordinate of p is given by equation (4):

$$(X, Y, Z) = (dx, dy, dz) \cdot t \qquad (4)$$

3.4.2 Outlier elimination

When more than one image pair exists and the image pairs overlap, there will be pixels describing the same surface. Each pixel produces a point in the point cloud, and therefore, there can be multiple points in the point cloud that are describing the same point of the surface. These points can have small deviations in the depth and have to be removed to avoid producing a high-frequent noise in the mesh.

Beeler et al. (Beeler et al., 2010) suggests that the points should be projected down to each image. Consider the case with two points A and B that lie on the same ray for a pixel in an image. If no point lie between A and B on that ray and if both the points' normals are facing towards the camera, then no surface can exist that can represent

these points, and one of them has to be removed (see Figure 32).

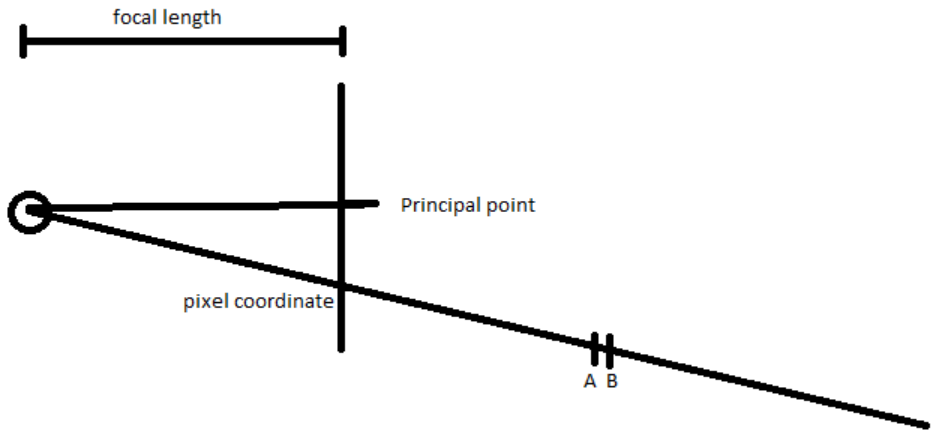


Figure 32: *The same camera as in Figure 31, but now, two points, A and B, in the world projects down to the same pixel in the image. This might indicate that they are duplicates and that one of them should be removed.*

A problem with this approach can be seen when using sub-pixel accuracy. If two pixels p and q in an image, I, is reconstructed with sub-pixel accuracy, they might both project down to the same pixel due to rounding from floating point to integer. The problem is that they can still be valid, since they actually represent different points in the world.

Example:

Two neighboring pixels in an image, I, with indices 0 and 1, and with disparities 10.2 and 9.4, are matched against pixels with indices 10.2 and 10.4 in image J. Both of these pixels will then project down to the pixel with index 10 in image J, and at least one of them would be removed.

3.5 Meshing

Even if the point cloud is very dense, it is often preferred to have a surface (Frey, 2004). There are different ways to represent the surface, but the most common one is by triangles. There are several ways to go from a point cloud to a triangulated surface (see Section 3.5.1 Section Delaunay triangulation, 3.5.2 Poisson surface reconstruction, and 3.5.3 Marching Cubes). These algorithms are commonly used for meshing, with respect to object reconstruction, from scanned point clouds (Rosenthal & Linsen, 2006).

3.5.1 Delaunay triangulation

The requirement for Delaunay triangulation (Maur, 2002) in two dimensions is that no vertex can be inside the circumcircle of a triangle. That is, a circle whose boarder goes

through the vertices of the triangle can not include any other vertices (see Figure 33).

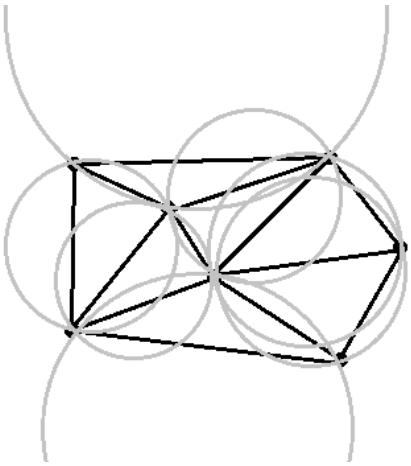


Figure 33: *A triangulation in 2D where the circumcircles are drawn and the constraints for Delaunay triangulation are fulfilled.*

In three dimensions instead of a circumcircle, triangle, and three vertices, a circumsphere, tetrahedron, and 4 vertices are used. The condition is that no vertex is inside any circumsphere of the tetrahedrons. To fulfill this requirement, there are some classes of implementations. There are incremental construction (Joe, 1991), incremental insertion (Watson, 1981), higher dimension embedding (Goodman & O´Rouke, 1997), and divide and conquer (Scopigno, 1992).

3.5.2 Poisson surface reconstruction

In Poisson surface reconstruction (Kazhdan & Bolitho & Hoppe, 2006), an indicator function is defined to make everything inside the model 1 and everything outside of the model 0. The gradient to the function is then zero everywhere but at the surface. To be able to calculate this indicator function, the normals for the points are needed. The normals can be estimated by approximating a plane with the closest neighbors. The oriented points can be seen as samples of the gradient of the indicator function. The problem of calculating the indicator function can be solved by approximating the function that best represents the sampled oriented points that are given. The set of points are said to be Poisson distributed.

By approximating a function, this method is very resilient of noise and different densities of the samples, because the whole set of points are taken into consideration at once. To triangulate the surface, the Marching Cubes (see Section 3.5.3) algorithm can be used. An advantage of Poisson surface reconstruction is that the depth of the octree of the marching cubes algorithm can be chosen and by that be able to decide the ratio of speed and detail of the triangulated model.

3.5.3 Marching cubes

Marching cubes can be used to triangulate a function that describes a surface of a model (Lorensen & Cline, 1987). This function returns a value for each point in space and is positive if it is inside the model and negative if it is outside the model. The surface is where the function returns zero.

The volume is divided into a voxel grid, and for each voxel, the function is calculated for the positions of the corners. If two adjacent corners have different signs on the returned value, the surface is somewhere at the edge between these corners. A vertex is placed at the edge where the interpolation between its corners' values is zero.

