# Simulation of a setup for phase measuring deflectometry

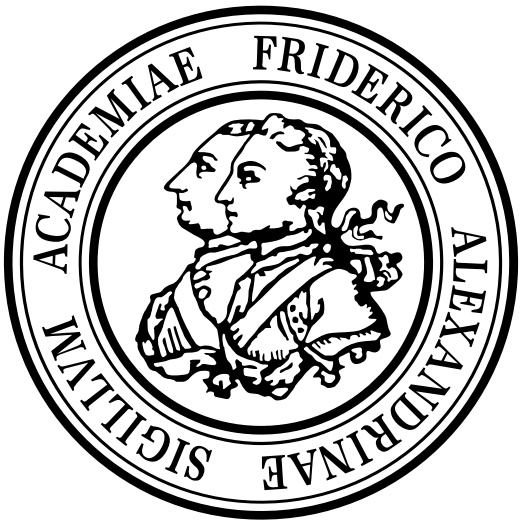
## Masterarbeit aus der Physik

vorgelegt von

**Stefan Pickel**

Tag der Abgabe: 19. Dezember, 2014

Erlangen Centre for Astroparticle Physics
Physikalisches Institut
Friedrich-Alexander-Universität
Erlangen-Nürnberg




1. Gutachter: Prof. Dr. Christopher van Eldik
2. Gutachter: Prof. Dr. Gisela Anton

# Contents

# Abstract

In this thesis, a simulation of Phase Measuring Deflectometry (PMD) was generated. PMD is a method for measuring specular surfaces and is considered to be used for measuring optical properties, the point spread function (PSF) to be precise, of mirror facets of a future Imaging Atmospheric Cherenkov Telescope (IACT), the Cherenkov Telescope Array (CTA).
PMD uses the distortion of a pattern that is reflected on a specular surface by taking and evaluating images of the reflection. These images were simulated by ray tracing and introduced into the existing evaluation chain.
Multiple studies were performed to test the capabilities and limits of the method.
PMD could be shown to be capable of the required precision, as long as the setup is properly calibrated, though already small deviations from this calibrated state were found capable of causing great errors.
Special attention was directed towards the current methods of evaluation, that include a recalibration of the geometry of the setup. This was found to potentially lead to an overestimation of the reliability of PMD. Actually incapable mirrors could be found suitable for use on CTA telescopes.
There are still systematics of the current PMD setups that need to be understood, but the simulation that was created over the course of this thesis can be used as a framework for studies in this direction.

In dieser Arbeit wurde eine Simulation der Phasenmessende Deflektometrie (PMD) erstellt. PMD ist eine Methode zur Vermessung spiegelnder Oberflächen und wird für die Mesungen von optischen Eigenschaften, die Punktspreizfunktion (PSF) um genau zu sein, von Spiegelfacetten eines zukünftigen Cherenkov Teleskopes (engl: Imaging Atmospheric Cherenkov Telescope, IACT), dem Cherenkov Telescope Array (CTA).
PMD nutzt die Verzerrung eines Musters das auf einer spiegelnden Oberfläche reflektiert wird, indem es Bilder der Reflektion aufnimmt und auswertet. Diese Bilder wurden mittels Raytracing simuliert und in die bestehende Auswertekette eingebracht.
Mehrere Studien wurden durchgeführt, um die Fähigkeioten und Grenzen der Methode zu testen.
Es konnte gezeigt werden, dass PMD zu der nötigen Genauigkeit fähig ist, soweit der Aufbau genau kalibriert ist, allerdings wurde entdeckt, dass bereits kleine Abweichungen vom kalibrierten Zustand große Fehler verursachen können.
Besondere Aufmerksamkeint wurde auf die derzeitigen Auswertemethoden gewandt, die eine Nachkalibration der Geometrie des Aufbaus beinhalten. Es wurde entdeckt, dass diese zu einer Überschätzung der Verlässlichkeit von PMD führen kann. Eigentlich ungeeignete Spiegel könnten so als geeignet befunden werden, auf CTA Teleskopen verwendet zu werden.
Es gibt noch immer Systematiken in den verwendeten PMD Aufbauten, die noch verstanden werden müssen, doch die Simulation die im Laufe dieser Arbeit erstellt wurde kann als Grundgerüst für darauf abzielende Studien dienen.

