

universität wien

MASTERARBEIT

Titel der Masterarbeit

Structured light Face-Scan Implementation for OGRE

Verfasser

Bsc. Philip Beutl

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl. -Ing.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 066 935
Studienrichtung lt. Studienblatt:	Medieninformatik
Betreuer:	Ao. Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Abstract

In dieser Arbeit wird mit Hilfe eines interactive Whiteboards ein einfacher Gesichts-Scanner implementiert. Basierend auf dem structured light Prinzip wird mit dem Beamer des interactive Whiteboards ein dünner Lichtstrahl auf das Gesicht einer Person projeziert, der sich langsam über das Gesicht bewegt. Dies wird mit einer handelsüblichen HD-Webcam aufgenommen, die vorher kalibriert wurde. Danach wird das gescannte Gesicht mit Hilfe des Open-Source 3D-Programms Ogre3d dargestellt.

0.Index

1. Introduction.....5
2. Overview structured light scanning techniques.....7
2.1 Basic structured light scanning.....7
2.2 Spatial neighborhood.....8
2.2.1 non-formal codification.....9
2.2.2 De Bruijn Sequence.....9
2.2.3 M-arrays.....11
2.3 Temporal codification.....12
2.3.1 Binary Codes.....12
2.3.2 n-ary codes.....13
2.3.3 Gray code with phase shifting.....14
2.3.4 Hybrid Methods.....14
2.4 Direct codification.....15
2.4.1 color-based codification.....15
3. Related Work.....17
4. Implementation.....28
4.1 Pin-hole camera model.....29
4.2 Camera Calibration.....30
5. Scan Process.....36
5.1 Obtaining the pictures.....36
5.2 Processing the pictures.....37
5.3 calculating the points.....39
5.4 Deleting of impossible data.....41
6. displaying the points.....43
6.1 Meshes.....43
6.2 building the vertex matrix.....44
6.3 adding missing data.....45
7. smoothing.....47
7.1 laplacian algorithm.....47
7.2 weighted laplcacian algorithm.....48

7.3 taubin smoothing.....49
7.4 biplacian flow.....50
7.5 mean curvature flow.....50
8. Displaying the points in OGRE.....51
8.1 Mirror.....52
9. Performance.....54
10. Conclusion.....58
11. References.....59
12.Appendix.....61

1.Introduction

In computer vision there are a lot of applications that can profit from 3D surface reconstruction through scanning. From medical measurements over reverse engineering to usage in computer games there are many fields of application. Due to its many possibilities for usage there already has been a lot of research in the field of scanning and reconstructing the surface of an object. There are two methods which are most commonly used which can be divided in a passive and an active method.
The passive method uses multiple cameras to recreate the surface of an object. It works a bit like the human eyes, where you get a different image from each eye and therefore can estimate the distance of an object. The cameras also take pictures of the same object. Then the extract certain feature points of the image which can be easily distinguished and calculate through triangulation the coordinates of the object surface. A prerequisite is that the cameras have to be calibrated before so that there position and angle are known.
The active method, also called structured light method, uses instead of two cameras only one camera and a light source which illuminates parts or the entire object. Because the object gets illuminated its called active method. There are different methods of structured light mainly different in the pattern used or the light source used, like if it is visible or invisible to the human eye. A basic method to scan an object is to shine a sheet of light onto the object. Because of the intersection, between the sheet of light and the scanned object, there is a stripe of light visible on the object. Due to the objects proportions, the stripe of light gets distorted and this can be captured by the camera. With the help of this distortion we can again triangulate the coordinates of the objects surface. Again the camera and light source have to be calibrated before.

In my Master thesis I want to concentrate on the structured light scanning method. Using this method I want to create a face scanner using an interactive whiteboard and an off-the-shelf HD web cam. I chose to built a face scanner because reconstructing a scanned face has many appliances like using as a avatar in computer programs or for face recognition.
The advantages of the interactive whiteboard are that it has an built-in projector that can be used to project the light plane. And the projector has a fixed place so it only has to calibrated once. Further is the whiteboard an even surface so that no distortions, except the human standing in front are visible.

Commercial scanner can be expensive, although there are already some approaches for an inexpensive scanner using off-the-shelf hardware. This scanner should also be an inexpensive one that only needs a cheap HD webcam additionally to the whiteboard. Although you could argue that an interactive whiteboard itself is a bit expensive, the scanner can be used in schools or universities that already have one available.

The implementation of my Master thesis can be divided into three parts:

1. Calibrating of the camera using the method introduced in [1]
2. Scanning a persons face using the structured light method and calculating the points of the face surface.
3. Representing the face as a 3D model in OGRE, an open-source 3D program.

2. Overview structured light scanning techniques

2.1 Basic structured light scanning

The idea behind structured light scanning is to project a pattern of light on the surface of the object you want to scan. Further you need a camera, which has been calibrated before so that you know its position and angle, to record the scene. The most simple example would be to project a plane of light on the object you want to scan, causing a thin line of light to be seen on the surface of the object. As seen in Fig.1, because of the non flat surface of the object there is be a certain distortion of the light beam. This distortion then can be captured on an image through a specific angle from the camera, and so you can calculate the points of the light beam on the surface through triangulation. When you move the plane of light from one end of the object to the other you can get a scan of the whole object. This method already gets pretty good results

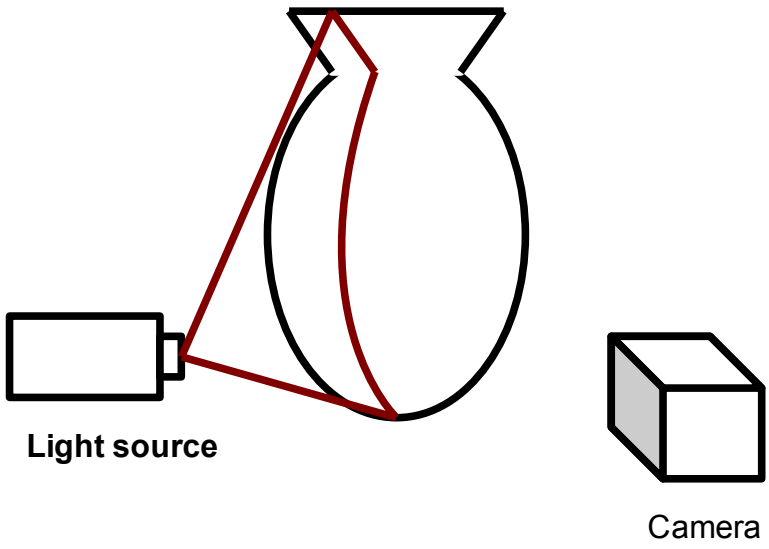


Fig.1 Example of a simple light scan

Of course the thinner the light beam and the slower the movement the preciser the scan will be. But this is also a disadvantage of this method. Because by projecting only a single ray of light, on the object you need to capture, and by moving it step for step it takes a while to finish the scan. And you also need a lot of pictures taken by the camera. Therefore there has already been some research in how to shorten the amount of images needed to finish a scan. Following I want to introduce some

techniques that give a solution to this problem.

Structured light techniques can be categorized into three different categories as proposed in [8] and each of them can be again divided into different pattern methods. An overview can be seen in Fig.2.

The three categories are:

- Spatial Neighborhood
  - Non-formal codification
  - De Bruijn Sequences
  - M-arrays
- Time-multiplexing
  - Binary Codes
  - N-ary Codes
  - Gray code with Phase Shifting
  - Hybrid Methods
- direct coding
  - color-based codification

2.2 Spatial Neighborhood

Spatial neighborhood means that the points of a certain pattern get encoded with the information of the points in their neighborhood. This can be achieved for example based on the colors or the intensity of the points. This category we can divide again into three methods commonly used, non-formal codification, codification with De Bruijn sequence and codification with M-arrays (also known as pseudo-random arrays). The advantage of this category is that one-shot scans are possible, which means that the patterns only need to be projected once to capture the object. This makes it also applicable to dynamic scenes with moving objects. Disadvantage is, that in the decoding stage when errors happen, it has also impact on its neighbors because the decoding relies on their information.

2.2.1 Non-formal codification

Non-formal coding means to encode the pixels without using any mathematical coding theory like a previous existing algorithm. An example for this method is given in [11]. There they suggested a pattern suitable for scanning a human trunk, and capable of looking for abnormalities like scoliosis. The pattern used was generated by combining three color values. 255 stood for white (w), 127 for gray (g) and 0 for black (b). Always combining four pixel values so that B = bbbb, W = wwww and G = gggg they could combine them by simply changing the order: BWG, WBG, WGB, GWB, GBW, BGW. Like this 72 pixel where encoded and then the pattern was repeated as seen in Fig.3

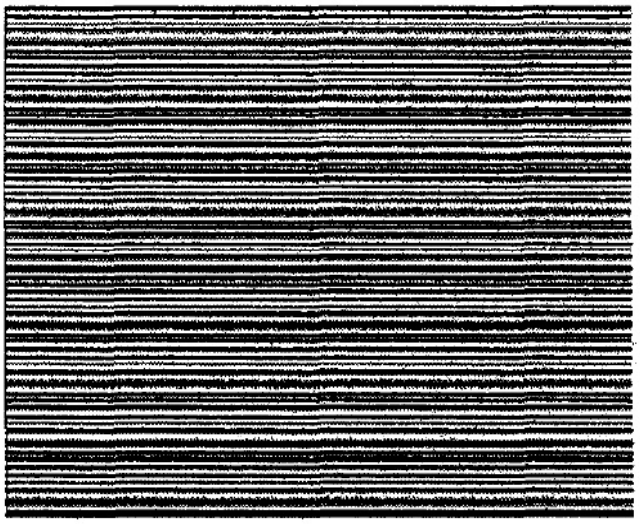


Fig3. Pattern generated in [11]

2.2.2 De Bruijn Sequences

The de Bruijn sequence is named after the Dutch mathematician Nicolaas Govert de Bruijn.

"A de Bruijn is a sequence of order m over an alphabet of n symbols is a circular sequence of length n^m that contains each sub-string of length m exactly once." [8]

Such a sequence can be obtained for example by running an Eularian circuit or a Hamilton path over a de Bruijn graph. As explained in [31] a de Bruijn graph is a directed graph representing overlaps between sequences of symbols. To create the directed edges you have to look if a vertex v_i can be represented by another vertex v_j by shifting the characters of v_i one place to the left and adding a new character. An example with for this with $n=2$ and $m=2$ can be seen in Fig.4.

An Eularian circuit is used to get the De Bruijn sequence 0011.

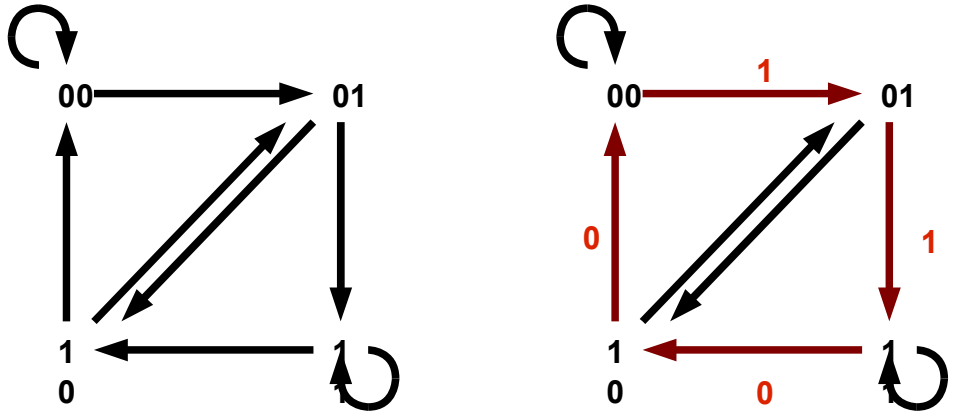


Fig.4 Left side a de Bruijn graph with m = 2 and n = 2 and on the right side a de Bruijn Sequence by using an eularian circuit.

In [9] a de Bruijn sequence is used to get a color sequence with 125 stripes with the challenge that three consecutive color transitions had to be unique. The color values had a binary value in RGB color space, for example 010 which would stand for green. To calculate the next color XOR was used, for example if we XOR 010 green with another color like red 100 we get yellow 110. So to get the color sequence a XOR pattern had to be found where three consecutive color transitions had to be unique. For this the de Bruijn sequence could be used with $n=3$ and $k=7$, $k \in (001...111)$. Because only a sequence of 125 stripes was needed 110 and 111 were dismissed, resulting in $k=5$. Then a XOR pattern d_j was created from a pool of k elements where any sub-sequence n only appeared once. With this we can build a color index sequence by choosing a start color value c_j from (001...111) and using the iteration $c_{j+1} = c_j \ XOR \ d_j$.

In Fig.5 we can see an example of a color sequence. The method in [9] could be used with multiple images but also for one-shot captures which can be useful for scanning moving objects.

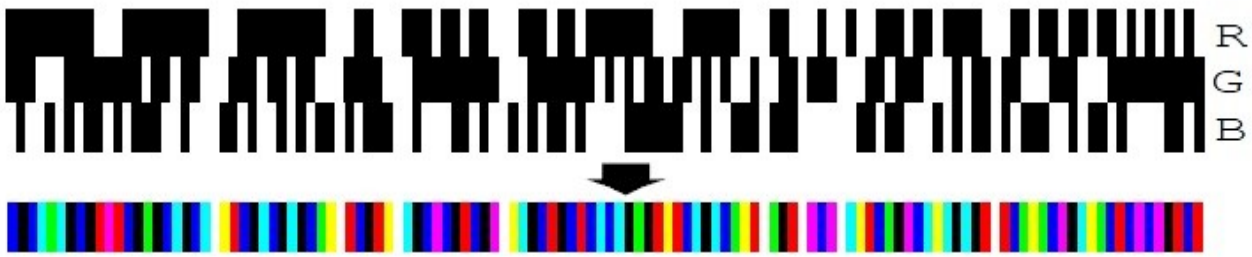


Fig.5 A de Bruijn sequence for which each consecutive color transitions are unique [9]

In [7] an optimized de Bruijn sequence was introduced where not only the hue of a color but also the luminance value was used for encoding. As seen in Fig.6 we can differentiate between the hue pattern and the luminance pattern. This gives the advantage to increase the size of the de Bruijn sequence from n^m to 2*n^m and also got rid of the limitation that two hue values could not be next to each other. Because now they could with different luminance value. The advantage of increasing the size of the sequence also means that less colors can be used and so the colors are easier to distinguish. That optimizes the accuracy and resolution of the pattern.

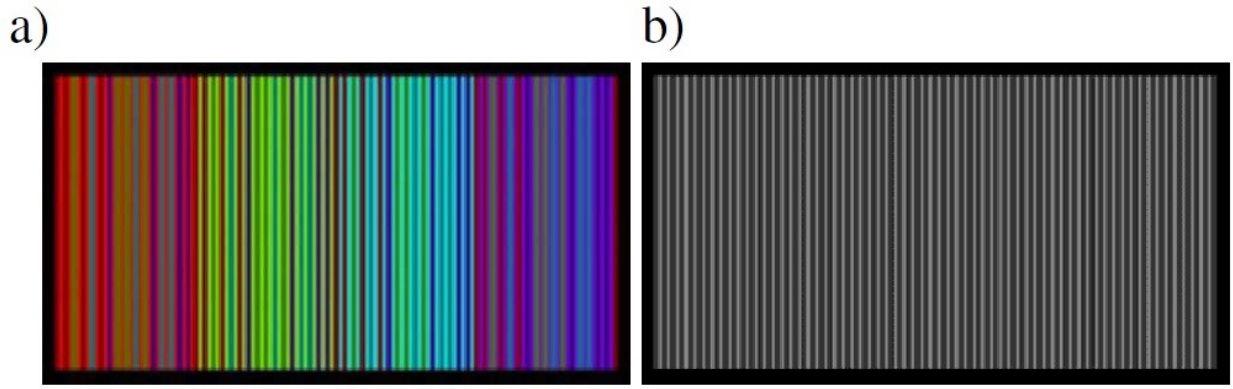


Fig.6 a. Is the color pattern b. Is the luminence pattern [7]

2.2.3 M-arrays (pseudo-random arrays)

M-arrays are pseudo-random array patterns in which a sub-array of size a x b only exists once in the whole pattern. In [30] an algorithm called "piece growing" was introduced to create M-arrays. The idea is to create a number of n*m sub-arrays with the condition that none of them are alike. Then by splicing them together in a specific order and only if the sub-array doesn't already exist in the array, the M-array is created.