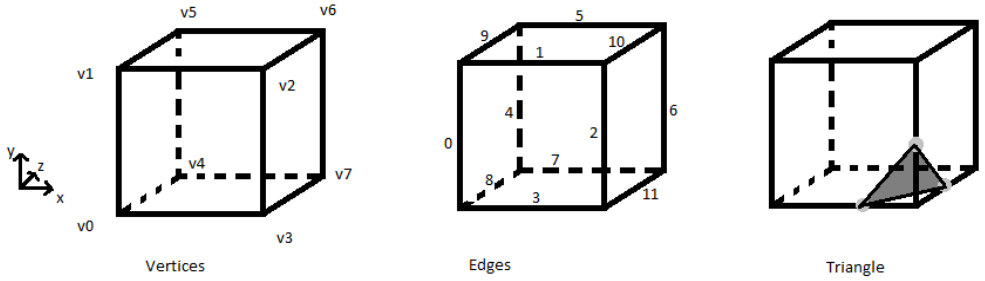


Figure 34: *An example of a triangulation of a cube in marching cubes. Here, vertex three has a different sign than the adjacent vertices. The vertices of the triangle are placed along the edges between vertex three and vertices zero, two, and seven, which can be seen in the cube to the right.*

4 Implementation

This chapter describes the choice of camera and algorithms and how part of the algorithms are implemented. Section 4.1 describes the importance of the studio settings to get high quality images and getting a good base for the pixel matching. In Section 4.2, the implementation of the calibration step is explained, and the pixel-matching step is described in Section 4.3. In Section 4.2, the implementation of the calibration step is laid out, and in Section 4.4, the implementation of the texturing of the mesh is described.

4.1 Image quality

The quality of the images is arguably the most important part of 3D reconstruction, because, if the images from the different positions differ too much, it is close to impossible to get a good match between the images and a good reconstruction.

The settings and specifications of the cameras and the environment affect the quality of the image and will be described in the subsections of 4.1.

4.1.1 Resolution

With higher resolution, there will be more pixels that describe the same area on the object-to-be-reconstructed, and finer details can be found. A disadvantage of high-resolution cameras is that, generally, the size of each pixel sensor decreases when the resolution increases and the risk of noise in the image is higher.

4.1.2 Sensor Sensitivity

The sensitivity of the sensors can be set by the ISO-value (International Organization for Standardization), and a higher ISO-value is required if the environment is dark or a fast shutter time is used. The sensors all have small differences in the sensitivity from the production, and this difference is amplified if the ISO value is set to a high value. Therefore, it is important to have a well-lit studio.

4.1.3 F-stop

The opening of the aperture is called the f-stop and regulates the amount of light that passes into the sensors. If the f-stop is large, less light is let in and a larger depth is in focus. A large depth of field is wanted because the whole object needs to be in focus. There could be cases where areas that are in focus are matched against areas that are not in focus, and a precise match is hard to find. It is also important to keep the focus from the calibration throughout the photo suite, because the angle of view is slightly changed when focus is changed.

4.1.4 Shutter speed

The time the sensors are exposed to the light is regulated by the shutter, and the time it is open is called shutter speed. The longer it is open, the more light will hit the sensors. Objects in the scene will be blurred due to movement when the shutter is open. Therefore, a faster shutter speed will make sharper images of moving objects. A balance between the shutter speed, ISO, and F-stop is required to get the right amount of light in the image.

4.1.5 Focal length

The focal length is the distance between the principal point and the sensors in the camera (see Section 3.4.1). This distance, together with the total size of the sensors, decides the angle of view and the magnification value of the lens. As described in Section 3.3.1.2, a problem with high angle of views and low magnifications is that there will be much perspective distortion. Another problem is that each pixel will represent a large area of the object and, therefore, details can be lost.

4.1.6 Lighting

To be able to set the camera settings as described in Section 4.1.1 to 4.1.4, a very well lit studio is required, and if the images are going to be used to collect textures to the models, it is important to have an even and color neutral light. To get a strong even neutral colored light, the studio needs to be lit from many directions and preferably no direct light, because they create large bright spots on shiny objects. A white colored LED will not create any specular spots or affect the natural colors of the objects. If many lights are positioned evenly around the object, it will be evenly lit.

4.1.7 Result and Discussion

With the settings described in Section 4.1.1 to 4.1.6 in consideration, a system camera with as high resolution and sensor size as the budget allows is recommended. A lens with at least a focal length of 50 mm is preferred, and when the focus is set, it should be constant until a new calibration. The more indirect light from evenly positioned sources, the better.

4.2 Calibration

For the calibration step in this thesis work, the most important aspect is that it should be without any user interaction, but also to work for several cameras and to be robust.

The chessboard pattern algorithm (presented in Section 3.1.1.1) is robust because the pattern used is known. Therefore, the 3D coordinate system, including the chessboard points, is known. The problem with the technique is that it only works for stereo pairs, since it use a flat chessboard pattern that cannot be seen from every angle.

To calibrate without a known pattern (see Section 3.1.1.4) is more and more common for 3D reconstruction systems for end users, because it does not require any pre-

processing.

To find points, some key features are masked out in each image, and these features are being matched together and will act as the calibration points. The problem with this technique is that matching uncalibrated images is difficult. Consider that pixel p in image i is used as a point in the calibration, and its match is to be found in image j. Then, potentially every pixel in image j is a match to pixel p. In calibrated images, a pixel can only match against a small subset of pixels, which is a simpler problem. This means that using a known calibration pattern is more robust than using uncalibrated images. As key feature finding algorithms get better, the differences might be reduced.

The laser pointer algorithm (see Section 3.1.1.2) is robust because of the ease to find the points in each image. However, it requires too much human interaction to be an appealing option for the automatic system wanted in this thesis.

The calibration algorithm that is chosen and implemented uses a ball (introduced in Section 3.1.1.3). This algorithm can calibrate several cameras and only requires one image, taken from each camera, to do so. The algorithm uses a ball with markers (see Figure 35). The markers are from now on called fiducials. In each fiducial, there is a chessboard pattern, and in the corners of the pattern, there are yellow dots.

Figure 35: *Ball with markers used for calibration. The fiducials have two yellow dots in them.*

The first step in the calibration method is to 3D reconstruct the ball. Reconstructing a sphere or a ball is a much simpler problem than doing a general reconstruction of an object or environment. The active contour algorithm (described in Section 3.1.2) is used to find the ball in the image, and because the real ball radius is known, the only

thing that is left, is to calculate the 3D position of the ball in relation to the camera in the view space.

The ball center can be calculated using some basic algebra if the real ball radius, the ball radius in the image, and the focal length is known. It is enough with an educated guess for the focal length. A new coordinate system is then created for each camera. In the coordinate systems, the ball is centered in origin, so that the only differences between the new coordinate systems are the orientations.

The next step is to reconstruct the two yellow points in each fiducial. They are identified and their image coordinates extracted by the blob detection and component labeling algorithms (defined in Sections 3.1.3 and 3.1.4). They can be reconstructed by shooting a ray from the image coordinate of the point through the focal point and intersect the sphere. The position where the ray intersects the sphere is the position of the fiducial point.