# 1 Introduction

## 1.1 The Cherenkov Telescope Array

Among the components of the cosmic radiation, gamma rays stand out as one of the few kinds of particles that do not lose the information of their origin on the way to earth. This makes them very interesting for finding the accelerators of the cosmic radiation and other areas that produce these high-energy photons, like molecular clouds that interact with the cosmic radiation of nearby accelerators.
One type of telescope that is designed to detect gamma rays is the Imaging Atmospheric Cherenkov Telescope (IACT). Using Cherenkov light generated by the charged particles of the electromagnetic cascade induced by high-energy gamma rays when interacting with the atmosphere, these telescopes take images of the cascade and use them to reconstruct the energy and origin of the gamma ray.
Cherenkov light is generated when charged particles travel through a medium with a velocity higher than the speed of light in that medium. Due to the particles' high velocity, the relaxation of the polarisation generated in the medium leads to the coherent emission of light, generating a cone shaped shock wave around the path of the particle. The emission angle can be used to reconstruct the velocity $v$ of the particle, as it follows

$$\cos\theta = \frac{1}{n \cdot \beta} = \frac{c}{n \cdot v}$$

with the refractive index $n$ and $\beta = \frac{v}{c}$.
The intensity of the Cherenkov light has its maximum in the blue to ultraviolet part of the spectrum.
When a gamma ray photon of very high energy interacts with a particle in the atmosphere, it produces an electron positron pair, each with roughly half of the photons original energy. Both of the charged particles can then emit new photons via bremsstrahlung, dividing the original energy approximately equal among all four particles. As long as the new particles have enough energy to interact with the atmosphere by means of bremsstrahlung (electrons and positrons) and pair production (photons), the number of particles in this cascade increases exponentially. One byproduct of this electromagnetic particle shower is Cherenkov light induced by its fast, charged components. This results in a faint flash of light, which can be observed with sufficiently sensitive detectors.
IACTs use (segmented) mirrors to focus the Cherenkov light of the particle shower into a camera made of photomultipliers. This basically generates an image of the cascade, which can be used to reconstruct the shower axis and calculate the energy and origin of the original photon. Using an array of multiple IACTs, the precision, energy threshold and effective area can be improved greatly, especially regarding the direction of the particle. This has been used by multiple IACT systems like Hegra (High-Energy-Gamma-Ray Astronomy), Veritas (Very Energetic Radiation Imaging Telescope Array System), Magic (Major Atmospheric Gamma-ray Imaging Cherenkov Telescopes) or H.E.S.S. (High Energy Stereoscopic System).
In the near future the Cherenkov Telescope Array (CTA), the then biggest array of IACTs, will be constructed. Current plans include two separate arrays on both the northern and southern hemisphere.
Both will consist of multiple IACTs of different size (Large, Medium and Small Size Telescopes; LST, MST and SST) that provide coverage for different parts of the gamma spectrum in total. An artistic expression of how a CTA array may look like can be seen in figure 1.

**Figure 1:** Artistic expression of a CTA array [5]

## 1.2 CTA mirror facets

One important part of an IACT is the mirror, which consists of multiple individual segments. In case of the MSTs, these segments are hexagonal mirror facets with a diameter of 1.2 m flat-to-flat and a radius of curvature of $\approx 32$ m. Unlike the mirrors used in most IACTs or optical telescopes, they are not massive bodies of glass but instead consist of a thin layer of mirrored glass on a base with a stable honeycomb structure. This design makes each facet much lighter compared to conventional mirrors of the same size.
Each of the segments has to focus the light into one of the photomultipliers of the camera. To ensure this, the point spread function (PSF), which is the reflection of a point-like light source at a certain distance of the mirror, has to be smaller than the opening of the photomultiplier. To quantify the size of the PSF, the quantity d80 was introduced, which is the diameter of a circle that contains 80 % of the light reflected by the mirror. The requirement for the optical properties is specified as a d80 that is smaller than $\frac{1}{3}$ of the opening of the photomultiplier, which is 17 mm in case of the mirrors intended to be used on MSTs.