In [12] a M-array was presented that uses a monochromatic pattern instead of using multiple colors for encoding. The advantage of a monochromatic pattern is that it is robuster than using multiple colors because it is easier to distinguish. The pattern is based on three symbols, a disc, a circle and a stripe. Using 783 of these symbols it designs 3x3 sub-arrays all of them unique. To further improve the robustness the pattern was designed that even if the upper left corner or the lower right corner of a sub-array was missing it would still be unique. The results showed that even with interference of a white light source still up to 80% of the detected symbols were decoded correctly.

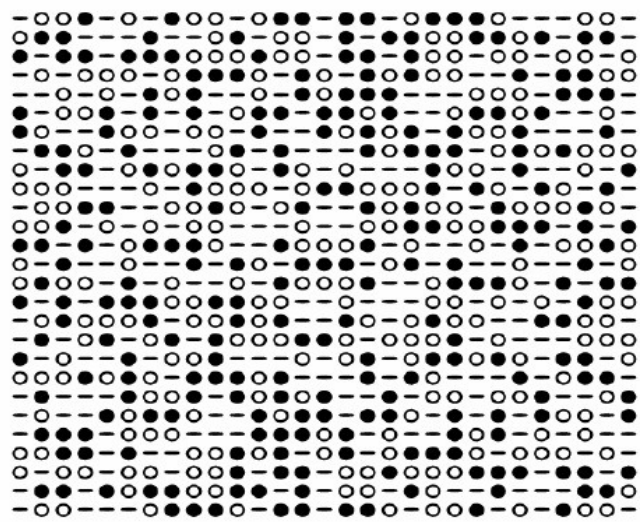


Fig 7. Monochromatic M-array used in [12]

2.3 Temporal codification (time multiplexing)

In temporal codification a sequence of pattern gets projected on to the to scanning object. So over time a sequence of values gets assigned to every pixel. It is a very commonly used method that we can divide into four categories. Binary Codes, n-ary Codes, Gray code with Phase shifting and Hybrid methods. The advantage of time multiplexing is that it usually has a good resolution and precision. The disadvantage however is that due to the use of multiple patterns, one-shot scans are not possible.

2.3.1 Binary Codes

Binary Codes uses as it name states only two values to encode the data, 0 and 1. For this the technique uses two colors, mostly black and white to encode the values The process starts by projecting only two thick stripes, one black and one white. Every pixel which has the color white has the value 1 and every pixel with black color gets the value 0. After a picture was taken the next time the projector projects four stripes, two black and two whites in alternate order. Again white stands for 1 and black for 0. So now the pixels could have four different values: 00, 01, 10, 11. At the next step 8 stripes get projected 4 whites and 4 blacks again in alternate order. This can be done for a sequence of m patterns to encode 2^m stripes, where in the end the last stripes, each has its own binary code. In Fig.8 we can see a nice picture of the binary code projection taken from [13].

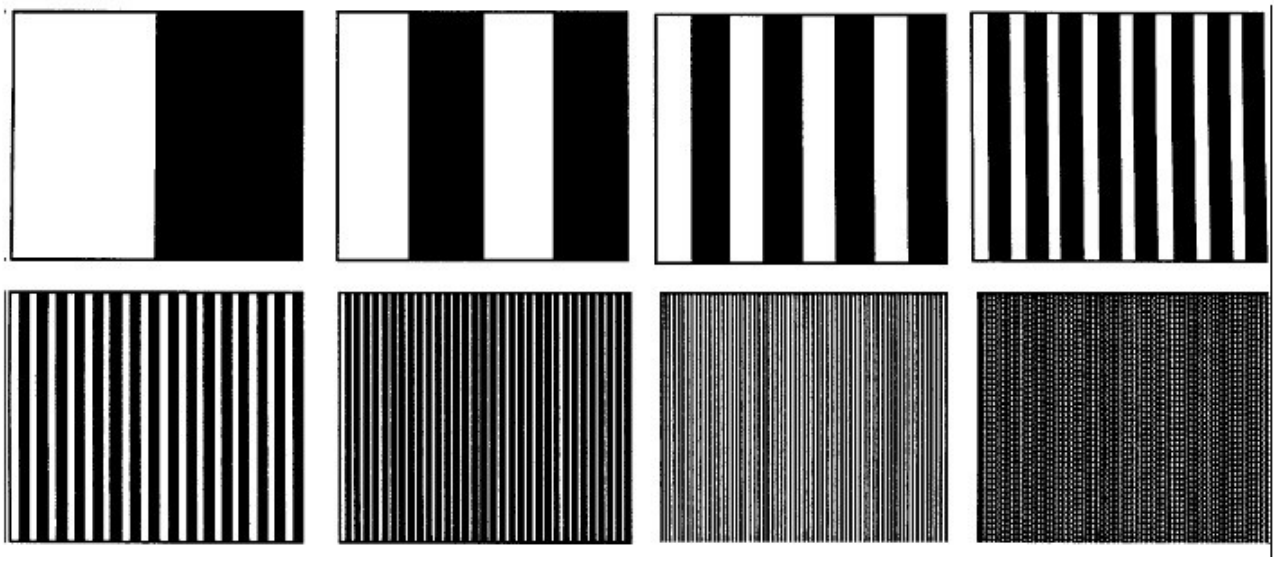


Fig 8. Binary Code patterns used in [13]

2.3.2 N-ary Codes

An disadvantage of binary codes is the amount of patterns you need to encode all pixels. This could be reduced by starting with more unique stripes at the beginning. You could use more colors or different gray values for the stripes. By starting with four stripes and using m patterns you could encode 4^m stripes, by using eight stripes at the beginning you could encode 8^m stripes. Therefore the amount of patterns used can be reduced. In [14] a ternary and quarter-nary scanner were used for scanning using different colors.

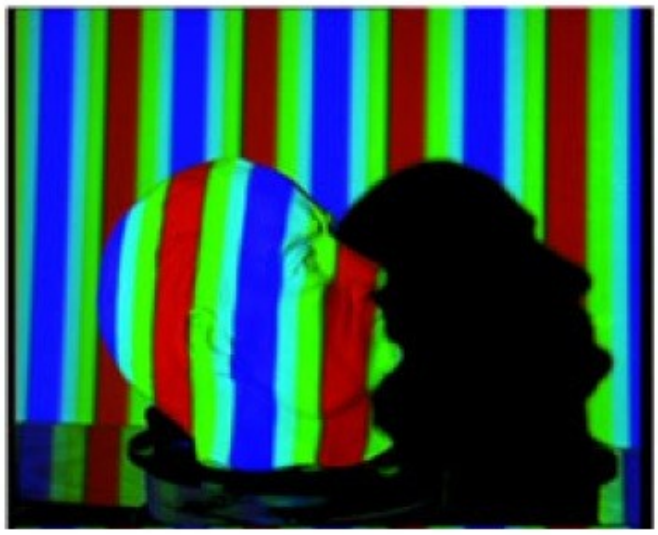


Fig 9. Quarter-nary scan pattern [14]

2.3.3 Gray code and Phase shifting

Phase shifting uses phase-shifted pattern. A very common approach is the three-step phase-shifting algorithm which uses three phase-shifted sinusoidal pattern. The equations of the intensities (1) can be written as described in [32]:

$$\begin{aligned} I_1(x,y) &= I'(x,y) + I''(x,y)\cos[\phi(x,y) - \alpha] \\ I_2(x,y) &= I'(x,y) + I''(x,y)\cos[\phi(x,y)] \\ I_3(x,y) &= I'(x,y) + I''(x,y)\cos[\phi(x,y) + \alpha] \end{aligned} \tag{1}$$

where $I'(x,y)$ is the average intensity, $I''(x,y)$ is the intensity modulation, $\phi(x,y)$ is the phase and α is the phase step size.

With the recorded intensities the phase angle of a pixel within a wavelength can be calculated.

Gray code and Phase shifting as the name states, is a combination of gray code and phase shifting techniques as described in [33]. The advantage is a better resolution and range of measurement. The disadvantage in this method is again the number of pattern used.

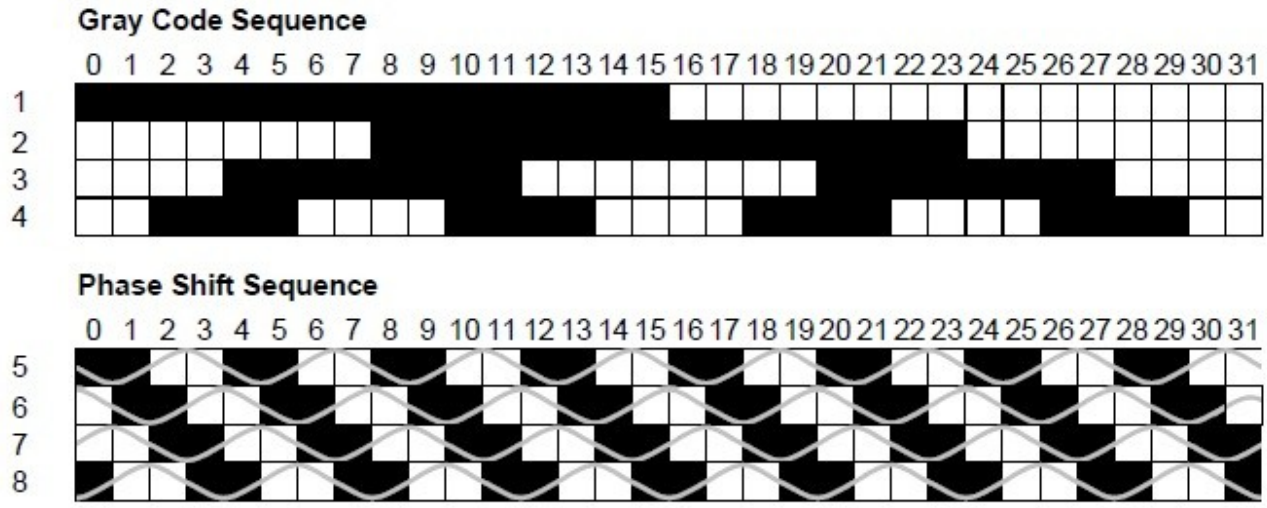


Fig 10. Example of Gray Code and Phase shifting used in [15]

2.3.4 Hybrid methods

Hybrid methods use elements of spatial neighborhood and time multiplexing. An example for hybrid method is presented in [10]. They divided four patterns into 111 vertical stripes using only black and white. The interesting idea was that the encoding wasn't the color but the border between

two stripes. So this method not only uses time multiplexing, by using different patterns over time, but also spatial neighborhood by looking at multiple points at the same time. In the implementation described four frames were used, as seen in Fig.11. This method also can be used for slow moving objects.

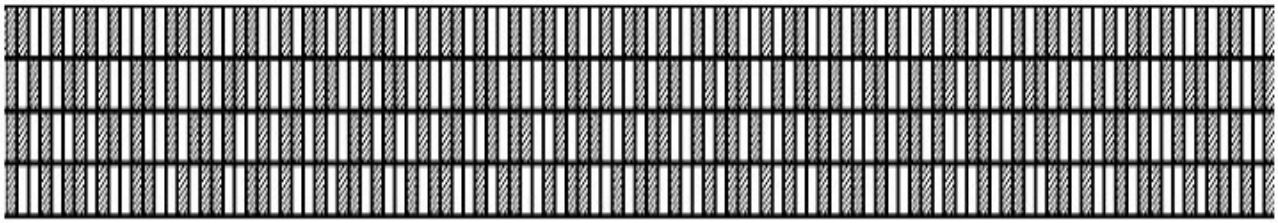


Fig.11 Used rows of Frames from [10]

2.4 Direct codification

In direct codification each point gets encoded and identified by itself. This can be done by either using many grey levels or using many colors. Advantage of this method is a good spatial resolution but the disadvantage is the sensitiveness to light and noise which can easily alter the scan.

2.4.1 Color base coding

Color base coding can be done by using a large variety of colors. In [18] a direct coding approach was presented by using a rainbow range finder (RRF).



Fig.12 usage of Rainbow Range finder [17]

A RRF is a range finding system that can find the range for all image pixels based on spectral analysis. The principle behind this is the same as basic structured light scanning. Instead of using

only one plane of light, now we use many planes of light in different colors. If we can differentiate between the different colors we can calculate the coordinates of an illuminated object. In [16] a RRF was used and the scanning showed good results of only 0.2-0.3 mm error by a 150mm*100mm*100mm measurement size.

3. Related Work

Structured light scanning can be used for various fields of application. For example in medical treatment where the reconstruction of certain body parts could be helpful, or for face recognition which can be used for various security application. In my master thesis I wanted to implement a scanner optimized for face scanning and there have already been some good approaches for this challenge. Some are using structured light methods like the de Bruijn pattern but others use completely different methods like morphable models. Following I want to mention some of them briefly.

In [19] they implemented a face scanner using a color based stripe pattern. The colors used were red, green, blue, white, cyan, magenta and yellow as seen if Fig.13. A constraint was that two color stripes next to each other had to be different in at least two channels. In between the color stripes black stripes were used to easier distinguish the color stripes. To generate the pattern a depth-first search was used.

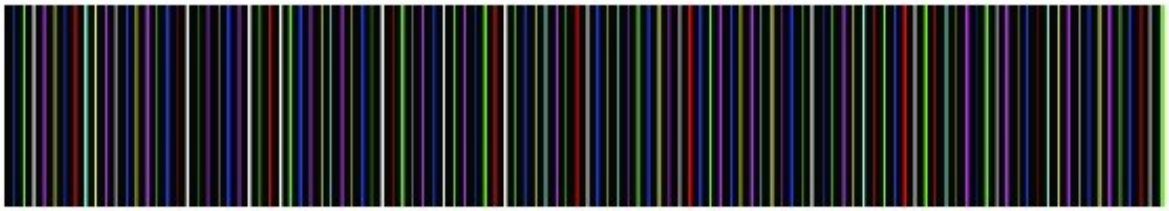


Fig.13 Pattern used in [19]

The procedure consisted of following steps. An Image was taken of the color pattern projected on the face of a person. A second image could be taken without the color pattern to use later as the texture of the face. Then the prospective stripes were extracted. To get the region of interest a simple face shaped mesh was used. All the other pixels outside the region of the mesh were set to black. So only the remaining colored pixel were searched for the projected stripes. To get sub-pixel resolution the raw CCD sensor format was used. Which means there were no RGB pixel values but columns of single-channel pixels with alternating color sensitivity depending on the sensor type used (red, green, blue). The prospective stripes were extracted for each scan column and each color channel. Then the center of the stripes were measured with parabolas fitted through their intensities. Afterward the different color channels were merged to one common list, by calculating the common

center. After the three lists get combined to a common list, the probability of a correctly recovered projected stripe is calculated and stripes which derivate to much are canceled out. When the prospective stripes were known the corresponding colors were extracted and the prospective stripes were matched with the projected stripes. This was done by calculating which combinations of correspondences fitted best. For that the likelihood to be a valid stripe, the likelihood that the stripe was projected with the corresponding pattern color and the likelihood that this color occurs in the sequence of the projected pattern had to be taken in account. After the matching of the corresponding stripes the 3D coordinates can be calculated. The result was a triangle mesh that could be displayed as wireframe model, surface or textured face model. In summary this method presents a robust, high accurate one-shot Scanner with good results as seen in Fig.14.

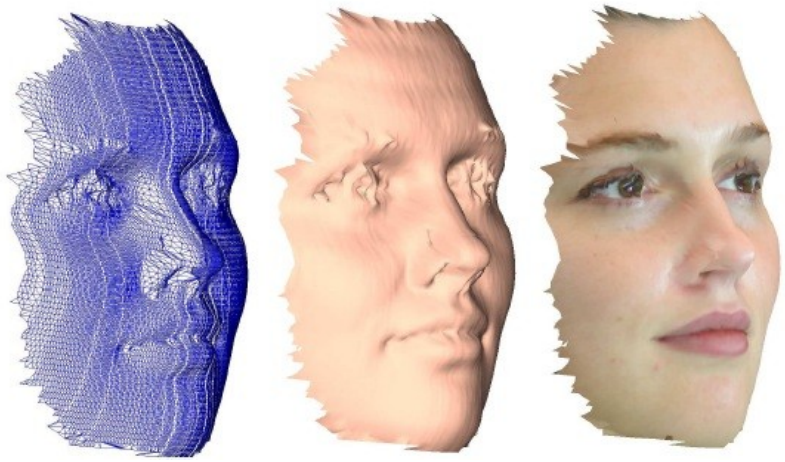


Fig.14 Recreation of wireframe, surface and texture model of a face in [19]

In [20] two systems based on the same principle of infrared coded Light scanning were introduced. The advantage of using infrared Light is, that it isn't visible to the human eye so there are no blinding patterns discomforting the person during scanning. Also the systems are robust to ambient light and can be implemented with off-the-shelf components. The first system can be used for one-shot scans and uses a spatial coding method. 2 infrared wavelength were used so in sum 4 colors can be used: black, wavelength 1 (λ_1), wavelength 2 (λ_2), wavelength 1 + wavelength 2. To create the pattern two infrared sources and a dichromatic mirror were used to create the light beam and a customized interference slide was used to produce the final pattern, as seen in Fig.15.