Then, a triangle is created for each fiducial, with two vertices in the triangle lying in the two yellow points in the fiducial and the third in the origin. RANSAC (which is described in Section 3.1.5) is then used to calculate the orientation between each camera's coordinate system to get the orientation of the cameras.

At this step, every parameter is known; the focal length, the principal point (which lies in the middle of the image), the position, and rotation of each camera. However, it is not the true values, but good estimations, because of the approximated focal length, principal point, and that the ball used is not a perfect sphere.

The result is then refined using the minimization algorithm described in Section 3.1.6.

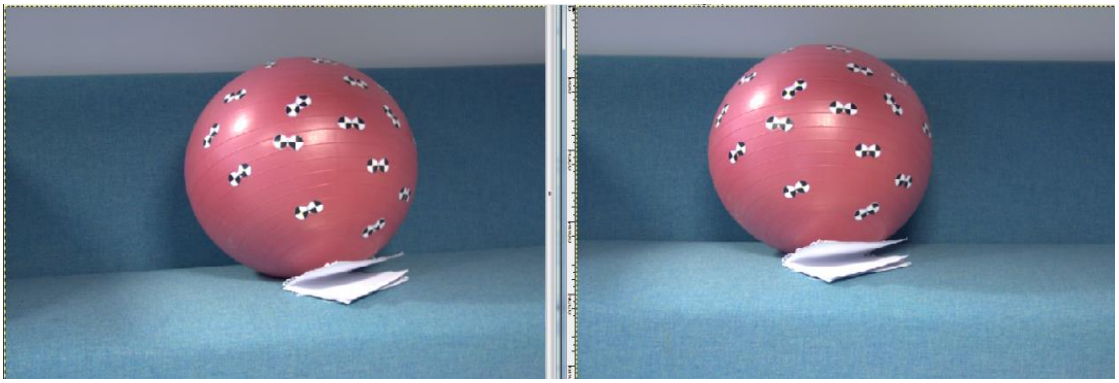


Figure 36: *Two images taken from different angles, at a ball with fiducials. The points in the left image are not on the same height as the points in the right image.*

The images in Figure 36 are two of the input images to the calibration algorithm that is implemented. In Figure 37, the two images are shown after the calibration step. Here, all pixels in one image and their corresponding pixels in the other image are on the same row, as can be seen by the lines in Figure 37.

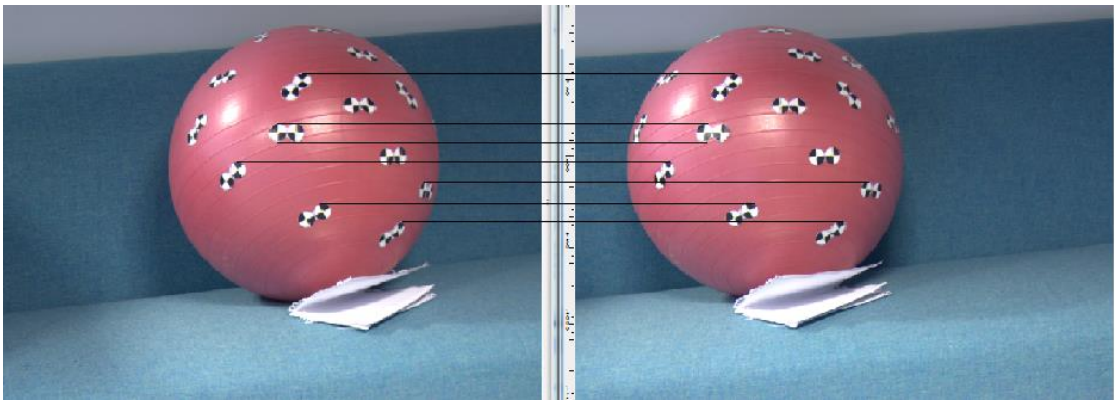


Figure 37: *The same images as in Figure 36, but now the images have been rectified, which mean that a given point in 3D is projected down to the same row in both of the images. The black lines are drawn to highlight this.*

4.3 Pixel matching

The choice of algorithm for pixel matching is very dependent on the object to be reconstructed, if there is a time limit and a minimum error allowed. In this thesis, the goal is to get the best possible reconstruction in sense of error, completeness, small extent of human interaction, and possibility to extend the algorithm to motion capture. The objects reconstructed are mainly human faces, and there is no precise time limit on the algorithm, but it should be reasonably fast to allow for motion capture.

There are two classifications of the algorithms: global and local matching. In the global case, the whole image is considered at once, and in the local case just a part of it at once. The global approach is preferable, for the accuracy, because it can easier solve problems with occlusion and it is not dependent on which order the matching is done. The drawback of the global approach is that it becomes NP-complete and thus very computationally heavy. The local approach is computed in polynomial time and can be implemented to run in real time.

Beeler et al. (Beeler et al., 2010) use a local approach for the matching, and there are some local constraints that must hold. If the constraints do not hold for a pixel, a rematch is done at the depths where that pixel can be positioned according to the constraints. This algorithm is reasonably fast and gives a good result for human faces; therefore, this algorithm is implemented. Some parts in the paper from Beeler et al. (Beeler et al., 2010) leaves much to interpret our interpretation and suggestion to improvements are described in the remainder of this chapter.

As described in Section 3.3.1.1, a hierarchical approach to minimize the texture repetition problem is used. For each layer in the hierarchy, the depth or disparity is computed for each pixel and three constraints are applied on the pixels. The constraints are called ordering, smoothness, and uniqueness. The pixels that did not pass the constraints are rematched, but against fewer pixels, by using the already computed information from their neighboring pixels. The final step is a refinement iteration to acquire sub-pixel accuracy of the matches.

4.3.1 Constraints

This section will describe how the three constraints from Section 3.3.2, can be implemented. Some problems of the implementations are presented and possible solutions are proposed.

4.3.1.1 Ordering Constraint

In Figure 38, two epipolar rows of pixels are drawn below each other, and lines are drawn to show which pixels match. The blue lines are matches from the top row against the bottom row and the red the other way around.

Figure 38: *The above figure shows two rows of pixels in two images. The lines display matches between pixels in the rows. Two lines with the same color that intersects each other imply that the ordering constraint failed in that region.*

Since some of the lines intersect each other in Figure 38, those pixels must have changed their order in relation to each other, as in Figure 29. This cannot happen in a face, which means that some of those matches must be invalid.

The implementation suggested in Section 3.3.2 may not identify all crossing lines. Consider that the first two pixels, from the top row to the left in Figure 39, are miss-matched. The implemented constraint in Section 3.3.2 will fail to identify the first pixel from the left as a mismatch.