## 1.3 Measuring methods

To ensure that each mirror meets these requirements, they have to be tested.
One common approach for measuring the PSF of a mirror is the 2f method, which is shown in figure 2. For this, a small, as close to point-like as possible, light source is placed at 2 times the focal length of the mirror, which is equal to its radius of curvature in case of a spherical mirror, and the reflection spot of the light source is measured right next to it. The reflection can be measured by scanning the illuminated area with a photo detector and reconstructing the PSF from this data, or, if it is small enough, simply taking one image with a conventional CCD camera. Independent of the way the PSF is measured, this method needs precise alignment of the light source, mirror and detector.

A different method that is not that sensitive to the placement of the mirror was developed together with the Optical Sensing, Metrology and Inspection (OSMIN) group at the University

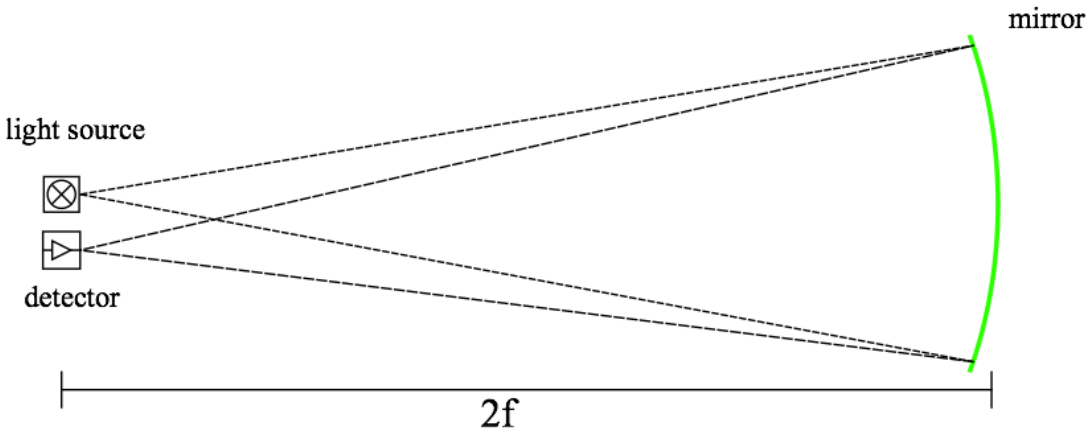



**Figure 2:** Sketch of 2f measurement setup.
The distance between the light source and the detector is shown extremely increased to demonstrate the light rays more clearly. [2]

of Erlangen. Using Phase Measuring Deflectometry (PMD) the surface of the mirror is measured and the PSF is calculated using a simple ray tracing algorithm.
The method is based on observing a defined pattern after it is reflected by the measured surface and using the distortions it has undergone to calculate the local slope, curvature and height profile of the mirror. For this, a striped pattern with sinusoidal brightness distribution is projected on a screen and its reflection in the specular surface is observed using one or more cameras.
A more detailed description of this method can be found in chapter 2.1.

There are actually two types of PSF, the 1f and the 2f PSF.
The 1f PSF is generated by reflecting light from a distant source into the focal point of the mirror. For distances $\gg f$, the light can be assumed to arrive in parallel rays and its reflection will be smallest in size at a distance $f = \frac{r}{2}$ from the mirror. To focus the incoming light into a single point, the mirror actually would have to be to be parabolic, but the ones planned for use at CTA consist only of a segment of the whole sphere, small enough to allow for spherical aberration to be neglected. Due to the long distances that are required, the 1f PSF is seldomly measured directly.
The 2f PSF is generated by reflecting the light from a point-like source in the centre of the sphere back into the same spot (or very close to it), similar to what is actually done for the classical 2f measurements.
Due to the way the mirror is intended to be used, the 1f PSF is the relevant one, but it can not be measured as easily as the 2f PSF. For an ideal mirror, both PSFs would have close to no extent, and the 2f PSF for real mirrors is enlarged by a factor of 2 compared to the 1f PSF, allowing for it to be used as a viable condition for testing the mirrors.