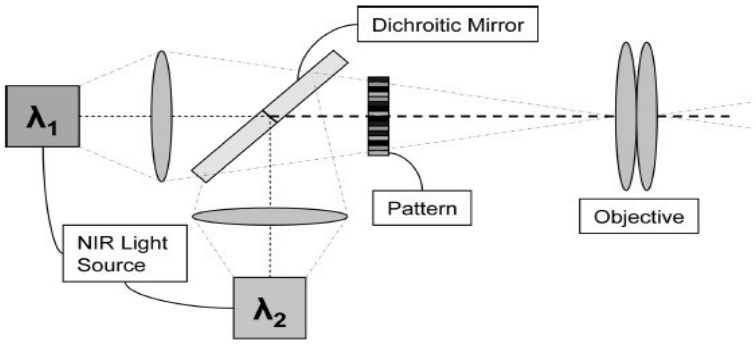


Fig.15 Experimental installation as used in [20] with interference slide

To capture the scene a video camera with a CCD sensor was used also capable of locating infrared light. To distinguish the different wavelenghts an additional filter was used. The filter consisted of stripes which were alternating transparent for λ_1 and λ_2 . That was necessary to know which wavelength was projected on which pixel. With this information the 3D coordinates of the points could be calculated.

The second system uses a dual-shot hybrid method of time multiplexing coding and spatial coding. The pattern used in this system gets split into two channels resulting in two slides projected consecutively as seen in Fig.16.

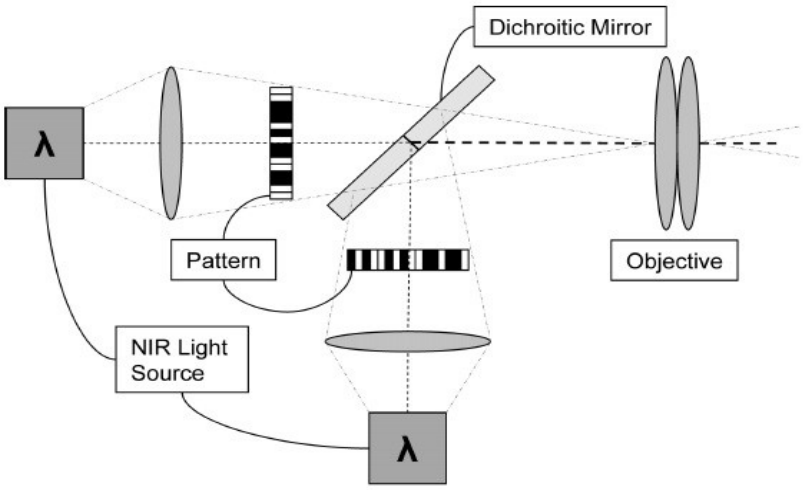


Fig.16 experimental installation used in [20] with chrome masks

Because these slides don't have to be penetrable by specific wavelengths but could be either penetrable or reflective, chrome masks could be used instead of an interference slide. The advantage of chrome masks is that they are cheaper, and easier to produce, and puts more emphasis on the goal

of mainly using off-the-shelf products. The result of the scan are two gray-scale images which get combined to a two channel coded light image. In this method no special filter has to be used and because only gray-scale images are used it allows the usage of a cheaper camera with lower resolution than the first method.

By comparing the two methods we see that the first technique makes one-shot scans possible which makes it also usable for moving objects. The second method however is a bit more accurate and allows the usage of cheaper components. An example for the result of both systems can be seen in Fig.17.

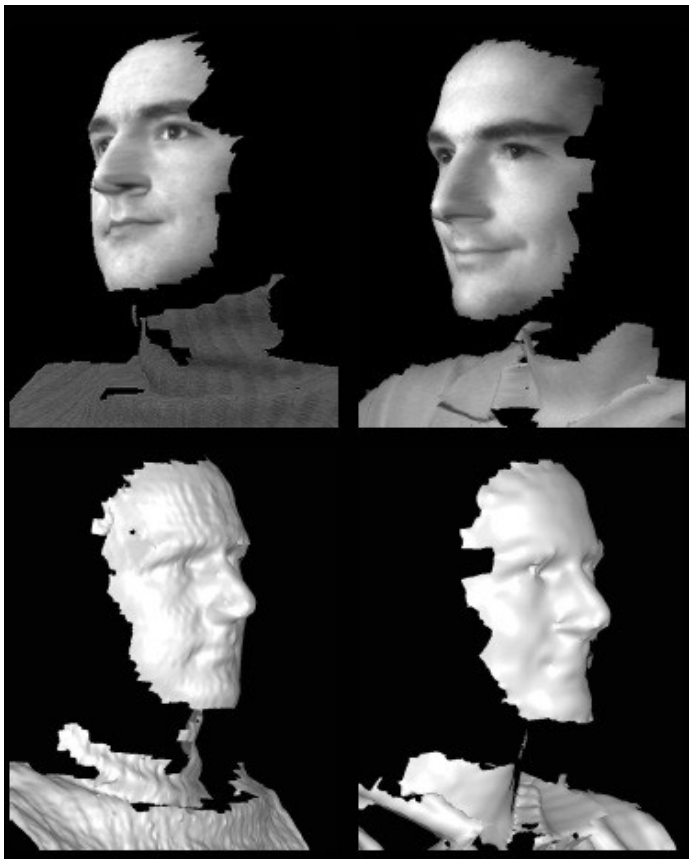


Fig.17 Left shows the one-shot approach and right the dual-shot of [20]

In [25] a one-shot method was introduced using the de Bruijn Pattern. The pattern consisted of 125 stripes with the limitation that the sequence of color transition from three neighbors are unique. First a gaussian filter was applied to filter out high frequencies, which blurred the image a bit, but the boundaries between the color lines were still visible. Then they searched for the color transitions. Based on the intensities of the RGB channels the likelihood of a match was determined

between the color transitions and the de Bruijn pattern. To improve the data small breaks, e.g. due to the shadow of the nose, were interpolated. The resulting curves where smoothed using a gauss filter to eliminate high frequencies. And finally the points of each stripe were calculated through optical triangulation. The last step was to apply the texture on the face model.

In [21] another structured light scanner was proposed by using a de Bruijn Pattern. To better distinguish the color stripes and to improve the robustness, black stripes were put in between them. Because the scanner is primarily for scanning faces which have an average size of about 0.3 m a pattern with 64 horizontal stripes was enough. The pattern was created using a de Bruijn sequence with a six element sub-sequence. Additionally a constraint was that two lines with the same color had to be at least separated by two lines of different colors. Then the first step was to find the region of interest by applying face detection. To decode the colors they didn't use the RGB color space but the HSV color space. Because the V component has the biggest value of the components in RGB space, it can be used to determine the center of the strip. To locate the stripe centers a gaussian filter with a large kernel was used. The advantage of using a filter are that small noises and disturbances get canceled and that the peaks of the stripe get focused in the middle. To find the stripe center the local maximum has to be found in the smoothed signal which can be done by looking for the point where the difference to the previous one is negative and the difference to the following one is positive. If by chance several points have the same value the point in the middle is taken as maximum. After finding the strip center an interesting step happens. Instead of looking at each point separately the stripe centers get connected with their neighbors to a line which can be seen in Fig.18. A maximum slope for the line could be arranged which by exceeding leads to a discontinuity of the line.

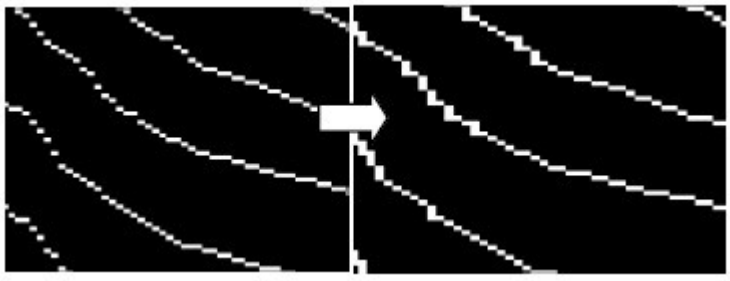


Fig.18 Line connecting as used in [21]

For the detection of the color of the line the H component of the HSV color space is used, which stands for the hue value. This happens by searching for save color values and setting the

undetermined region between them to white. Now we can distinguish three different cases.

If the undetermined region is surrounded by the same color the region gets the same color.

If the undetermined region is surrounded by different colors it gets deleted to avoid errors.

If the undetermined region is at the beginning or end of a line it gets the color from the one save color next to it. Finally the identified stripes get matched with the projected pattern by building a sub-sequence of six adjacent stripes and comparing them to the de Bruijn pattern. If any problems occur like, errors due to shadows, the information of the remaining points are used to find the match. At the end again the result is a model of the face in wireframe mode or textured with a image of the face. An example for the result of this method can be seen in Fig.19.

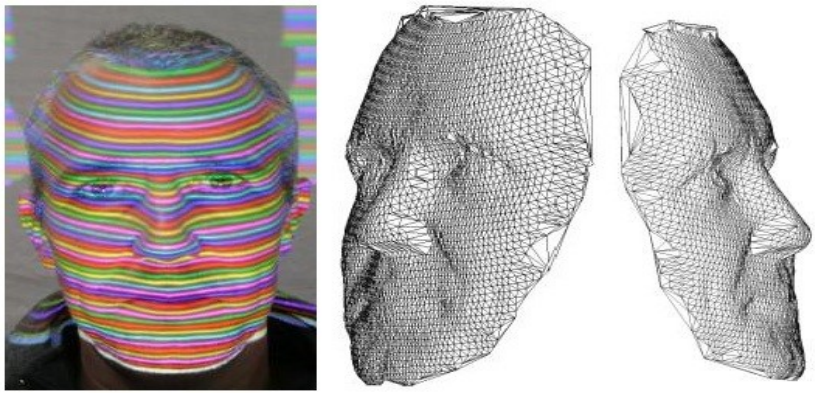


Fig.19 Wireframe model result from [21]

In [22] a method for face recognition was proposed using a morphable model of a human face. This approach doesn't use structured light but a database of different models to recreate any persons face. The basic model is created using 100 models of male heads and 100 models of female heads with their age between 18 and 45. To recreate a specific face a image of the persons face is needed. Then the following steps happen (Fig.20). First you have to manually set about seven specific feature points, for example eyes, nose or mouth. Following the program rotates the basic model of the head to fit the position and angle of the face in the image. Then the color and illumination get adjusted. Iterating over the face model first all the face features get optimized to fit the parameters. Finally the important parts like nose, mouth and eyes get optimized separately. Because this method was thought of for face recognition it was tested on two databases the CMU-PIE and FERET. Using many images of these databases the method achieved promising results of about 95% correct identifications. Faces with beards, glasses etc. were excluded from the test.

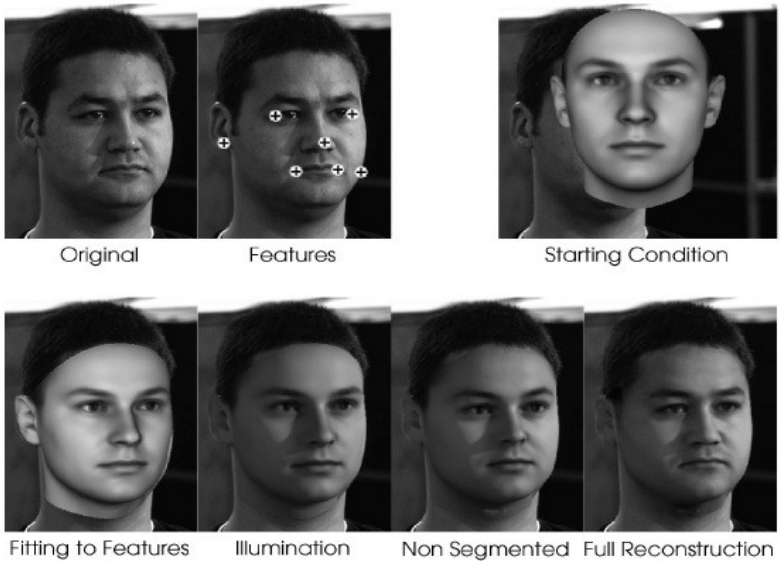


Fig.20 Steps to recreate a face from an image as described in [22]

The approach in [23] uses a combination of two techniques, Morphable Models and Support Vector Machines (SVM). Using an image or a frame of a video the face gets detected using SVM. If more than one possibilities get detected, the one with the highest confidence value is chosen. A square with the detected face gets cut out and resized to 200x200 pixels. Then the feature points of the face get detected and unlike the method described in [22] this method doesn't need any user input. Instead it uses a new algorithm based on the kernel ridge regression that can predict the position of the relevant points. By using a 12 x 12 array of regressors 144 estimates are computed and the most likely chosen as seen in Fig.21. The use of 144 regressors makes detection very robust.

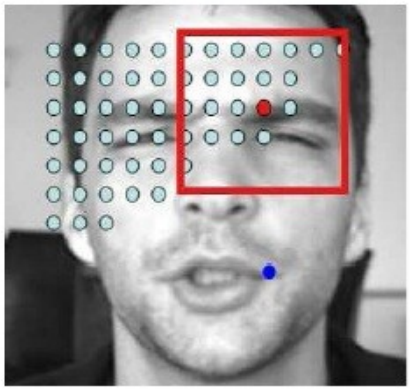


Fig.21 Algorithm to calculate feature points used in [23]

The feature points like the corners of the eyes and mouth and the nose are used to fit the 3D model. Especially the position of the nose tip is important as it also determines the rotation angle of the

head. By knowing the image positions of the feature point and the corresponding vertices the position of the face can be recreated by rotating, scaling and translating. Then using a database of 200 faces the principle of Morphable Models is used. Which means a statistically face using a database is generated. In [23] a little survey was done, letting 6 participants rate 48 created faces. The result can be seen in fig.22

participant	1	2	3	4	5	6	mean
very good	6	8	3	9	11	8	7.5
good	21	20	16	14	12	18	16.8
acceptable	13	14	14	11	13	14	13.2
bad	8	6	15	14	12	8	10.5

Fig.22 Rating of 48 scans by 6 participants [23]

In [24] an approach was made using a generic head model and pictures of a persons head from different perspectives. Three images were used, one showing the frontal view of the head, and one each for the side view of the head. It was important that the facial features like eyes and mouth were visible on all adjacent images. The rest of the images except of the face was colored black. First the generic model was stretched to fit the contours of the images. Face organs like ears or nose get neglected in this step. The next step was to accurately handle the face organs, like nose or mouth. Because of the complexity a number of simplified generic organs were used. For the size and placement some feature points were set manually describing for example the eye - or mouth corners. The last step was to texture map the constructed model. The texture was created by using all three images. An example for this technique can be seen in Fig.23



Fig.23 example of finished model in [24]

In [36] a low cost scanner was implemented using a time multiplexing method with color patterns. The hardware used were a video projector and a digital camera. The patterns in this method used

three colors, blue, red and green. In each step a region was divided in two sub-region colored blue and red with a thin green stripe in between them, with a width of one pixel (Fig.24). This has the advantage that you don't have to search for the discontinuity between two regions of different colors but for the green lines.

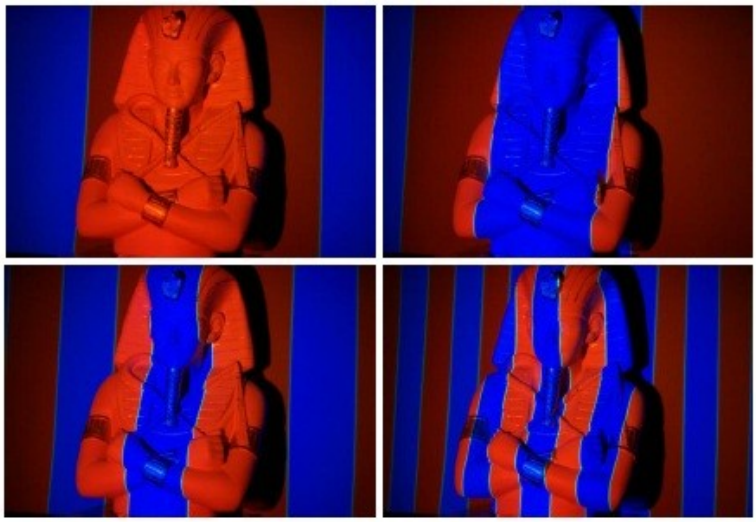


Fig.24 Pattern used in [36]

In the decoding phase they looked for these green stripes in the images and computed the medial axis from each. If any pixels are dark due to being in a shadow, they get extracted and not further considered. Then with medial axis the space coordinates of the points could be calculated with triangulation.