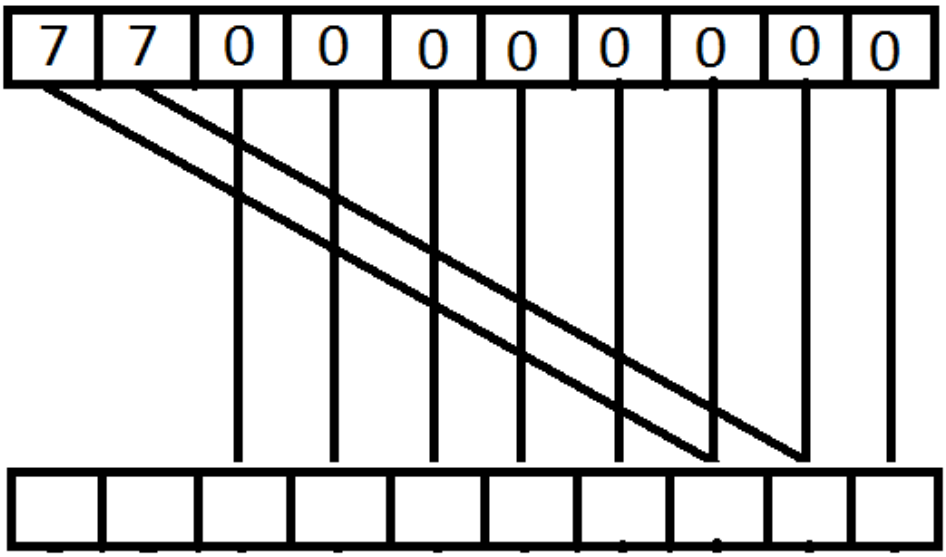


Figure 39: *A row with 10 pixels. The two pixels with disparities 7 are mismatches and should be removed with the ordering constraint.*

One approach used in this thesis was to enforce the ordering constraint on the whole row globally. The greedy heuristic used is to remove the match whose line crosses most other lines until no lines cross each other.

Figure 40 shows the resulting matches after the global ordering constraint has been applied to the case in Figure 38.

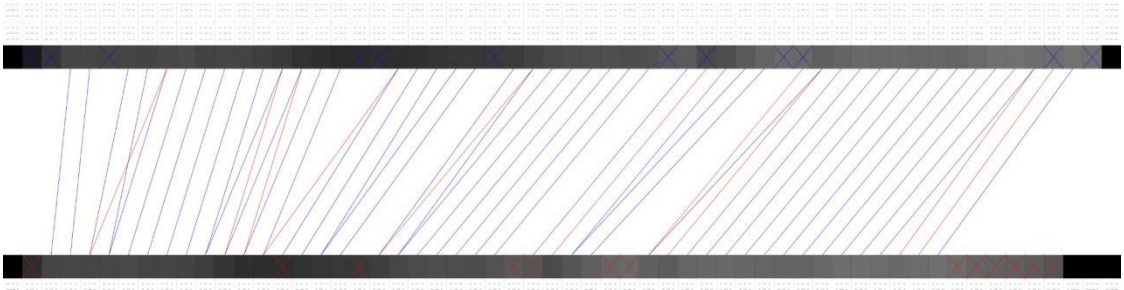


Figure 40: *The same two rows of pixels as in Figure 38. The difference is that now all the disparities that did not fulfill the ordering constraint have been removed.*

4.3.1.2 Smoothness Constraint

In Section 3.3.1.4, the problem of slanted surfaces has been described. In this section, a solution to solve the problem is proposed. The proposed solution is to use a smoothness constraint as described in Section 3.3.2.2. The constraint solves the problem by removing pixels with low depth coherency, which is the case for slanted surfaces.

A problem for how Beeler et al. (Beeler et al., 2010) enforce the constraint is that it is enforced locally, but all the pixels depend on each other. That is, when a disparity of a pixel is to be computed, all the neighboring pixels need to be known. Some of them may have been removed by the smoothness constraint before, and therefore, they

depend on each other. To get around this problem, the constraint is enforced globally by assigning a cost to each pixel according to how they differ from their neighbors in terms of disparity. The cost is then minimized by a minimization algorithm.

4.3.1.3 Uniqueness Constraint:

The uniqueness constraint is implemented as in Section 3.3.2.3 with a few extensions.

The threshold of the uniqueness constraint (described in Section 3.3.2.3) requires more attention. Consider a threshold of up to one disparity and see the two cases in Figure 41 and Figure 42.

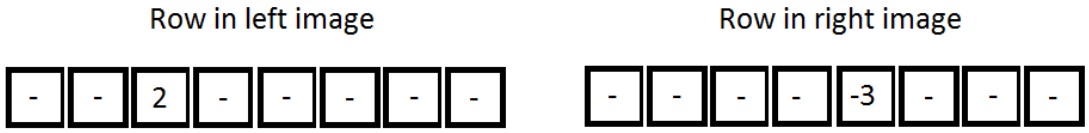


Figure 41: *Two rows of disparities from different images. The '–' means that the pixel does not have any valid disparity.*

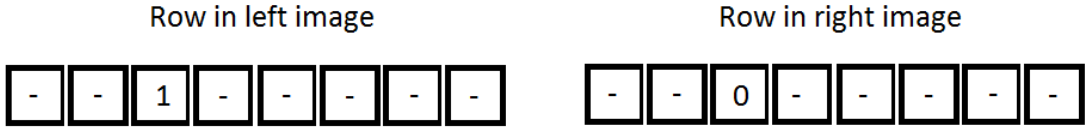


Figure 42: *Two rows of disparities from different images. The '–' means that the pixel does not have any valid disparity.*

In the first case above, the third pixel in the left image matches against the fifth pixel in the right image, and in return, the fifth pixel in the right image matches back to the second pixel in the left image, which adds up to a uniqueness error of 1, which is below or equal to the uniqueness threshold.

If it were to go the other way around by beginning at the right image, then the fifth pixel in the right image matches against the second pixel in the left image, and the second pixel in the left image does not have any disparity, which means that the uniqueness constraint is not fulfilled. Therefore, for the uniqueness constraint with a threshold to be of any use, it has to be formulated as below:

If a pixel p in image i matches against pixel q in image j, then one of the following has to be fulfilled: one of pixel q's neighbors matches back to p; pixel q matches to any of pixel p's neighbors; or pixel q matches back to pixel p.