## 1.4 Simulating

PMD is not established for measuring objects of the size of CTA mirror facets with the required precision. The PMD setups that are currently used and the methods to evaluate the measurements are also not properly tested yet.
It is therefore necessary to verify the capabilities and limits of the method.

A simulation that emulates the whole measurement process by ray tracing the images that would actually be taken by the setup's cameras was chosen for this purpose.
Ray tracing is a method of generating images by tracing the path of individual light rays backwards from the camera through a 3D environment to determine the color values for each pixel.

To reduce interference with the methods of evaluation, the simulated images were converted to a format that is readable by *softPMD*, the software for PMD and used as primary input instead of the cameras. From there, the evaluation process could be used without any major changes.
The method for generating the images is described more detailed in chapter 2.2, while several studies performed with this simulation technique are presented in chapter 3.

# 2 Background of the simulations

## 2.1 PMD measurements

### 2.1.1 General overview

A PMD setup consists of a screen and a number of cameras. The object that is to be measured is placed in a way that allows the cameras to see the reflection of the pattern on the screen (figure 3). The cameras focal point is adjusted to produce a sharp image of the object, which results in a blurred image of the pattern on the screen.
By shifting the phase of the sinusoidal patter multiple times and measuring it using the brightness, it is possible to identify the point on the screen that is seen for each of the camera pixels.

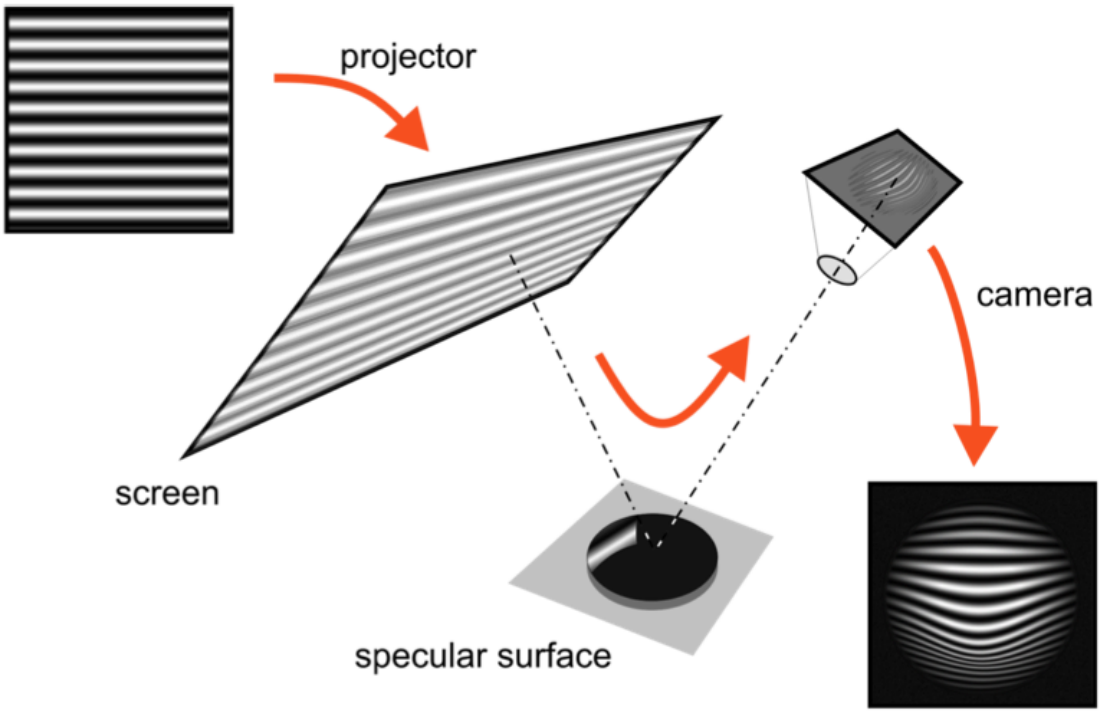



**Figure 3:** Sketch of PMD measurement principle.
The pattern is projected on a screen or ground glass and reflected by the object. The camera takes pictures of the then distorted pattern. [3]