The scanner was intended for Cultural Heritage artifacts and delivers good quality as can be seen at the scan of Minerva of Arezzo in Fig.25



Fig.25 Scan of the statue of Minerva of Arezzo [36]

In [34] they developed a very fast phase shifting system capable of scanning the human face while in motion, like speaking. Instead of the usual three step phase-shifting algorithm they introduced a modified 2+1 step phase-shifting algorithm. This algorithm uses only two fringe images with a 90° phase shift and a third computer generated flat image as seen in Fig.26.

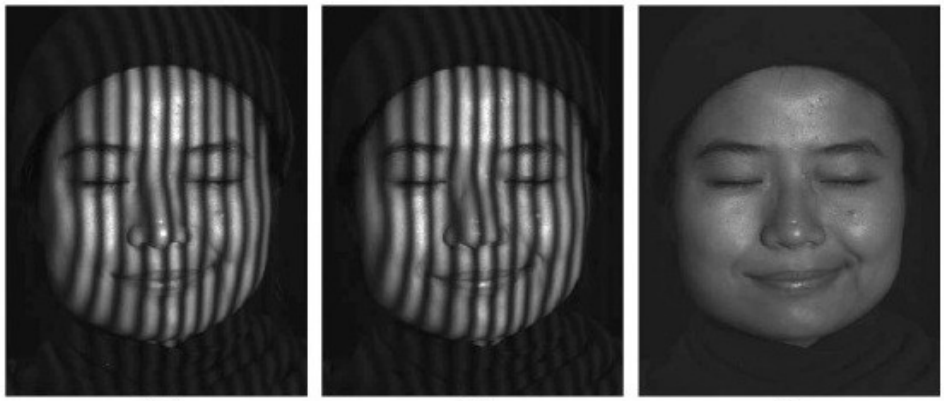


Fig.26 a is the first fringe image, b is the second fringe image, c is the flat image used in [34]

The advantages of this method over the three step phase-shifting algorithm are that it has faster

motion measurement. Because it only uses two fringe images and the flat image has the advantage of being less sensitive to motion than the fringe images. It has a faster procession speed, more than two times faster than the three step phase-shifting algorithm. And the texture can be used directly from the flat image so no additionally image is needed. The system was tested with a human face speaking even faster than normal speed and as seen in Fig.27 has led to satisfying results.

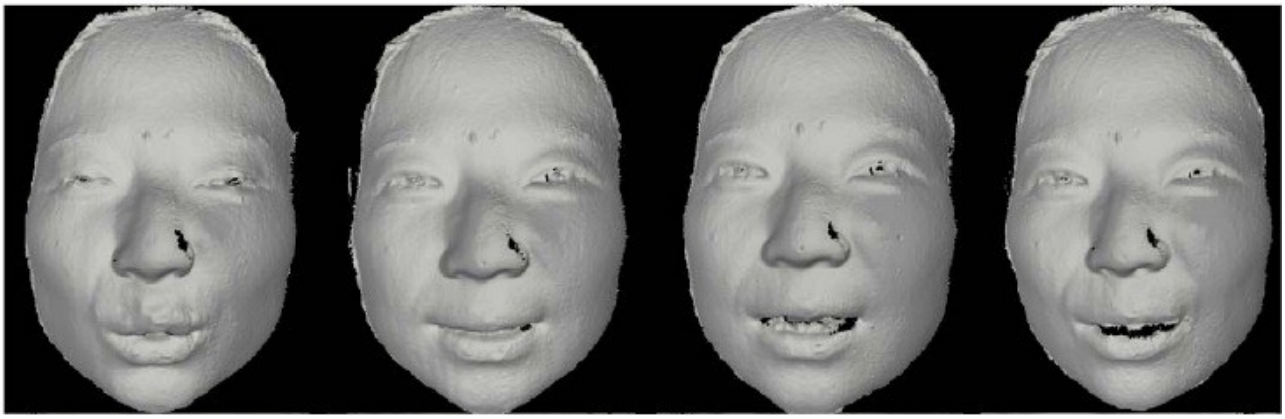


Fig.27 some examples of human face scan while speaking from [34]

In [35] a high-resolution scanner for face recognition was implemented. To encode the points eight binary patterns were used. A full-dark picture I_k and a full-illumination picture I_H were also taken for later, to compensate for ambience light. Because the illumination value $I_L(x,y)$ for a dark pixel, which would normally has the value 0, can differ from it due to ambient light. So the images were normalized by using following equation:

$$J_k(x,y) = \frac{I_k(x,y) - I_L(x,y)}{I_H(x,y) - I_L(x,y)} \tag{2}$$

Normally the images only should have binary values, 1 and 0. But practically there were some variations so it was defined that:

$$1: \; J_k(x,y) > 0.5$$

$$0: \; J_k(x,y) < 0.5$$

A Gaussian Filter applied before binarization also improved the decoding. Then the images were looked at in sub-stripe resolution. An example of the method described in [35] can be seen in Fig.28.

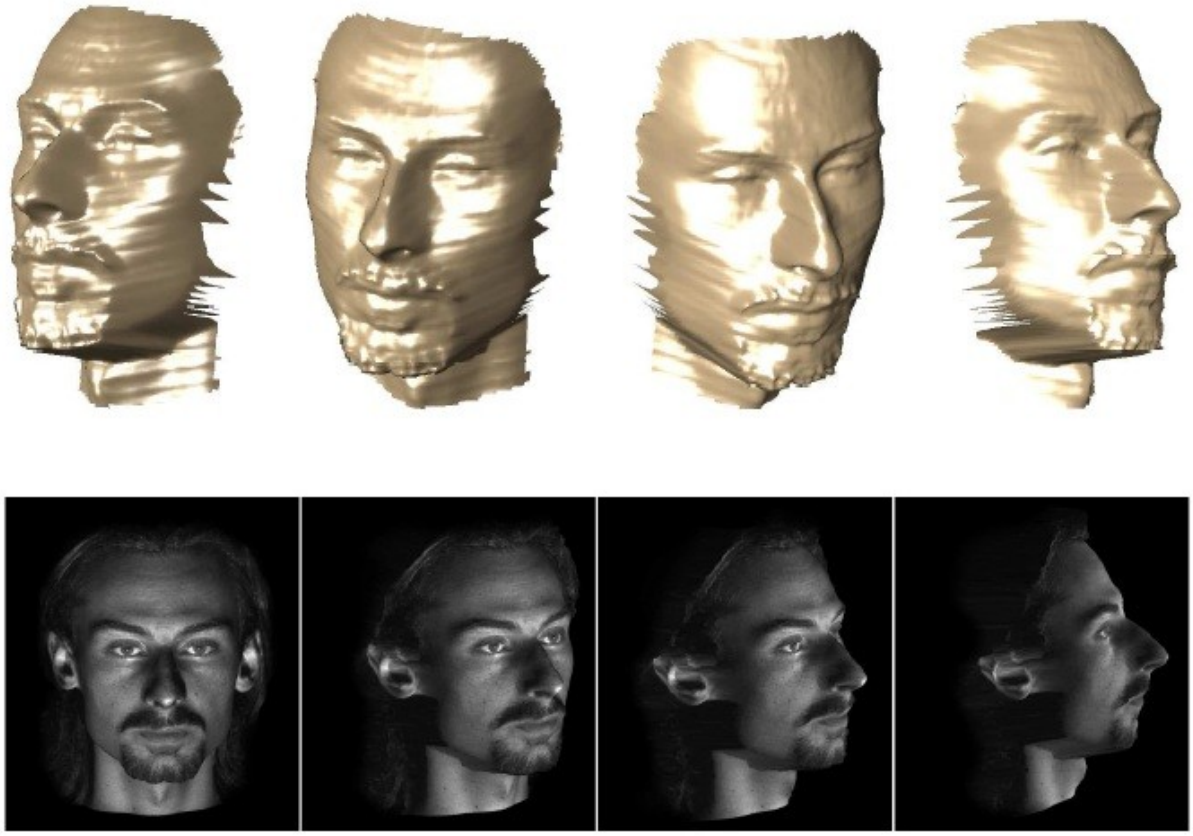


Fig.28 Result for face scan in [35]

4. Implementation

The first step was to calibrate the camera. For our future calculations , it was important to know the position and angle of the camera as exact as possible. One option of course would be possible to measure this by hand, but it is difficult to measure the camera position accurately in space and to get more reliable data we decided on a different approach based on the Pin-hole camera model principle as used in [1] and [2]. In [1] there is a way described to calibrate a camera using the pin-hole camera model.

4.1 The Pin-hole camera model:

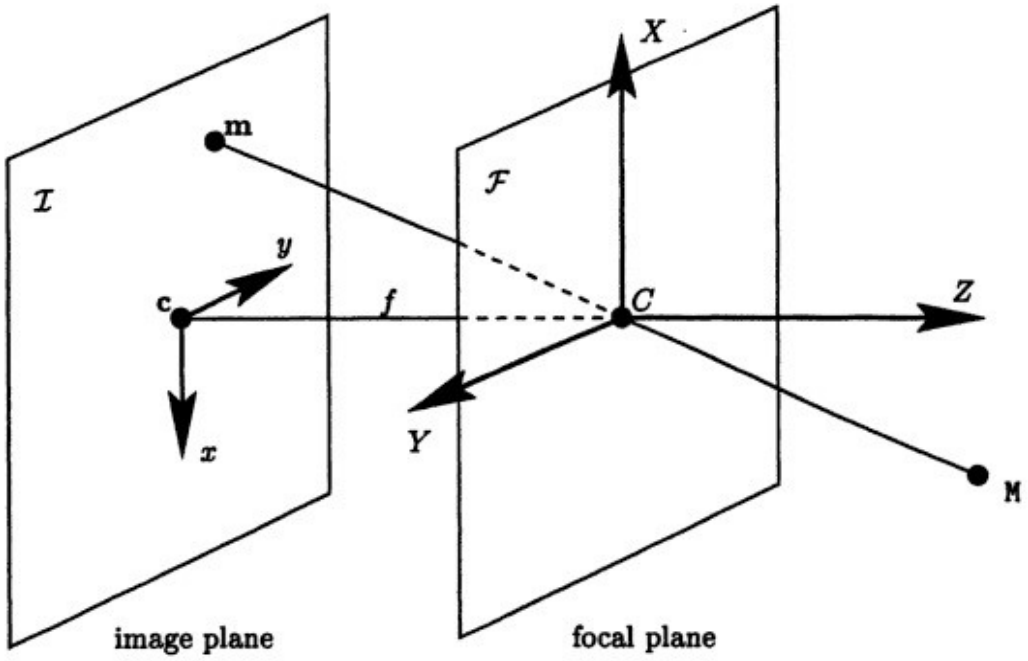


Fig.29 Pinhole-camera model [2]

The Pinhole-camera model describes the basic principle of a camera. The installation consists of an enclosed case with a small hole (pinhole) in it. Through the hole, light from a scene outside gets focused and projected on the back of the case, but inverted. Now with the use of a light sensitive Material a photo can be taken. So the Pinhole-camera can take a scene from a 3D world and map it to a 2d image. In Fig.29 we see a pinhole camera model.
We can recognize two planes:

1. The focal plane F is the plane where the light is focused.
2. The image plane I is the plane where the scene gets mapped upon.

The focal plane F has a fixed distance f to the image plane I. On the plane F we can find the pinhole C. It's important to keep in mind that the three points, the point of the object, the pinhole and the point on the image plane, build a straight line. This projection of a point in 3D-space to a plane in 2D-space is called perspective projection.

4.2 Camera Calibration

After this brief explanation on how the pinhole camera works we can extract some important points we need for our further calculations. The important parts of the pinhole camera model are the image plane I, the focal plane F which is parallel to I and the pinhole point C on F. The distance between I and F is fixed and called the focal length f.

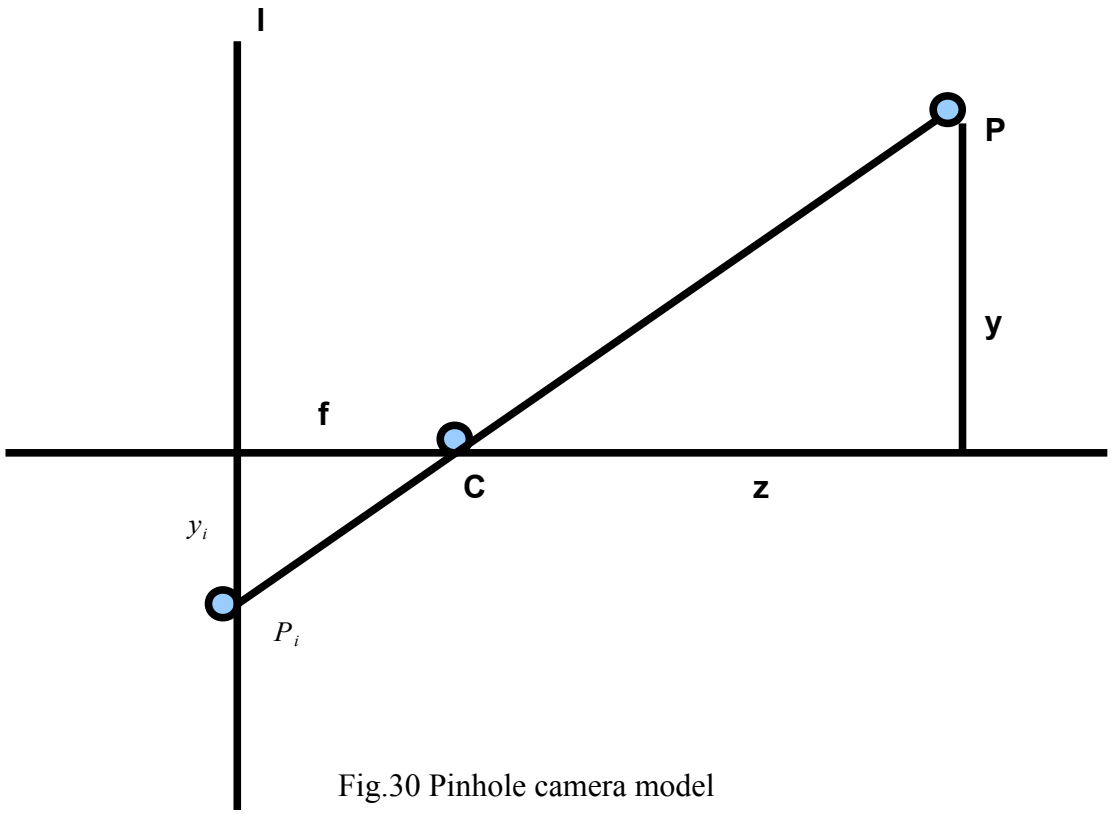


Fig.30 Pinhole camera model

We define the coordinate system of the image plane so that it's origin is at the point c which corresponds to the pinhole point C on the focal plane. This gives us the coordinate system for the

image plane, which is (c,x,y). For the origin of the three-dimensional space we use the pinhole point C. So the coordinate system is (C,X,Y,Z) which is the camera coordinate system. As noticeable in Fig.29 the X- and Y-axes are parallel to the image x- and y-axes only in the opposite direction.
So we can set the two coordinate systems easily in relation by writing x/X = y/Y = f/Z.
As mentioned before we used the way described in [1] to calibrate the camera. The essence of it will be explained in the following.
In Fig.30 we see another model of the pinhole camera, but from the side perspective. As stated before, using the pinhole camera model, a point $P = (x, y, z)$ in space which gets mapped on the image plane can be connected with a straight line to the corresponding point on the image plane. If we call the Point of the Intersection P_i with coordinates (x_i, y_i, f) we can see set the coordinates of the two points into relation:

$$\frac{y}{z} = \frac{y_i}{f} \quad \text{and} \quad \frac{x}{z} = \frac{x_i}{f}$$

through conversion we can write the coordinates of our point P_i as

$$y_i = \frac{y}{z} f \quad \text{and} \quad x_i = \frac{x}{z} f$$

to get resolution independent coordinates (u,v) we divide the coordinates from P_i through the screen width c and the screen height d .