4.3.1.4 Search boundaries

The constraints of the objects make some pixel disparities invalid. If there are some pixels that do not pass the constraints, these pixels can be rematched. If the matching is done again without any modification, their match will still be invalid according to the constraints. If the new information from the valid neighboring pixels is taken into account, an interval of possible valid pixels can be calculated. Valid pixels mean

pixels that would pass the constraints.

Figure 43: *A row with disparities.*

Consider that the first pixel to the left in Figure 43 has index 0 and the last pixel to the right has index 7. This implies that the first pixel match to index 10, since the definition of disparity is that a pixel's index plus its disparity is the index of the pixel that it matched against. The rest of the pixels follow this pattern as well. For example, the pixel with index 2 will match against a pixel with index 12 in another image.

The ordering constraint will then remove the pixel with index 3, and disparity 8, which results in the pixel row shown in Figure 44.

Figure 44: *The same pixel row as in Figure 43 but the fourth pixel has been removed due to the ordering constraint.*

The pixel with index 3 in Figure 44 will now be re-matched, and according to the ordering constraint (see Section 3.3.2.1), it can only have a disparity between 9 and 11.

If there is more than one constraint, the boundaries will be the interval that is common for all constraints.

4.3.2 Refinement sub pixel.

To achieve even more precision of the disparities, sub-pixel refinement can be calculated. The problem here is not the sub-pixel calculation, since the disparities can be interpolated between pixels, which works perfectly fine (Campbell & Vogiatzis & Hernandez & Cipolla, 2008). The problem is that no matter how small the delta between the discrete disparities to be tested is, it will always be discrete, and the smaller the delta, the more resources it will take to compute.

Beeler et al. (Beeler et al., 2010) solves this problem by, given a pixel p in image i and its match, pixel q in image j, then the correlations between p and q´s left and right neighbors are also tested. The results are three correlations. A second degree polynomial is fitted onto these three correlations.

This is, however, just an approximation which only works well in areas with small texture changes. On edges, the result would be that the matches are drawn away from edges (see the pixel matrix in Figure 45 – for simplicity the size of the blocks used in the template matching is 1x1).

8.2	9.2	10.2	100.2	100	7	8	9	10	100.2

Figure 45: *A pixel row from two different images with sub pixel values.*

The problem that occurs now can be explained as follows: if a polynomial is to be fitted on the correlations between the third pixel in the left images with the third, fourth, and fifth pixel in the right image, then the correlation would be very low, and the resulting maximum in the second degree polynomial would be somewhere between the third and the fourth pixel in the right image, which is not correct.

On the other hand, if the first pixel in the left image would match against the second pixel in the right image, and if the three correlations between the first pixel in the left image and the first, second and third pixel in the right image were computed, then the first pixel has a grey scaled color of 8.2, which is 1.2 from 7 and 0.2 from 8 and 0.8 from 9, which means that the maximum curve would probably lie somewhere between the second and the third pixel in the right image but closer to the second than the third, which seems correct.

Figure 46 shows two models. The one to the right is without sub-pixel refinement and the one to the left is with sub-pixel refinement.

Figure 46: *The above image displays two meshes. The mesh to the right has been created without any sub-pixel disparities or refinement, and it is very apparent with the discrete levels on the surface. The left image has been refined with both a photometric and smoothness term. The dataset used to create the above meshes was given by Beeler et al. (Beeler et al., 2010).*

4.3.3 Discussion

Most of the time, during the work with this thesis, has been used into the disparity map research and implementation. The hardest problem to get robust is the feature matching. To get a small feature matching routine running is very simple. An example could be to just match single pixel colors where the best match is kept. This could solve the pixel matching problem perfectly for an ideal set of images in an ideal setup, where the object to reconstruct is a completely flat surface that is exactly parallel to the cameras.

In the beginning of this thesis, the pixel-matching problem was underestimated, as well as the importance of the environment and the devices used. We now understand that those factors are equally or even more important than the software used for 3D reconstruction.

A reason why this is such an interesting field is because it solves the same problem as, for example, the very expensive laser scanners that only can be used in sequence. Further improvements could also have been implemented. One improvement that has been studied in this thesis is called multi-view pixel matching, meaning that, instead of matching a pixel in one camera against pixels in one other camera, the pixels are matched against pixels in all cameras that are close to the reference camera and letting the different cameras vote for the best match. This method is more likely to find the correct match.

4.4 Texturing

In most applications, other than perhaps motion capture, the objects reconstructed needs textures.

A very simple routine could be used to texture a reconstructed object. Since the system is multi calibrated, it means that the reconstructed mesh can be projected down to every image. The problem is that, usually not a single camera sees all the points on an object (except for very simple objects), and even if it does, the angle between the camera direction and a point’s normal on the surface of the object might differ a lot. This means that there are few pixels in that image to texture a large area on the surface.

A solution to the problem is to, for each vertex in the reconstructed mesh, use the camera with the least difference in angle, between the normal of the vertex and the direction of the camera that sees that point.

Texturing puts a lot of constraints on the lighting used when reconstructing due to that specular and reflective materials have a color depending on the viewing angle. Without ideal lighting, there are specular bright spots that give the final texture small differences in colors depending on the image used to texture that area.

5 Result

The result of this thesis is a system that, given a set of cameras, a ball with fiducials, and a set of images of a face, produces a 3D model of the face. The face can be reconstructed in very high detail (see the Figures 48 and 50 for some results that were created using our implemented software). The system does not require any manual input, neither in the calibration nor the feature matching (see Sections 3.1 and 3.3).

Figure 47 shows the different levels in the depth map hierarchy of a head. Level 0 has the lowest resolution and level 5 has the original resolution.