$$y_i = \frac{y}{z} \frac{f}{c} \quad \text{and} \quad x_i = \frac{x}{z} \frac{f}{d}$$

$$(u, v) = (\frac{y}{z} \frac{f}{c}, \frac{x}{z} \frac{f}{d}) \tag{3}$$

now by multiplying both sides with z we get

$$(zu, zv) = (y\frac{f}{c}, x\frac{f}{d}) \tag{4}$$

Until now we only used the 3D points in the camera coordinate system. But they normally can be expressed in any 3D coordinate system which is also called world coordinate system or World space. At the beginning the points normally are in World space, which has it's origin at a specific point. But for our further calculation we want the points to be in camera space. When the points are in camera space their coordinates are relative to the camera origin. To accomplish this we have to define the View Matrix V . The View Matrix V is the inverse matrix of the camera World matrix W , which transforms the camera coordinates from the local coordinate system to the world space. The camera World matrix W usually consists of a Translation T and a Rotation R which are also called the extrinsic parameters of the camera. T consists of a translation to a

Point $B\ (b1, b2, b3)$ and R consists of 3 vectors which represent the rotation around the x, y and z axis.

$$T = \begin{pmatrix} 1 & 0 & 0 & b1 \\ 0 & 1 & 0 & b2 \\ 0 & 0 & 1 & b3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} r11 & r12 & r13 & 0 \\ r21 & r22 & r23 & 0 \\ r31 & r32 & r33 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

By switching to 4 Dimensions we can multiplicate R and T to get the camera World matrix W .

$$W = T * R = \begin{pmatrix} r11 & r12 & r13 & b1 \\ r21 & r22 & r23 & b2 \\ r31 & r32 & r33 & b3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Now we only have to invert the Matrix W to get the View Matrix V

$$V = W^{-1} = \begin{pmatrix} r11 & r21 & r31 & -Br.1 \\ r21 & r22 & r23 & -Br.2 \\ r31 & r32 & r33 & -Br.3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

So a point X in world space can be transformed to a point X_c in camera space by multiplying with V .

$$X_c = V X$$

Next we define the perspective projection Matrix P which projects a point X_c in camera space to the image plane.

$$P = \begin{pmatrix} \frac{f}{c} & 0 & 0 & 0 \\ 0 & \frac{f}{d} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{7}$$

By multiplying the point X_c with the projection Matrix P we get its Screen coordinates (u, v, 1)

$$(u, v, 1) = P X_c \text{ or } (u, v, 1) = P V X$$

due to our earlier multiplication with z we can also write

$$(zu, zv, z) = PX_c$$

Now we substitute $P * V$ with the matrix A

$$P*V = \begin{pmatrix} fr11 & fr21 & fr31 & f(-Br.1) \\ fr21 & fr22 & fr23 & f(-Br.2) \\ r31 & r32 & r33 & -Br.3 \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \end{pmatrix} = A \quad (8)$$

When we have the matrix A we want to obtain a matrix C by multiplicating A with $\frac{1}{a34}$

$$C = \frac{1}{a34} A = \begin{pmatrix} \frac{a11}{a34} & \frac{a12}{a34} & \frac{a13}{a34} & \frac{a14}{a34} \\ \frac{a21}{a34} & \frac{a22}{a34} & \frac{a23}{a34} & \frac{a24}{a34} \\ \frac{a31}{a34} & \frac{a32}{a34} & \frac{a33}{a34} & 1 \end{pmatrix} = \begin{pmatrix} c11 & c12 & c13 & c14 \\ c21 & c22 & c23 & c24 \\ c31 & c32 & c33 & 1 \end{pmatrix} \quad (9)$$

$$(z\frac{u}{a34}, z\frac{v}{a34}, \frac{z}{a34}) = C$$

now we have three equations

$$\begin{aligned} z\frac{u}{a34} &= xc11 + yc12 + zc13 + c14 \\ z\frac{v}{a34} &= xc21 + yc22 + zc23 + c24 \\ \frac{z}{a34} &= xc31 + yc32 + zc33 + 1 \end{aligned} \quad (10)$$

We can substitute $\frac{z}{a34}$ into the other two equations and get two equations with 11 unknown variables

$$\begin{aligned} uxc31 + uyc32 + uzc33 + u &= xc11 + yc12 + zc13 + c14 \\ vxc31 + vyc32 + vzc33 + v &= xc21 + yc22 + zc23 + c24 \end{aligned} \quad (11)$$

This leaves us now with two equations with 11 unknown variables. To solve this we need at least 11 equations. By measuring a point P in space we get its space coordinates (x, y, z) and its corresponding screen coordinates (u, v) . So to solve the 11 equations we get in the end we need at least six points with at least two different z-values so they have to be in different Z-levels. Then we can calculate the position and rotation of the camera by retracing the steps above. By trying with six points it became clear that for an exact calculation that wasn't enough because by checking and measuring by hand we got errors up to 5 cm. So we decided to use 12 points which got us satisfying

results. The calculation was also more exact if the camera was positioned a bit to the side so that it didn't faced frontal to the whiteboard.
The same procedure had to be done with the projector to get its exact position values also described in [1].
And finally we needed the height and width of the whiteboard we used for scanning which was easy to obtain by measuring or looking it up on the manufacture's website.

The camera calibration in our program works like this. It opens a window that shows you a live-picture of the web-cam. By clicking on a pixel of the image the coordinates get shown in the upper left corner and also in the shell command window. If you have a point in space you know the coordinates of, first click on the camera screen so that the pixel coordinates are shown. Then by pressing the space bar the command shell window gets active where you can input the 3D-space coordinates of the point as seen in Fig.31. My method was, so that I don't always have to measure the points in space again, to create a raster with with 17 columns and 8 rows. Then I measured them and saved the data to a table. To measure the points in another Z-level you can use any object with known length, ideally with a pointy end you can use as the point to measure. By putting that object onto the whiteboard you can measure it. Things that worked out fine, were a set square or a chopstick.

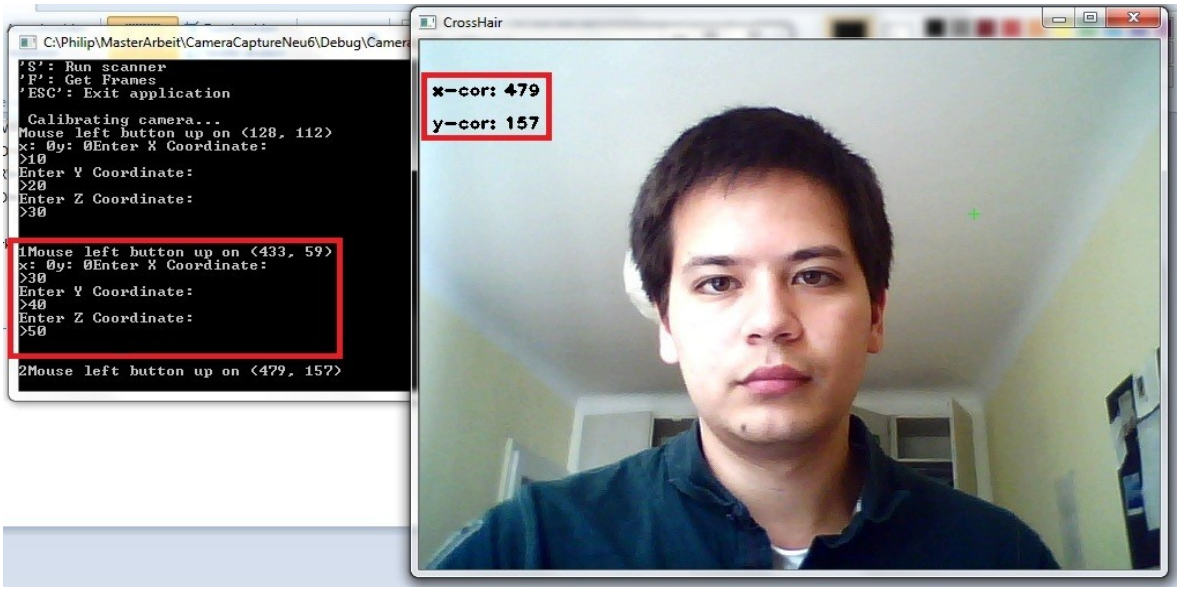


Fig.31 The camera screen with the pixel coordinates and the shell window to input the space coordinates.

After the input of the last point the window closes automatically and calculates the camera position in centimeter and angle in degrees as seen in Fig.32. The camera matrix which is more important for our future calculation gets saved internally.

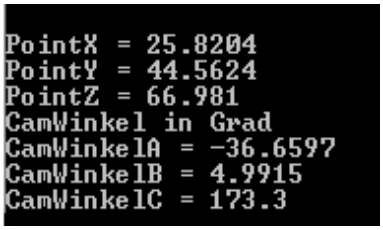


Fig.32 output after Camera calibration.

5. Scan Process

The necessary hardware for the scan were a interactive Whiteboard and a standard HD-webcam. The software used are C++ with the open source library OpenCV. The Scan Process is using the very basic principle of structured light. A plane of light, which is projected by the projector of the interactive whiteboard, gets moved slowly from left to right. Therefore the head of the person standing in front of the whiteboard gets illuminated with a thin line of light. This gets captured with a HD-webcam that takes a picture every time the line moves a step.
Important for a good scan is to darken the room as good as possible so no other light can interfere with the projector. And the camera should be positioned so, that the head of the person is in the center and fills out the frame.

The Scan process can be divided into four steps:

- Obtaining the pictures
- Processing the pictures
- calculating the points
- deleting of impossible data

5.1 Obtaining the pictures

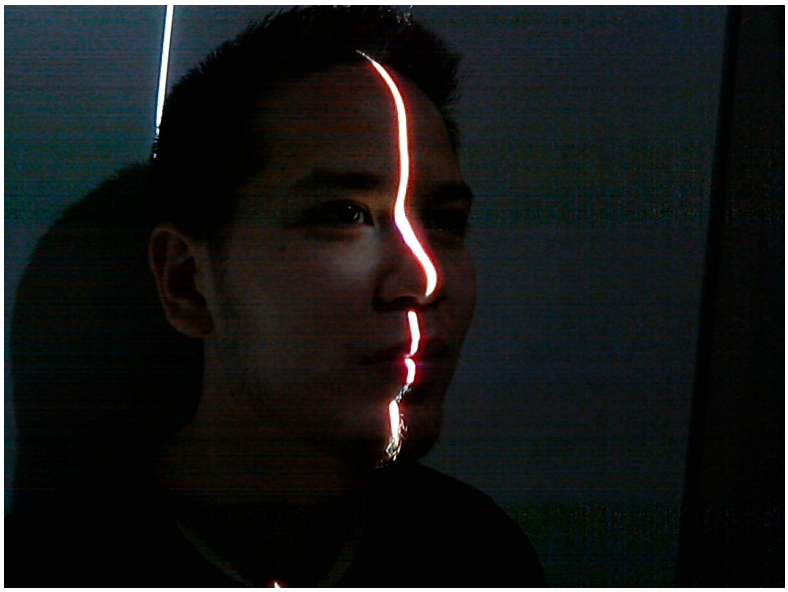


Fig.33 Picture right after scan

The first step was to obtain the pictures of our scan. Using OpenCV we created a black full screen window that gets projected on the whiteboard. Beginning on the left side we build a white line which slowly moves to the right side with one step being one pixel of the whiteboard. The line's width should be as small as possible to get good results so we chose the width to be one pixel. Looking at Fig.33 we see that although we only chose the width to be one pixel the line of light seems relatively thick, which is of course was due to the big size of the whiteboard. During the process a normal HD-webcam, which was positioned slightly on the side of the head, made a photo every time the line moved one step.
Some things to consider are that it was important to darken the room as good as possible, so no light other than the white line of the beamer illuminated the face. The head of the person should be in the center and should fill out the frame as much as possible. And the person should try not to move the head during the scan process.

5.2 Processing the pictures



Fig.34 Grayscale Picture

The next step is to take the images we obtained earlier and process them to only get the relevant data we need later on. First we change the image into a gray-scale image which can be easily done using an OpenCV function. As we can see in Fig.34 the white line is pretty good to distinguish from the rest of the picture. To extract the white line we orientated us on the brightness. To choose the limit is a bit tricky as it shouldn't be too low to not get any irrelevant data, but also it shouldn't be

too high so we don't lose too much relevant data. After some testing it seemed that from a scale 0 to 255 where 255 is the brightest value, the value 200 has shown to be a good limit that gives satisfying results. So after we set the limit we only want to extract the points with the specific brightness. This is done by comparing the brightness value of each pixel and filter out the relevant pixels. These pixels get copied to a new image and colored white. The rest of the new image is black so we get a black and white image as a result. A problem is, as mentioned before, the thickness of the light plane that gets projected. This happens due to the largeness of the whiteboard. The white line we use as a scan-beam is bigger than one pixel in our camera image. And with a person standing in front of the whiteboard the line gets even bigger in relation to the distance from the head to the whiteboard. But for our black and white picture we only want a thin line, best would be one pixel. Because for our future calculations we only want one white line with the thickness of one pixel. Our approach to accomplish this was to use another useful function from OpenCV called cvCanny which gets us only the borderlines of our image (Fig.35).



Fig.35 Edges of light after use of cvCanny function

The advantage of using cvCanny is that it returns the edges of a picture. This helps to get rid of small noise and leaves us with two white lines we can choose from, the left or the right edges. For example only using the left edges gets us a single white line which we wanted as a result. It doesn't really matter which side we choose, except if there was an interference or unwanted light on either the right or the left side. Than it would be appropriate to use the other side to cancel it out. To get only the right side for example, we just look at each horizontal line of the image and look for the

white pixel which has no white pixel as right neighbor any more. By doing this for each line we get all the white pixels on the right side. The finished result of the scan can be seen in Fig.36



Fig.36 Finished processed image

5.3 Calculating the Points

Now that we have our black and white image with the distorted line of the scanned face, the next challenge was to get the 3D coordinates from the image pixels. In Fig.37 we can see a crude draft of the process. At the beginning we have the Projector at point P with coordinates (x_P, y_P, z_P), which we calculated earlier. Then we have a Point S with screen coordinates (u_S, v_S), which we can extract out of the black and white image by filtering the brightness. And we have the Camera with the camera point C, which we also calculated earlier. Important is also that we know the Height H of the Whiteboard by measuring before, and the Width W from the left side of the Whiteboard to the Light beam L. W we can calculate because we know the width of the whole Whiteboard W_W and we know the x position of the Lightbeam L. W_W we also know due to measuring and L we can calculate. Because the camera takes a picture every time the Light beam moves a step, in our case a step equals 1 pixel, and the beamer has a resolution of 1024 x 768. So we get 1024 pictures. Now we just have to divide the width of the whiteboard W_W through the resolution and multiply it with the frame count f.

$$W = \frac{W_W}{1024} f \tag{12}$$

By knowing the three Points W, H and P we can create a Plane E in space. Next we multiply the point S with the inverse projection matrix P_i to get the points in camera space. We

can define P_i as

$$P_i = \begin{pmatrix} c & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & f \\ 0 & 0 & 1 \end{pmatrix} \tag{13}$$

Because we don't know the focal length f for our camera we just set it 1. This doesn't change the final result. Then by multiplying S with P_i we get S in camera coordinate system and to get it in world coordinate System we have to multiply it with the camera world matrix C_W which is the inverse of the View Matrix V. Then we get S_W in world space.

$$S_W = C_W P_i S \tag{14}$$

Now that we have S_W we can create the straight line g by calculating the Vector C - S_W. Finally by intersecting g with W we get the correct coordinates of S. This has to be done for every white pixel in the picture and for every picture. So finally we have the space coordinates from each point of each picture, which gets saved in form of text files for each image separately.