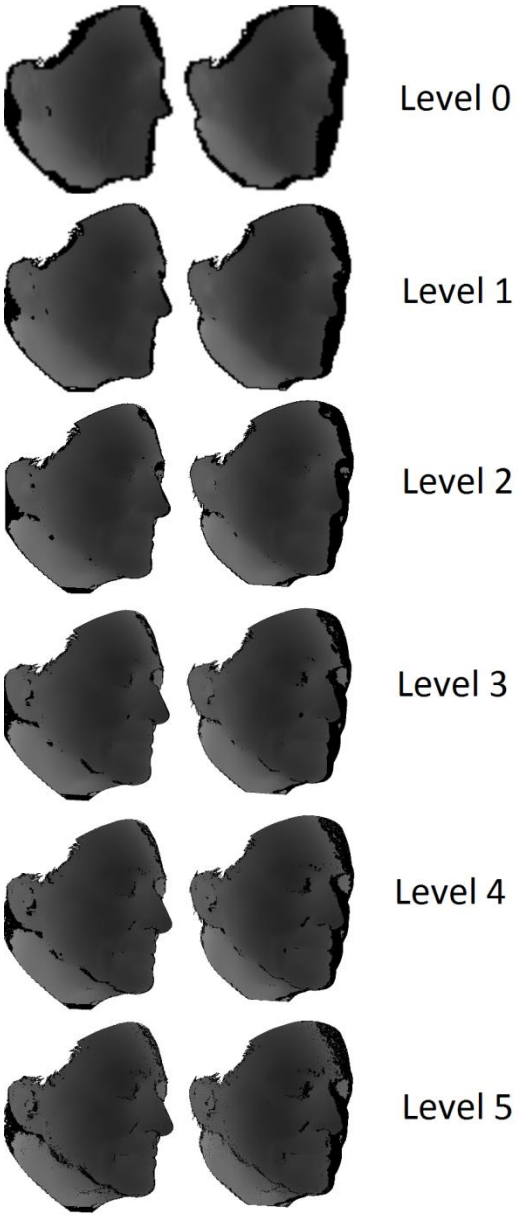


Figure 47 *The above images show a depth hierarchy of a face. Level 0 has the lowest resolution and level 5 has the original resolution. Notice how the occluded pixels get identified and removed as the resolution is increased, especially at the eyes. The depth maps were created of the dataset given by Beeler et al. (Beeler et al., 2010).*

Figure 48 shows three images of a mesh, reconstructed using the implemented software in this thesis, viewed from three different angles. The mesh can be displayed in modeling programs such as 3D Studio Max (Autodesk Inc., 2011a), Maya(Autodesk Inc., 2011b), and Blender(Blender Foundation, 2011).

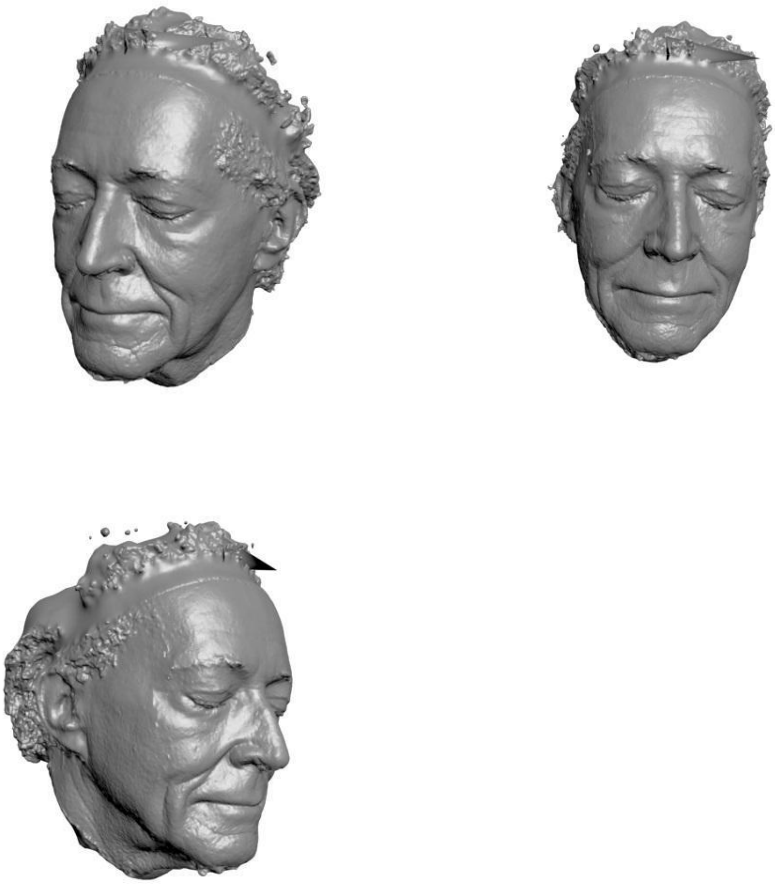


Figure 48: *Three different angles of a mesh of a face. The mesh was created of a data set given by Beeler et al. (Beeler et al., 2010). The mesh has about 10 million triangles and is created by the algorithm implemented in this thesis.*

A depth map hierarchy (see Section 3.3.1.1) of an image in the temple set (see Figure 51) can be seen in Figure 49. The point cloud produced by these depth maps can be seen from three different views in Figure 50. Notice how the stairs among other details is visible.

Figure 49: *The image above displays a depth hierarchy of a temple data set. Notice the missing depths between the pillars; this is because the depths differentiate too much for the smoothness constraint to pass (see Section 3.3.2.2).*