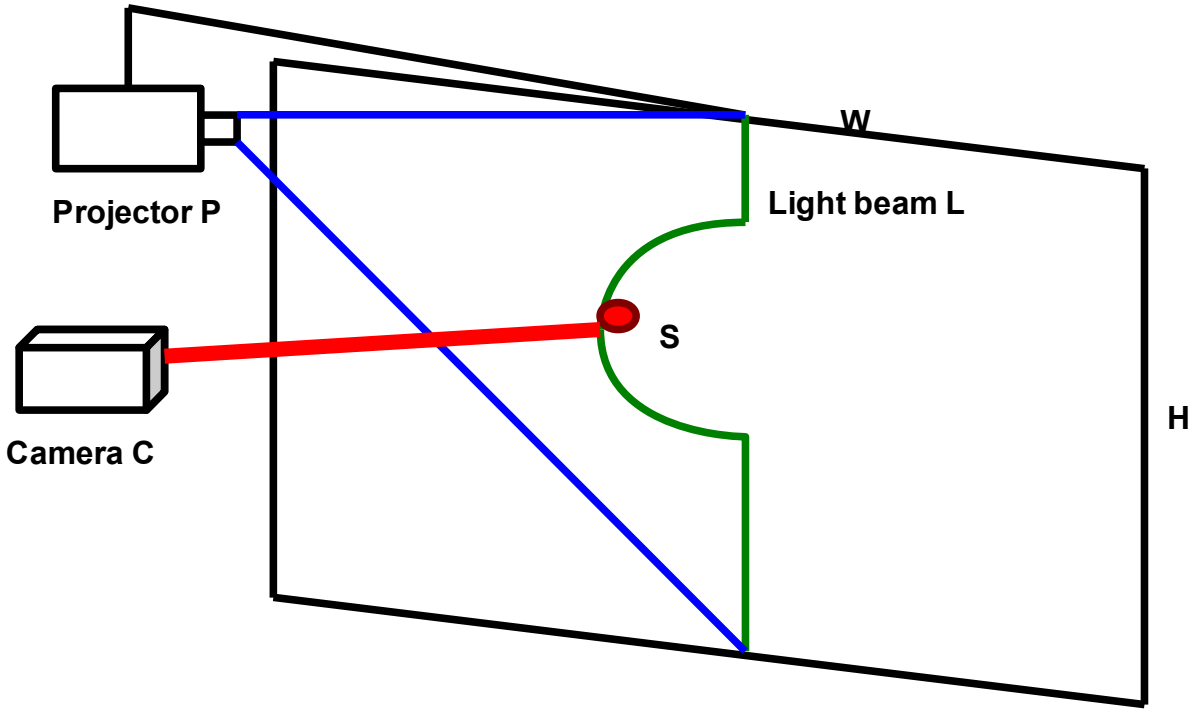


Fig.37 Draft of Point Calculation

5.4 Deleting of impossible data

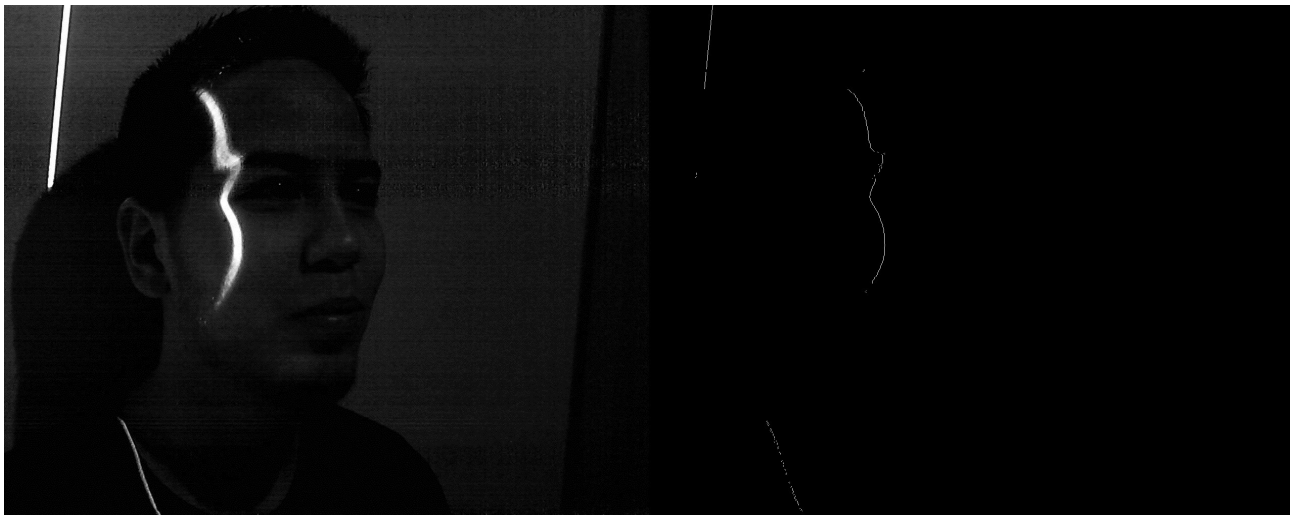


Fig.38 Picture with unnecessary data at the shoulder

As we can see in Fig.38 there can be a lot of unwanted data in an image. If the camera isn't exactly positioned to just capture the head and nothing else as in Fig.38 there is a high probability to have some unnecessary data in your capture. Even with the head being positioned perfectly there is always the chance of some incorrect data To counter this we filtered the data accordingly to delete impossible data. To figure out which data to delete we imposed some rules on the points we scanned.

First we searched for the maximum in the Z-plane which should be the nose tip. To exclude unwanted spikes which could have happened, for example due to protruding hair, we only looked in the second third of the image because this should be the place the nose usually is located. To raise the probability to find the nose we averaged the points with the same x-coordinates. Because if you draw a vertical line on the face you can see that most of the points have a high z-coordinate if they are on the same line where the nose tip is. So if the z-coordinate is the maximum for the face but all the other points with the same x-coordinate have a relatively low z-value we can eliminate it as the nose tip and mark them as a spike.

After we found the Nose tip with the coordinates (X-med, Y-med, Z-max) we know about where the center of the face is. Although the size of faces varies very strongly we can put some average limits on it's size. Usually the height of the head is 30 cm and the width is a bit less. But in our implementation we also took the width for about 30 cm. The depth of the head wasn't relevant for

us, because we only want to model the face and not the entire head. The depth of the face usually isn't more than 10 cm.
So we chose the limit to be
for Z-plane: Z-max – 10.
for Y-plane: Y-med +-15.
for X-plane: X-med+-15

Furthermore the limit on the Y-plane gets smaller when the x-coordinates reaches the limits because of the slight rounding of the head.

6. Displaying the points

6.1 Meshes

before we can display the points in OGRE we have to look how they get displayed. Usually to convert a point-cloud into a 3D model you have to use a mesh. In [3] we have three simple types of meshes explained.

Vertex-vertex meshes

Vertex-vertex meshes are the simplest types of meshes. An Object is only represented by vertices where each vertex indexes its neighboring vertices. The advantage of this method is the small storage space and efficient morphing of shape.

Face-vertex meshes

Face-vertex meshes represent an object not only through vertices but also through a set of faces. The improvement to Vertex-Vertex-meshes is that they allow to look up the vertices of a face. Every face has to have exactly three vertices. An example of a cube with a vertex v and a face f can be seen in Fig.39

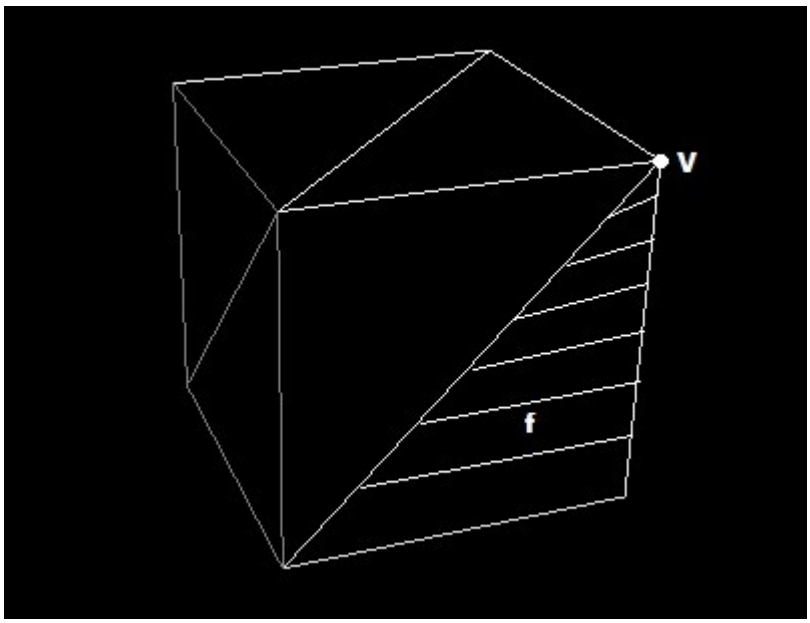


Fig.39 example of cube with vertex and face

For rendering usually the vertices not only have information on position but also on color/texture

and normal structure. The advantage is that if the shape of the mesh changes but not its geometry it can be dynamically updated without updating the face connectivity.

This is the most commonly used mesh representation, and also used in OGRE.

winged-edge meshes

another method worth mentioning was introduced by Baumgart 1975 [4]

the so-called winged edge meshes not only represent the vertices and the faces but also the edges of a mesh.

For any given edge, the number of outgoing edges may be arbitrary but usually to make things easier it's four edges. The nearest clockwise and counter-clockwise edges at each end. From there the mesh also gets it name because it looks like the wings of a butterfly. So the total data for an edge consists of 2 vertices (endpoints), 2 faces (on each side), and 4 edges (winged-edge).

It's advantage is the flexibility, for example split and merge operation can be done pretty fast. But one of it's setbacks is the large storage space required.

6.2 Building the vertex matrix

As we mentioned above in OGRE we could display the points as face-vertex mesh. To do this there is a helpful class *manual object* that allows us to transform the points into vertices and afterward into triangle surfaces. But before we can build the vertices we have to think about how we want to connect the points. Because we can't just hand the points we calculated over to OGRE and get a finished mesh back. We have to define which points get connected to triangles so that OGRE can build the mesh. The problem was that for each picture the points were saved in a text file, but due to the distortion of the face and the deleting of unnecessary data the number of points in the text files varied a lot.

Our solution was to build a $n * m$ matrix so that every column has the same amount of points. This way the points just have to be connected with their neighbors. The downside is that you have a lot of, sometimes unnecessary, points. Nevertheless we decided to follow this approach. To create the matrix we had to make sure that every column has the same amount of points. For this we took each text file which was chosen to be relevant and looked for the maximum y_{max} and minimum y_{min} value for the y-coordinate. Then we subtracted y_{min} from y_{max} and divided the result

through a user specified value y_{count} which is the size of the matrix column to get y_{step} .

$$y_{step} = \frac{(y_{max} - y_{min})}{y_{count}} \tag{15}$$

Then we just add y_{count} times y_{step} to y_{min} and get our values for the matrix column. Now every column / text file has the same amount of Points. The value for the x-coordinate is easy to find because in each text file the value for the x-coordinate are the same for all points. So we have the x and y values of our points but the z values are still missing. To get them we compare the y-values of the old Points with the new calculated y-values and if the y-value of the old point is the same as a new calculated y-value we use the z-value. If the y-value of the old point is located between two of the newly calculated y-values the z-value is mapped on the smaller one. For example if the previous y-value was 50.5 and now we have two y-values 50 and 51 the z-value belonging to 50.5 gets mapped on the new y-value 50.

6.3 Adding the missing data



Fig.40 Deformation around the side of the eye

As described above we calculate new Points so that every column in the matrix has the same size. Then the z-values get mapped to the fitting y-values. The problem is that now we have a lot more y-values than z-values depending on the chosen y_{count} . To compensate for that we have to interpolate the missing z-values. First we search the points for missing z-values. Then we take the z-

value before, z_1 , and after, z_2 , the gap and compare the two values. If z_1 is greater than z_2 we subtract $z_1 - z_2$ and divide through the amount of missing z-values which gives us z_{step} . Now we subtract z_{step} from z_1 and we get the z-value for next point where z is missing. Then we subtract z_{step} again and get the next z-value and that we do for every missing z-value between z_1 and z_2 . If z_2 is greater than z_1 the same happens with the difference that we subtract $z_2 - z_1$ and we add z_{step} to z_1 for every z-value missing.

A disadvantage of the above used method is that due to the m*n matrix the points on the side of the face share a very small space which leads the model to be a little be pinched together at the side. Unfortunately this can be seen sometimes at the eyes of the model which get slanted down a little as seen in Fig.40

7. Smoothing

Because of the slight variation of the point positions on y- and z-axes caused by the scanning device, the surface of the face-mesh wasn't as smooth as you expect from a face. Also there were some spikes we hoped to get rid of, so we decided to implement a smoothing algorithms to smooth the surface of the face.

There are already some smoothing algorithms out there which all have their advantages and disadvantages.

7.1 Laplacian algorithm

The idea of the laplacian algorithm is quite logical: the position of a point is replaced with the average position of all the directly connected points in the mesh. So if we have a vertex v with N adjacent vertices $(v_1, \ldots, v_n)$ then

$$v = \frac{1}{N} \sum_{n=1}^{N} v_n \tag{16}$$

a simple example for laplacian smoothing can be seen in Fig.41

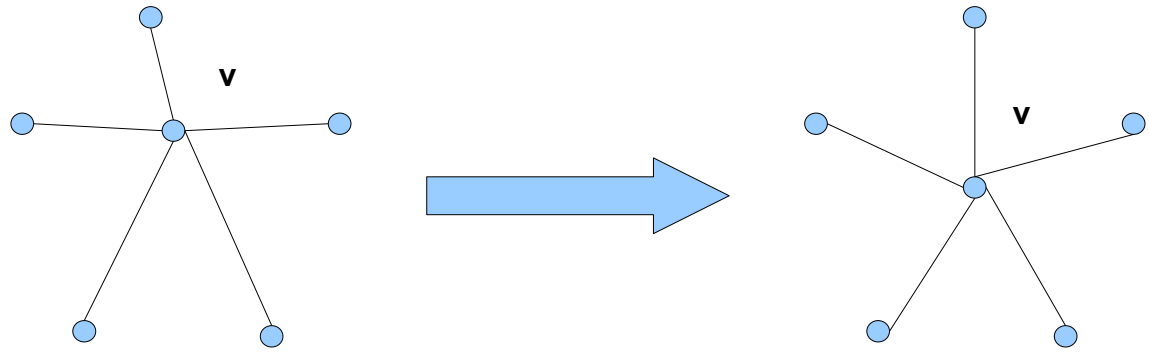


Fig.41 simple example of laplacian smoothing

laplacian smoothing can be done in two different ways:

simultaneously:

the position of all points are updated simultaneously, depending on the old position of the points.

Sequential:

the position of the points are updated immediately, already using the newly calculated values for the already moved points. Therefore the smoothing process is dependent on the order of the points.

The simultaneously version is more accurate but it also needs more storage space to save all the old and the new positions of the points.

7.2 weighted laplacian algorithm

another form of the laplacian algorithm is the weighted laplacian algorithm with w as weights that can be defined. Where v is the vertex to be updated, N is the count of adjacent vertices $(v_1, ..., v_n)$ and count of weights $(w_1, ..., w_n)$, the weighted laplacian can be defined:

$$v = \frac{1}{(\sum_{n}^{N} w_n)} \sum_{n=1}^{N} w_n(v_n - v)$$

for later usage we call it umbrella operator U which it got due to its form (Fig.42)

$$U(v) = \frac{1}{(\sum_{n}^{N} w_n)} \sum_{n=1}^{N} w_n(v_n - v) \tag{17}$$

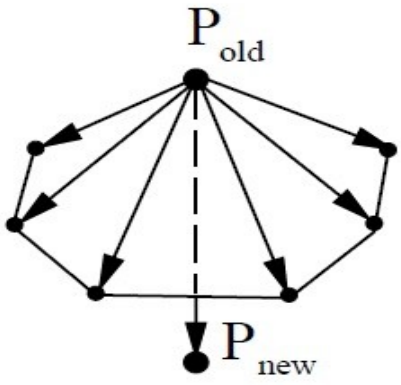


Fig.42 Umbrella operator [6]

The simplest weight would be to set $w=1$.

$$v = \frac{1}{N} \sum_{n=1}^{N} (v_n - v)$$

Another common method is to use the distance of the vertex v to it's neighbor v_n as weight

$$\text{e.g.} \quad w = \frac{1}{|(v-v_n)|}$$

Advantage of the laplacian algorithm is the linear time and storage consumption and it's simplicity. A noticeable disadvantage is the overall shrinking of the mesh as can be seen in Fig.43.

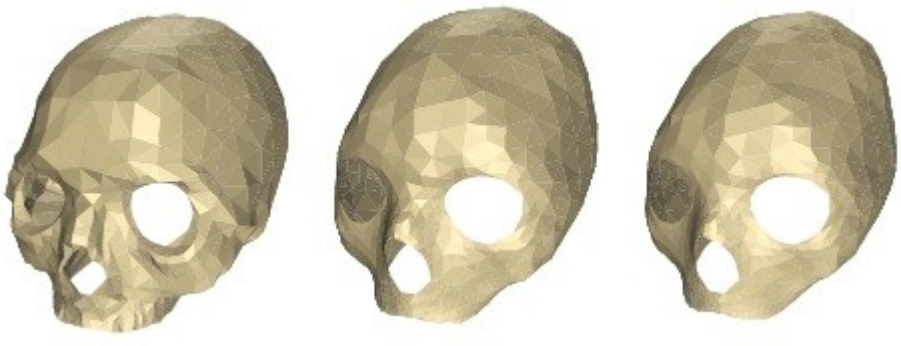


Fig.43 Example of mesh shrinking: left original, right after three iterations of laplacian smoothing. [5]

7.3 Taubin Smoothing

Taubin Smoothing addresses the problem of shrinking by introducing two scale factors a positive and a negative one. This has the advantage to suppress high frequencies of the umbrella operator, while on the other hand preserving its low frequencies. The taubin smoothing with v_{new} as the updated vertex, v_{old} as the not updated vertex, α as the negative scaling factor, β as the positive scaling factor and U as the umbrella operator can be written as:

$$v_{new} = (1 - \alpha U)(1 + \beta U) v_{old}$$

$$\rightarrow v_{new} = v_{old} - (\alpha - \beta) U v_{old} - \alpha \beta U^2 v_{old} \qquad (18)$$

7.4 Biplacian flow

if you use Taubin smoothing, with the negative and the positive scaling factors equal in value than its called the Biplacian flow.

$$v_{new} = v_{old} + \alpha U^2 v_{old} \tag{19}$$

7.5 Mean curvature flow

Mean curvature flow is a smoothing method using the mean curvature H and the unit vector n.

$$v_{new} = v_{old} + \alpha H(v_{old}) n(v_{old}) \tag{20}$$

In [26] a comparison between the above mentioned smoothing method was done. For comparison an original Mesh M and a distorted and afterward smoothed mesh M_1 were used. Then two error metrics were introduced E_v and E_n .

$$E_v = \sqrt{\frac{1}{(3\mathrm{A}(M_1))} \sum_{P \in M_1} A(P)\, dist(P, M)^2} \tag{21}$$

where P is a vertex on M_1 , $A(M_1)$ is the total are of M_1 , $A(P)$ is the sum of all triangles incident with P . E_v Is a resembles the metro mean distance introduced in [27] which is an approximation of the surface to surface distance.

$$E_n = \sqrt{\frac{1}{(AM_1)} \sum_{T_1 \in M_1} A(T_1) \left| n(T) - n(T_1)^2 \right|} \tag{22}$$

where T is a triangle of M , T_1 a triangle of M_1 , $n(T)$ the orientation unit normal of T , $n(T_1)$ unit normal of T_1 , and $A(T_1)$ the area of T_1 . E_n measures the deviations between the normals of the two meshes.

Some graphs in the paper showed the results of the tests, which of course varied a bit due to different model and noise. But most times did the mean curvature flow method show very good results.

8. Display the face in OGRE

After we have our m*n matrix of 3D Points and run a laplacian smoothing algorithm the data is ready to be displayed as a 3D mesh. As mentioned earlier we're going to use OGRE to display our face model.
OGRE stands for Object-Oriented Graphics Rendering Engine and is a open source 3D render engine. It is written in C++, supports Direct3D and OpenGL, and is available for all major systems, Linux, MacOS and Windows. OGRE is only a render engine but many features can be easily added like the open source physic engine Bullet to simulate physics. Because of this OGRE is also often used to produce games, there are even some commercial ones. A short overview can be found in [29] and more detailed info on OGRE can be found on its website [28].
In Fig.45 we can see an example of a scanned face displayed in OGRE.

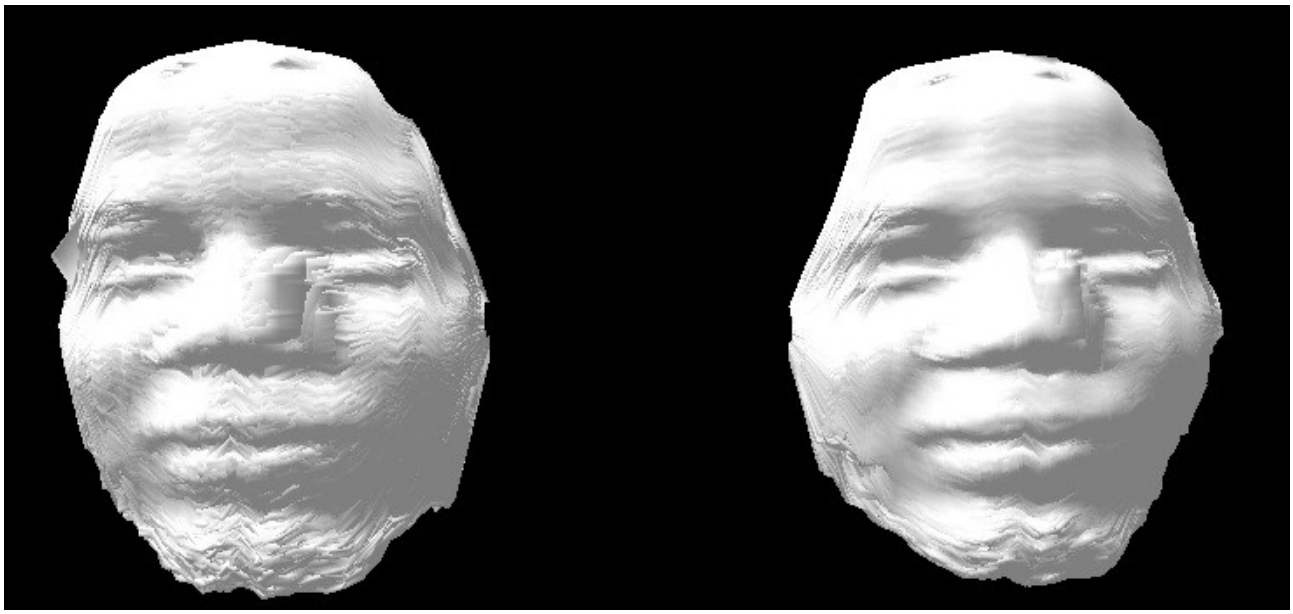


Fig.45 On the left side the normal model and on the right side the smoothed model displayed in OGRE

8.1 Mirror

Due to the angle of the camera or the prominent features of the face, like the nose, it could be that for a certain space of the face no data can be extracted. This can lead to errors as seen in Fig.46 where due to the nose, the light beam on the left side, next to the nose, can't be seen.

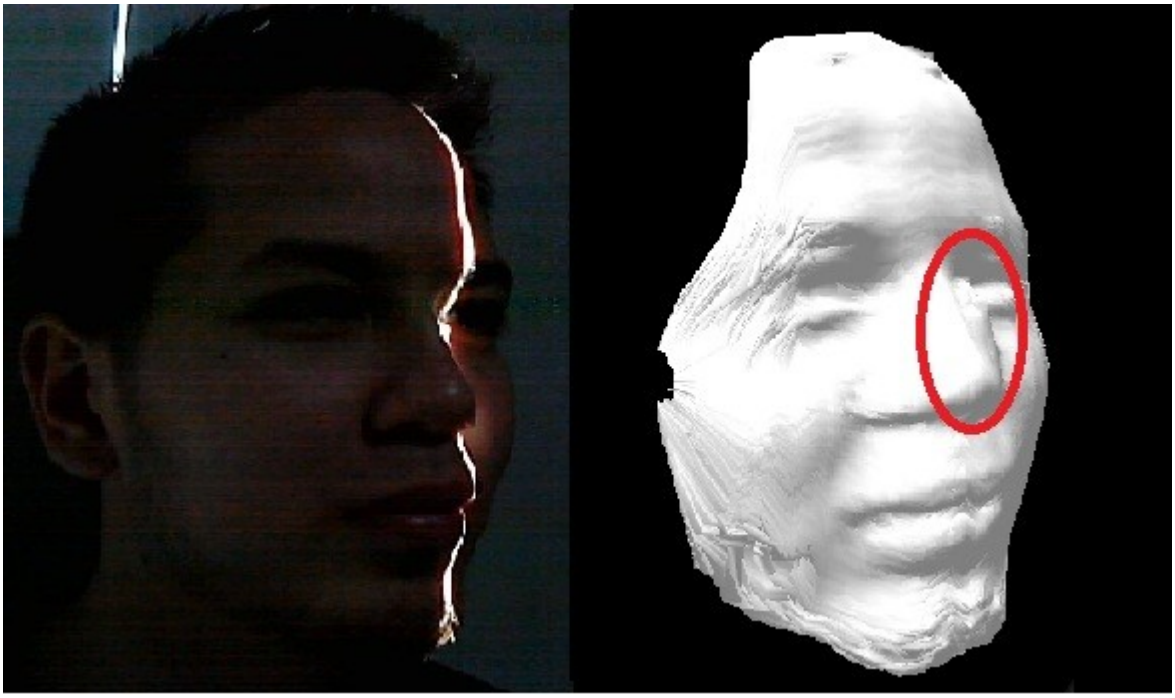


Fig.46 picture and model with error at the left side of the nose

To counter this effect we decided to implement a mirror function that mirrors the "good half" of the face. In this function you can decide whether you want to mirror the face on the left side or on the right side. Then as before the function searches for the nose tip through searching for the max z-value in the lower two thirds of the model. We chose the nose tip to be our mirror axis because it usually is located at the middle of the face. To mirror the points we only have to calculate the x-position of the point again, y- and z-value stay the same. If we call the x-coordinate of the mirror-axis x_{mir} the new x positions x_{new} and the old x-position x_{old} the calculation for mirroring right is:

$$x_{new} = 2x_{mir} - x_{old} \tag{23}$$

and for mirroring left:

$$x_{new} = x_{mir} - (x_{old} - x_{mir}) \tag{24}$$

So by mirroring the points we create a face where half of it is a reflection of the other half. This can get pretty good results as seen in Fig.47 but it can also deform the face if the nose tip isn't in the middle or another point is mistaken for the nose tip.

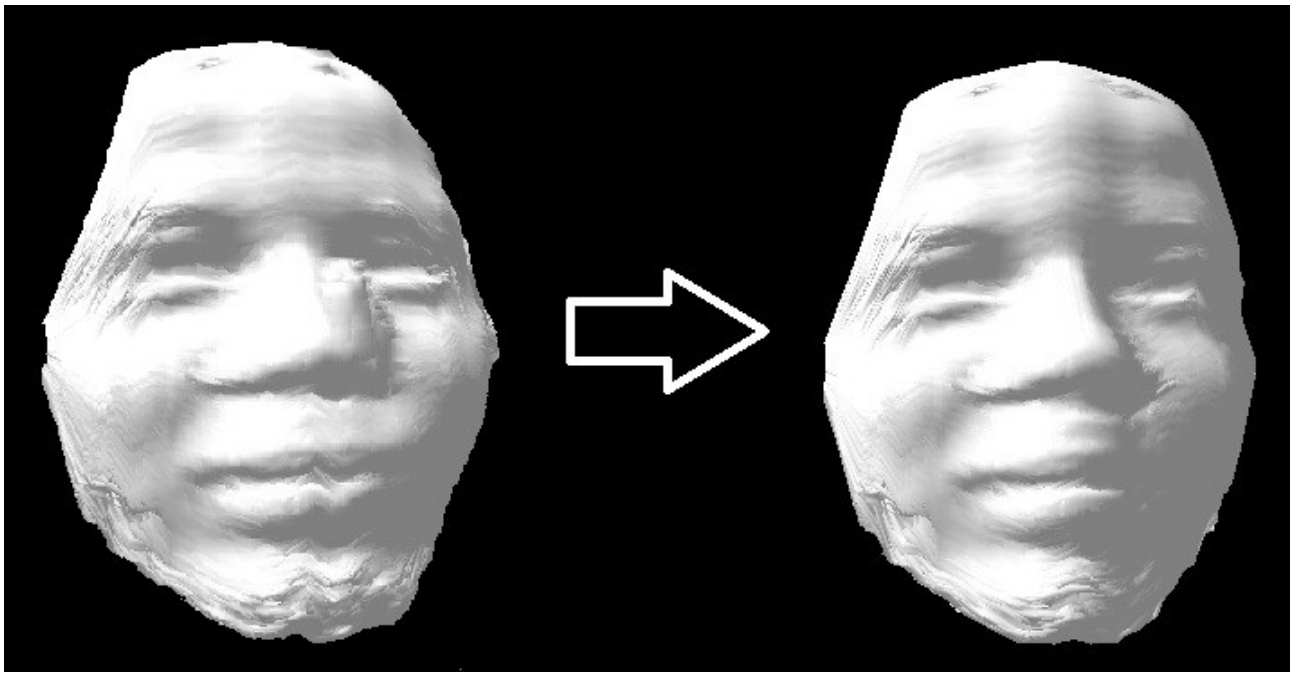


Fig.47 Left the smoothed model and right the mirrored model

9. Performance

The implementation was done on an average computer with 2 Gb RAM, an Intel Core i5 CPU with 2.27 Ghz (4 CPUs) and a NVIDIA GeForce GT 230M. It can be divided into four Steps for which the time was measured separately.

1. Scanning
2. Calculating the Points
3. Improve the data
4. Rendering the mesh

Scanning:

The resolution of the whiteboard was 1024 x 768 pixel. It takes about 150 seconds to move the white line over the whiteboard.

Calculating the Points

This step includes the processing of the pictures and calculating the space coordinates of the extracted Points. Normally out of the 1024 frames we get from a scan the face is covered with about 150 to 200 frames. The images have a resolution of 1280 x 960 and this step takes the longest with about 27 minutes for 200 frames.

Improving the data:

This step includes building the m*n matrix, deleting of impossible data and smoothing the points. Depending on the size of the matrix this step takes a different amount of time as can be seen in Fig.48. It was tested with the matrix line size of 100, 200, 400, 800, 1200 and 1600 and the duration takes between 100 and 600 seconds.

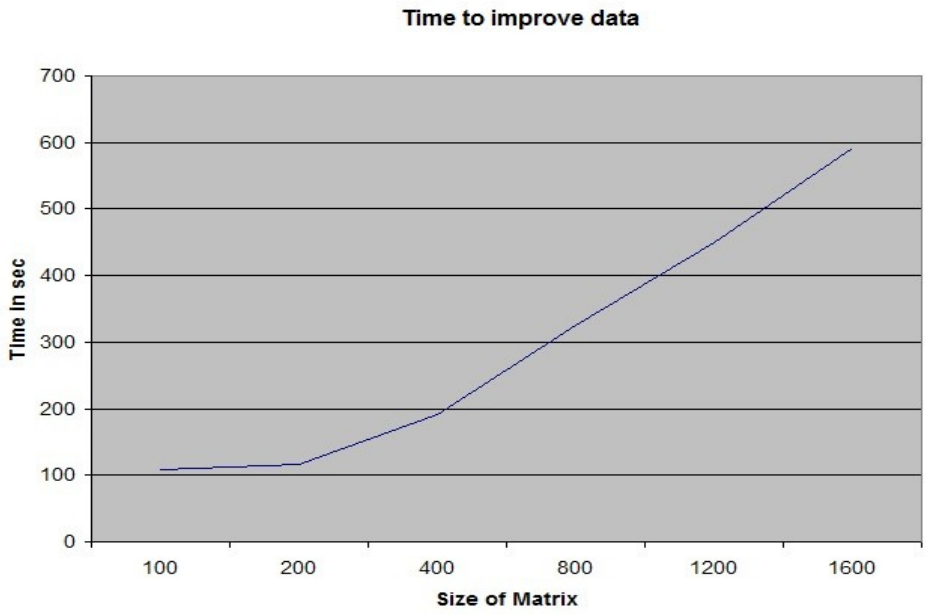


Fig.48 Graph that shows the time it takes to improve the data depending on the matrix size.