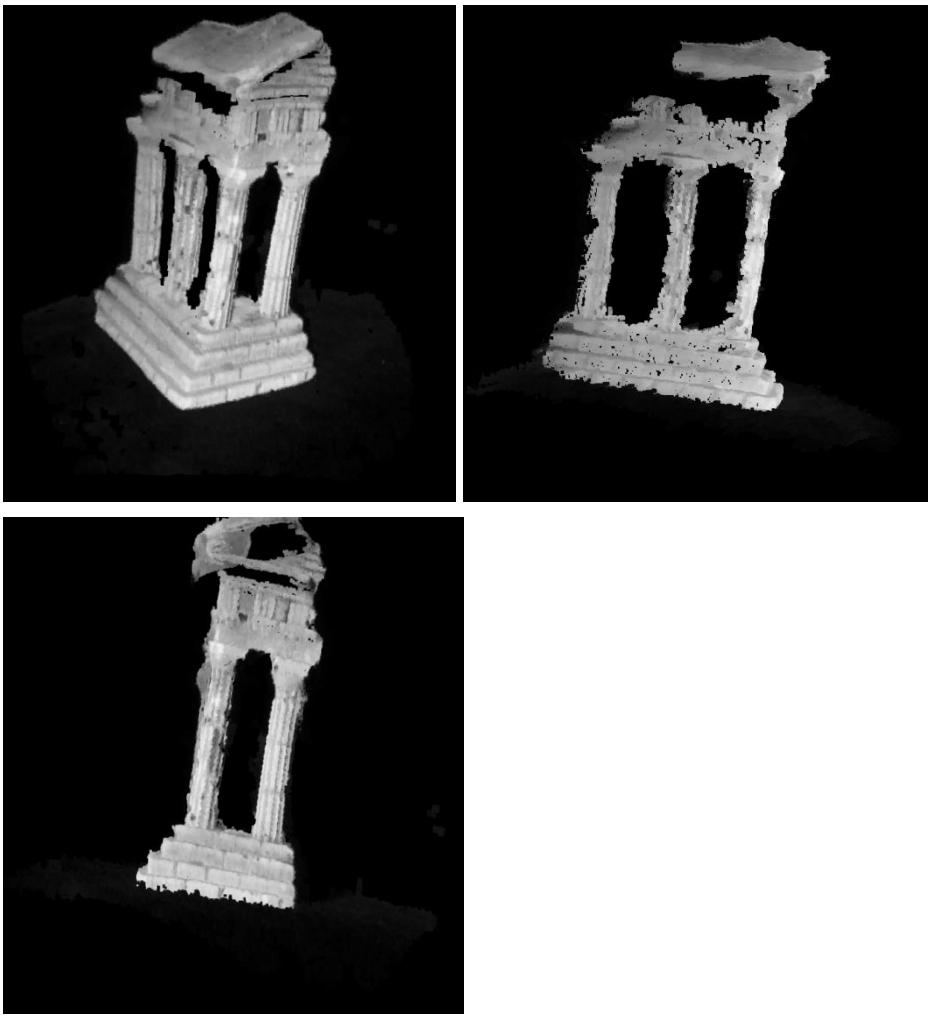


Figure 50: *Three different views of a point cloud created using the depth pyramid in Figure 49. The colors are sampled from the image.*

Figure 51: *An image from Middlebury data set (Scharstein) that was used to create the depth hierarchy and the point cloud in Figure 49 and Figure 50.*

6 Discussion

A large part of this thesis has been spent on understanding the limits and problems of 3D reconstruction and how to solve them. In the beginning of this thesis, implemented solutions were used, such as the block matching algorithm in OpenCV, for feature matching, and the problems were not to implement the actual code but to interpret the result.

An important lesson learned is that the environment and the cameras are at least as important as the software algorithms. The different 3D reconstruction algorithms studied and implemented in this thesis are based on the same core technique of 3D reconstruction (except for the visual hull solution, see Section 3.2), and the differences between them are how robust they are against different objects, lights, and environments.

Certain objects cannot be reconstructed very well, or as they appear in reality. One example is a pile of sand. In reality, the sand is a high amount of small rocks, but when reconstructed, it will look like a rock with highly detailed pattern on it. Another example is to capture hair realistically. Hair is semi-transparent and very thin, which implies that two cameras will not see the same color on a hair strand. Another problem is that strands of hair occlude each other as with the pile of sand. This means that no matter how well the hair is reconstructed, the hair strands will be connected, in the mesh, where they should not be.

The objective of this thesis was to reconstruct objects as a substitute for manual creation of models. However, the models created with the 3D reconstruction software implemented in this thesis may not be well suited to modify or to work with, due to the high amount of triangles, no matter the surface.

There is not currently any single best solution for doing 3D reconstruction. It depends on the system and on the object being reconstructed. This has been learned from our implementations and the constraints that only work for certain objects and images. For example, Hernandez et al. (Hernandez & Schmitt, 2003) has a good algorithm for systems with a high amount of cameras with high pixel coherency between the cameras, but not as good on systems with a sparse amount of cameras (Campbell & Vogiatzis & Hernandez & Cipolla, 2008). Beeler et al.'s algorithm (Beeler et al., 2010) works well for faces that fulfills the ordering constraint and that have very few occluded pixels between the cameras. Neither of them works at all on specular surfaces, while the visual hull algorithm does. On the other hand, the visual hull algorithm cannot reconstruct features that are not visible on a silhouette.

One suggestion could be that several algorithms are used to reconstruct an object and their point clouds are combined into a single point cloud. However, it might be problematic to combine them and determine which of the points that is correct. The chances are that the result would be worse than both of the algorithms, and there is also the additional computation time.

The quality that can be achieved on the reconstructed mesh depends on the cameras used to reconstruct, the object being reconstructed, and the distance between the camera and the object. Without any sub-pixel algorithm, it is possible that a pixel at position (x,y) matches against a pixel at position (i,j) when it actually should have

matched to position (i+0.5,j). Then, the depth on that pixel will differentiate with half a pixel. This depth depends on the intrinsic and extrinsic parameters of the cameras as well as of the depth between the point and the camera.

7 Future work

The majority of the time spent during implementation in this thesis has been to make the 3D reconstruction system more robust, and for future work, the algorithms can be made even more robust.

A voting system when merging the point clouds is a high-priority future work. If several point clouds, from different cameras, agree that a point is on the same position, then it is likely that that point is valid. This approach will remove bad points from specific cameras.

Another interesting topic is to use colored lights, around the actual object to be reconstructed (Hernandez et al., 2007), to estimate normals. Depending on the color, an approximate normal direction can be calculated. A point lit completely with red and nothing with blue and green would yield a normal N = (1,0,0). With this normal map, it could be estimated how well a certain camera sees a point, or even if it is pointing away from that camera (occluded). The color will also act as an artificial texture to help with the matching of pixels by forcing the normals to be approximately the same.

The number of triangles can also be reduced for flat surfaces in the reconstructed mesh so that they simulate manually created meshes. One way to reduce triangles of, for example, a brick wall with high detail would be to use a quad (i.e., two triangles) and a bump map. Then, it would be possible to visualize in real time, while still being able to see the high detail in the forms of shadows and lighting.

8 References

Ansoldi, S. (2005) *The 3D reconstruction algorithm*, http://www-dft.ts.infn.it/~ansoldi/Research/3DstereoNetService/HTML/, node2.html, (2013-05-09).

Autodesk Inc. (2011a) *3ds Max –3D Modeling, Animation, and Rendering*, http://usa.autodesk.com/3ds-max/, (2013-05-09).

Autodesk Inc. (2011b) *Maya–3D Animation, Visual Effects & Compositing Software*, http://usa.autodesk.com/maya/, (2013-05-09).

Ballan, L. & Cortelazzo, G. M. (2008) 'Marker-less-motion capture of skinned models in a four camera set-up using optical flow and silhouettes', *Proceedings of 3DPVT'08.*

Beeler, T., Bickel, B., Beardsley, P., Sumner, B. & Gross, M. (2010) 'High-Quality Single-Shot Capture of Facial Geometry', *Proceedings of ACM SIGGRAPH* , Los Angeles, USA, July, *ACM Transactions on Graphics*, vol. 29, no. 3, pp. 40:1-40:9

Blender Foundation (2011*) Blender.org Home,* http://www.blender.org/features-gallery/features. (2013-05-09).

Camera Calibration and 3D Reconstruction (2009) http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html, (2013-05-09).

Camera Calibration With OpenCV (2011) http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, (2013-05-09).

Camera calibration with square chessboard (2013) http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration_square_chess/camera_calibration_square_chess.html, (2013-05-09).

Campbell, N. D. F., Vogiatzis, G., Hernandez, C. & Cipolla, R. (2008) 'Using Multiple Hypotheses to Improve Depth-Maps for Multi-View Stereo', *Proceedings European Conference of Computer Vision (ECCV),* 2008.