Rendering the scan:

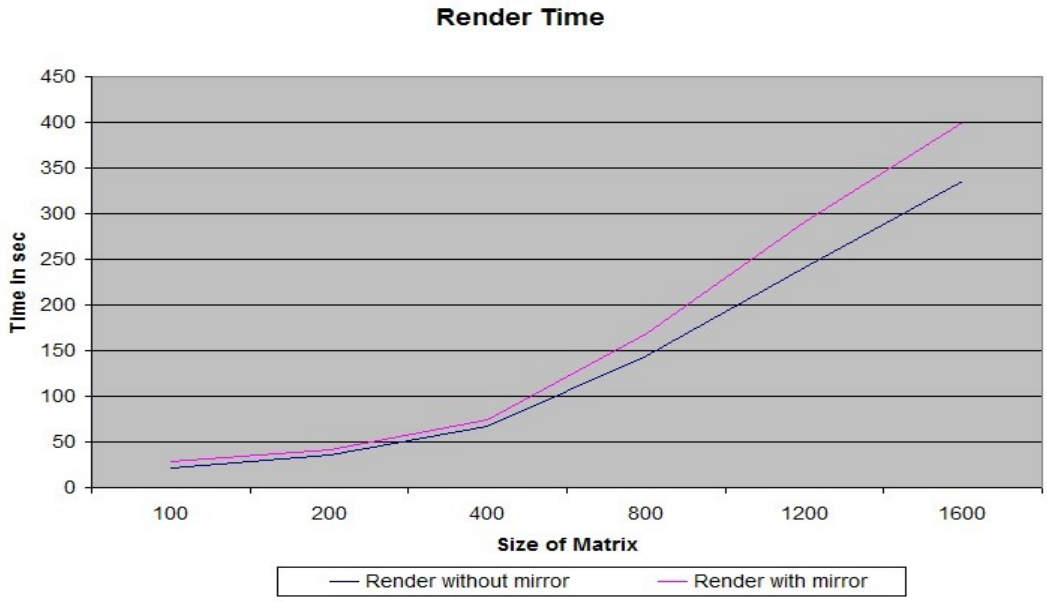


Fig.49. Purple line shows render time with mirroring, blue line without mirroring

The last step was to render the scan in OGRE. Again the time for rendering was dependent on the size of the matrix. If the model is mirrored or not was also a factor during measuring. In Fig.49 we can see a Graph showing the measured times.

All in all the the time from the scan to the finished mesh takes from 30 to 50 minutes. Compared to some other work mentioned in the related work section, that is quite a lot and there is definitely some room for improvement.

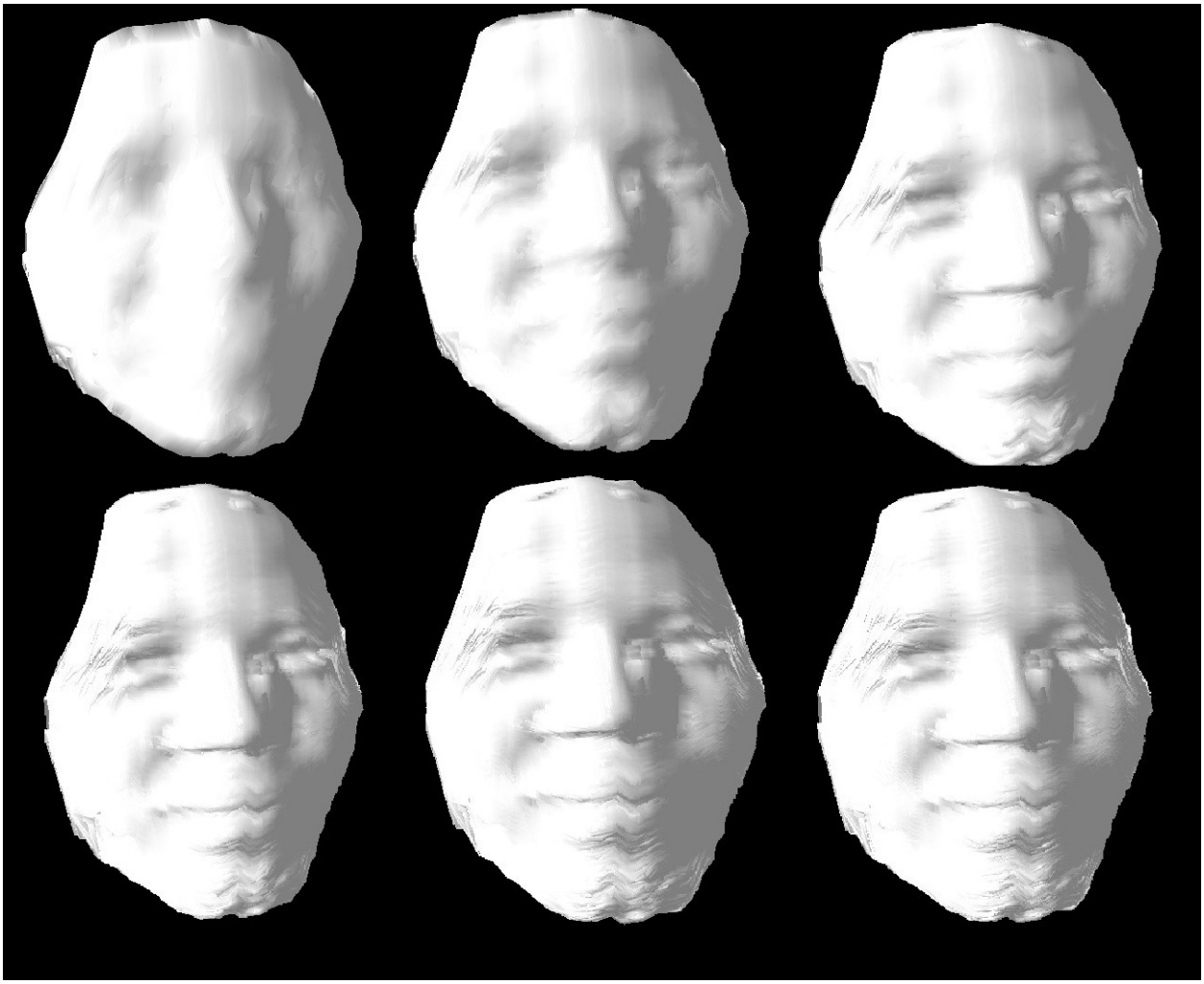


Fig.50. Matrix size in first row from left to right: 100, 200, 400.
Matrix size in second row from left to right: 800, 1200, 1600.

Another important question to consider is what matrix size is necessary and practical to use. Again we tested the matrix sizes of 100, 200, 400, 800, 1200 and 1600.
A matrix size of 100 is definitely not enough. Although rough features are recognizable like some dents where the eyes should be and the tip of the nose but most of the details are missing. A matrix size of 200 is a little better because some face features like eyes and mouth already are distinguishable. But again the features are only very crude and not very detailed. Next we tested a

matrix size of 400 which already gives some good results. The eyes, nose and mouth are clearly visible and recognizable. With a matrix size of 800 the features doesn't get much more detailed. But there is slight improvement like the separation of the lips which gets a bit deeper and the form of the nose gets a bit better and sharper. A size of 1200 and 1600 doesn't improve the data really. Especially the time it takes to render them as seen in Fig.49 is a big disadvantage that's probably not worth it. So for rendering, a matrix size between 400 and 800 should be sufficient. In Fig.50 we can see the meshes for the different matrix sizes. There we can see the improvement in the first row, by increasing the matrix size, but nearly no difference in the second row.

10. Conclusion

In my master thesis I implemented a simple face scanner using an interactive whiteboard. The only other hardware needed was a HD-webcam and a laptop to run the program. The built-in beamer of the interactive whiteboard projects a thin line of light from left to right over the whiteboard. This way the persons face standing in front of it gets illuminated. Every time the line of light moves a step the camera, which had to be calibrated before to know its position and angle, takes a picture. The illuminated points of the face get extracted and through triangulation the space coordinates get calculated. Afterward a m*n matrix was calculated to connect the points into triangles that build up the face mesh. This model finally gets displayed in OGRE, an open source render engine. An mirroring function was implemented that could be used to counter missing data in one half of the face.

The implementation done, is one of the most simplest and there is some room for improvement. The simple line of light used for scanning could be replaced by a time-multiplexing algorithm like n-ary patterns. A one-shot technique isn't really necessary because the resolution is probably a bigger issue then the speed of the scan. Further the algorithm to build the mesh needs a bit of improvement so that the sides of the face doesn't get squeezed that much.

Nevertheless, if time isn't a big issue, this scanner can be used in schools or universities that already have an interactive whiteboard. Though it's optimized for scanning faces, this scanner can also be used for different things.

11. References

[1] Kenneth M. Dawson-Howe, David Vernon; Simple pinhole camera calibration, 2005

[2]Gang Xu and Zhengyou Zhang (1996). Epipolar geometry in Stereo, Motion and Object Recognition. Kluwer Academic Publishers. ISBN 0-7923-4199-6 (page 7-19)

[3] http://en.wikipedia.org/wiki/Polygon_mesh

[4] http://www.baumgart.org/winged-edge/winged-edge.html

[5] J. Vollmer , R. Mencl , H. Müller (1999) Improved Laplacian Smoothing of Noisy Surface Meshes, Eurographics '99

[6] Yutaka Ohtake, Alexander G. Belyaev, Ilia A. Bogaevski; Polyhedral surface smoothing with simultaneous Mesh Regularization, 2000

[7] Jordi Pagès , Joaquim Salvi , Josep Forest; A new optimised De Bruijn coding strategy for structured light patterns, 2004

[8] Joaquim Salvi , Jordi Pagès , Joan Batlle; Pattern codification strategies in structured light systems, 2004

[9] Li Zhang, Brian Curless, and Steven M. Seitz; Rapid shape acquisition using color structured light and multi-pass dynamic programming, 2002

[10] Olaf Hall-Holt, Szymon Rusinkiewicz; Stripe Boundary Codes for Real-Time Structured-Light Range Scanning of Moving Objects, 2001

[11] Nelson G. Durdle, Jaishankar Thayyoor, V.J.Raso , An Improved Structured Light Technique for Surface Reconstruction of the Human Trunk, 1998

[12] Albitar, I.C.; Robust structured light coding for 3D reconstruction, 2007

[13] J. Battle, E. Mouaddib, J. Salvi; Recent progress in coded structured light as a technique to solve the correspondence problem: a survey, 1998

[14] Rıfat Benveniste, Cem Ünsalan; Nary coded structured light-based range scanners using color invariants, 2011

[15] Jens Gühring; Dense 3D surface acquisition by structured light using off-the-shelf components, 2001

[16] J Tajima, M Iwakawa;3-D data acquisition by rainbow range finder, 1990

[17] http://cmp.felk.cvut.cz/cmp/gallery/ccrf.html

[18] Ji Zhou, Anil K. Jain, Tianxu Zhang, Yaoting Zhu, Mingyue Ding, Jianguo Liu; Spectral analysis based range image acquiring method, 1998

[19] Fechteler, P.; Adaptive Color classification for Structured Light systems, 2008
[20] Daniel Modrow, Claudio Laloni, Guenter Doemens, Gerhard Rigoll; 3D face scanning system based on invisible infrared coded light, 2007
[21] Hussam Yousef, Regis Huez, Laurent Hussenet, Michel Herbin; A new robust method of line detection in structured light system, 2012
[22] Volker Blanz, Thomas Vetter; Face Recognition Based on Fitting a 3D morphable Model, 2003
[23] Breuer, P., Kwang-In Kim, Kienzle, W., Scholkopf, B., Blanz V.; Automatic 3D face reconstruction from single Images or Video, 2008
[24] Kam-sum Lee, Kin-hong Wong, Siu-hang Or, Yiu-fai Fung; 3D Face Modelling from Perspective-Views and Contour-Based Generic Model, 2002
[25] V. K. De Wansa Wickramarante, V. V. Ryazanov, A. P. Vinogradov; Accurate reconstruction of 3D model of a human face using structured light, 2008
[26] Alexander Belyaev Yutaka , Er Belyaev Yutaka Ohtake; A Comparison of Mesh Smoothing Methods, 2003
[27] P. Cignoni, C. Rocchini, R. Scopigno; Metro: Measuring errors on simplified surfaces, 1998
[28] http://www.ogre3d.org
[29] http://en.wikipedia.org/wiki/OGRE
[30] Xiao-jun Jia, Zhi-jiang Zhang and Qing-cang Yu; Construction for M-arrays and application in structured light, 2011
[31] http://en.wikipedia.org/wiki/De_Bruijn_graph
[32] Peisen S. Huang, Song Zhang; Fast three-step phase-shifting algorithm, 2006
[33] Giovanna Sansoni, Matteo Carocci, Roberto Rodella; Three-dimensional vision based on a combination of gray-code and phase-shift light projection: analysis and compensation of the systematic errors, 1999
[34] Song Zhang; Shing-Tung Yau; High-speed three-dimensional shape measurement system using a modified two-plus-one phase-shifting algorithm, 2007
[35] Alexander M. Bronstein, Michael M. Bronstein, Eyal Gordon, Ron Kimmel; High-resolution structured light face scanner with automatic calibration, 2003
[36] C.Rocchini, P. Cignoni, C. Montani, P. Pingi, R. Scopigno; A low cost 3D scanner based on structured light, 2001

12. Appendix

12.1 Zusammenfassung

Abstract: In dieser Arbeit wird mit Hilfe eines interactive Whiteboards ein einfacher Gesichts-Scanner implementiert. Basierend auf dem structured light Prinzip wird mit dem Beamer des interactive Whiteboards ein dünner Lichtstrahl auf das Gesicht einer Person projeziert, der sich langsam über das Gesicht bewegt. Dies wird mit einer handelsüblichen HD-Webcam aufgenommen, die vorher kalibriert wurde. Danach wird das gescannte Gesicht mit Hilfe des Open-Source 3D-Programms Ogre3d dargestellt.
Dabei werden die verschiedenen Schritte die der Gesichts-Scanner durchläuft genau beschrieben. Beginnend bei der Kalibration der Kamera, deren Position und Winkel mithilfe von Punkten im Raum berechnet wird. Danach wird das sehr einfache Prinzip eines Scanners mithilfe eines Lichtstrahls erklärt, mit deren Hilfe die Punkte des zu scannenden Objekt abgetastet werden. Eine Methode zur Extraktion der relevanten Punkte, mit der open source library OpenCV, wird vorgestellt. Dabei wird der Canny-algortihmus zur Kantenerkennung eingesetzt. Hat man die relevanten 2D-Punkte kann durch Triangulation die dazugehörigen 3D-Koordinaten berechnet werden. Um den Scanner auf menschliche Gesichter zu optimieren werden verschiedene Regeln festgelegt, die falsche Daten verhindern sollen. Zusätzlich werden fehlende Punkte interpoliert und der Laplace-Algorithmus verwendet um die Oberfläche zu glätten. Am Ende wird die open-source 3D Grafik engine OGRE verwendet um die Punkte als Mesh darzustellen. Dabei muss vorher die Reihenfolge festgelegt werden in der die Punkte miteinander zu einem Mesh verbunden werden. Dies wird mit einer Matrix erreicht, bei der die benachbarten Punkte nur noch miteinander verbunden werden müssen. Das Resultat ist dann ein 3D-Model des gescannten Gesichts.

12.2 Summary

Abstract: This thesis describes the implementation of a face-scanner with the help of an interactive whiteboard. Based on the principle of structured light, the projector of the whiteboard projects a thin light beam on the face of a person which moves slowly from over the face. An off-the-shelf HD webcam, which was calibrated before, was used to capture the scan. At the end the scanned face gets displayed as a 3D-model with the help of the open-source 3D engine OGRE.
The various steps of the face-scanner are described very precisely. Starting with the calibration of the camera, which position and angle get calculated with the help of Points in space. Then the simple principle of scanning with a light beam gets explained, with which the surface of the face gets scanned. A method to extract the relevant points, with the open source library OpenCv, is introduced. The canny-algorithm is used for border recognition. When the 2D points are known we can calculate the corresponding 3D-coordinates. To optimize the scanner for human faces some rules are defined to prevent wrong data. Additionally missing points are interpolated and the Laplace-algorithm is used to smooth the surface. At the end the open-source 3D graphic engine OGRE is used to display the points as mesh. Before the order to connect the points with each other to a mesh had to be defined. This is done with the help of a matrix, in which the adjacent points just had to be connected. The result is a 3D-model of the scanned face.

Lebenslauf Philip Beutl

PERSÖNLICHE ANGABEN

Name: Philip Beutl

Geburtsdatum: 03.02.1987

Geburtsort: Wien

Familienstand: ledig

AUSBILDUNGSWEG

2009 - *, Master Medieninformatik - Universität Wien
2006-2009, Bachelor Informatik - Universität Wien
2005-2006, Präsenzdienst
1997-2005, Gymnasium Ettenreichgasse, Abschluss mit Matura
1993-1997, Volksschule Tesarekplatz

BERUFSERFAHRUNG

Sommer Semester 2012 Tutor für Game Design an Universität Wien
August 2010 – Juni 2012 Dataflow- and Processmanager bei PVM Dataservices
2008-2010 Promotion für die österreichischen Lotterien