Casey, C. J., Hassebrook, G. H. & Lau, D. L. (2008) ‘Structured Light Illumination Methods for continuous motion hand and face-computer interaction, *Human Computer Interaction: New Developments*, Kikuo Asai (Ed.).

Chen, T., Liu, Y., Hsieh, W. & Hu, Y. (2012) ’BLOCK MATCHING METHOD’, Patent Application Publication, Pub. No.: US 2012/0224749 A1.

Cignoni P., Montani C. & Scopigno R. (1992) ‘A new Merge-First divide & conquer algorithm for ed delaunay triangulations’, *internal Report C92/16. Internal note CNUCE-B4-92-016*.

Duraiswami, R. (2000) *Camera Calibration,* University of Maryland: Institute of Advanced Computer Studies, http://www.umiacs.umd.edu/~ramani/cmsc828d/lecture9.pdf , (2013-05-09).

Faugeras, O. D., Luong, Q. T. & Maybank, S. J. (1992) ‘Camera self-calibration: Theory and experiments’, *Second European Conference on Computer Vision Santa Margherita Ligure*, Italy.

Fischler, M. A. & Bolles, R. C. (1981). ‘Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography’, *Commun. ACM*, vol. 24, no. 6. June, pp. 381-395.

Fisher, R., Perkins, S., Walker, A. & Wolfart, E. (2003) *Laplacian/Laplacian of Gaussian. Spatial Filters - Laplacian/Laplacian of Gaussian*, http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm, (2013-05-09).

Forsyth, D. A. & Ponce, J. (2003) Computer Vision: A Modern Approach, Prentic Hall.

Frey, J. P. (2004) Generation and adaptation of computational surface meshes from discrete anatomical data. INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, vol. 60, pp. 1049-1074.

Goodman, J. E. & O´Rouke, J. (2004) *Handbook of Discrete and Computational Geometry, Second Edition*, editors CRC Press LLC, Boca Raton, April.

Grimm, C. & Goldman, K. (2010) CSE131 *Perspective projection*, Washington University

Hernández, C. & Schmitt, F. (2003) Silhouette and Stereo Fusion for 3D Object Modeling. Computer Vision and Image Understanding, Special issue on "Model-based and image-based 3D Scene Representation for Interactive Visualization", vol. 96, no. 3, pp. 367-392, December

Hernandez, C., Volgiatzis, G., Brostow, G. J., Stenger, B. & Cipolla, R. (2007) 'Non rigid Photometric Stereo with Colored Lights'. ICCV, Rio de Janeiro, Brazil, October.

Hii, A. & Hann, C. E. & Chase, J. G. & Van Houten, E. E. W. (2006) Fast Normalized Cross Correlation for Motion Tracking using Basis Functions. Computer Methods and Programs in Biomedicine, vol. 82, no. 2, pp. 144-156.

Inder, K. R. (2003) *From geometry to algebra: An introduction to linear algebra*, India Mumbai: Indian Institute of Technology

Joe, B. (1991) 'Construction of three-dimensional Delaunay triangulations using local transformations'. *Computer aided Geometric Design*, vol. 8, no. 2, pp. 123-142

Kass, M., Witkin, A. & Terzopoulos, D. (1988) 'Snakes: Active Contour Models'. *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321-331.

Kazhdan, M., Bolitho, M., Hoppe, H. (2006) 'Poisson surface reconstruction', *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, Aire-la-Ville, pp. 61-70.

Kinect for Windows, http://www.microsoft.com/en-us/kinectforwindows/ (2013-05-09).

Lorensen, W. E. & Cline, H. E. (1987) 'Marching Cubes: A High Resolution 3D Surface Construction Algorithm'. *Computer Graphics*, vol. 21, no. 3, pp. 163-169, July.

Lowe, D. G (1999) 'Object recognition from local scale-invariant features', *International Conference on Computer Vision*, Corfu Greece, pp. 1150-1157.

Maur, P. (2002) *Delaunay Triangulation in 3D*. University of West Bohemia in Pilsen. Technical Report No. DCSE/TR-2002-02.

Piccardi, M. (2004) *Background subtraction techniques: a review*, University of Technology Sydney: Faculty of Engineering.

Rosenthal, P. & Linsen, L. (2006) 'Direct Isosurface Extraction from Scattered Volume Data'. *EuroVis06: Proceedings of the Eurographics/IEEE-VGTC Symposium on Visualization*, pp. 99–106.

Scharstein, D. (2003) *2003 Stereo datasets with ground truth*, http://vision.middlebury.edu/stereo/data/scenes2003/, (2013-07-27).

Scharstein, D. (2013) *Evaluation*, http://vision.middlebury.edu/mview/eval/ (2013-05-09).

Scharstein, D., *MultiView Stereo*, http://vision.middlebury.edu/mview/data/, (2013-07-27).

Sigal, L. (2012) *Lecture 3 (Marker-based) Motion Capture*, Carnegie Mellon University: School of Computer Science.

Stefano, L. D. & Bulgarelli, A. (1999) 'A Simple and Efficient Connected Components Labeling Algorithm'. *ICIAP*, pp. 322-327.

Svoboda, T., Martinec, D & Pajdla, T. (2005) 'A Convenient Multi-Camera Self-Calibration for Virtual Environments'. *Teleoperators and Virtual Environments*, vol. 14, nr. 4, August .

Szeliski, R. (1992) 'Rapid Octree Construction from Image Sequences', *CVGIP: Image Understanding*. vol. 58, no. 1, July, pp. 23-32.

Takahashi, D. (2013) 'Bringing characters to life with Pixar's animator magic', V*enturebeat*.

Triggs, B., McLauchlan, P., Hartley, R. & Fitzgibbon, A. (1999) 'Bundle Adjustment — A Modern Synthesis', *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*. Springer-Verlag. pp. 298–372

Vogiatzis, G.,Torr, P. H. S. & Cipolla, R. (2005) 'Multi-view Stereo via Volumetric Graph-cuts'. *In Proceedings IEEE Conference on Computer Vision and Pattern*

Recognition (CVPR), pp. 391-398

Wang, Z. & Zheng, Z. (2008) ‘A Region Based Stereo Matching Algorithm Using Cooperative Optimization’. *IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-8.

Watson, D. F. (1981) ‘Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes’, *Comput. J.*, vol. 24, no. 2, pp. 167-172.

Y. Xiao, R. B. Fisher, (2010) ‘Accurate Feature Extraction and Control Point Correction for Camera Calibration with a Mono-Plane Target’, *Proc. Int. Conf. 3D Data Processing*, Paris, May