As only the starting and end points of the sight paths as well as their direction vector after the reflection are known, but not the direction vector before the reflection, additional information is needed to identify the local slope of the mirror as seen from each camera pixel. As figure 4 illustrates, the slope can vary strongly depending on the height at which the reflection takes place. To solve this, a stereo method that uses the data of two cameras that observe the object from different angles was developed by Markus Knauer of the OSMIN Group [3]. By using two cameras with a sufficient stereo angle, it is possible to determine the slope that leads to both images, which is no longer ambiguous.
This method is not feasible for measuring the CTA mirror facets due to the geometry of the setups and an alternate method, which is described in chapter 2.1.4, is used.

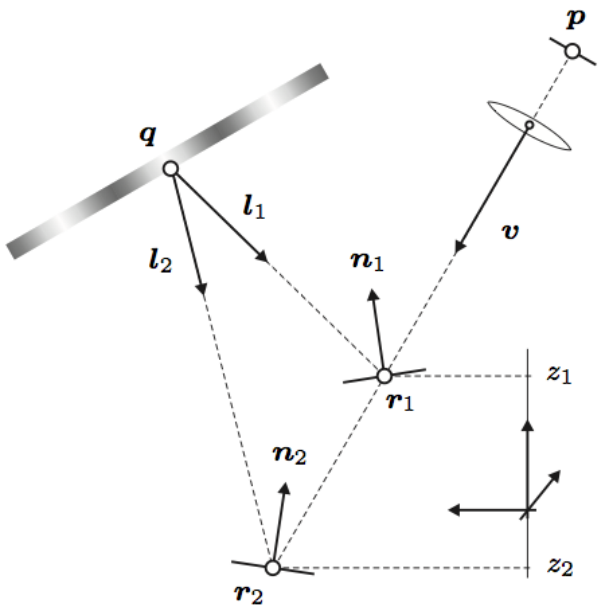



**Figure 4:** Ambiguity of the slope.
The known points $p$ and $q$ and vector $\vec{v}$ are not sufficient to identify the slope unambiguously. Depending on the height $z$, different sight paths $\vec{l}$ starting at the screen are possible, resulting in different normals $\vec{n}$ and slopes $\vec{r}$. [3]

### 2.1.2 The Long and Short Working Distance setups

Two different setups for measuring the CTA mirror facets were constructed in Erlangen, usually called Long Working Distance (LWD) setup and Short Working Distance (SWD) setup, that are shown in figure 5 and 6.

The LWD setup consists of one camera that is placed right next to the screen in a distance to the mirror equal to its radius of curvature. The setup needs the same amount of space as a classical 2f measurement of the PSF, as well as a comparable effort to place the mirror correctly, as the reflection of the screen is of a comparable size to the PSF of the mirror.
The original intention was to use two cameras to evaluate the data with the stereoscopic method, but due to the long distance between the screen and cameras and the mirror, the stereo angle is too small.
Due to the small reflection angles, the exact position the reflection takes place does not have a great influence on it and it is possible to evaluate the measurements with the LWD setup by assuming the height of the reflection point to be identical to the distance between the screen and the mirror, which can be measured with a laser distance sensor.

The SWD setup was designed as a more compact setup, with the possibility to fit it into a room instead of using a long passage, as one intended use is to test the temperature dependency of the mirrors properties in an environmental chamber.
It uses four cameras, each monitoring a part of the screens reflection in the mirror in a way that parts of the mirror are covered by multiple cameras. As the overlap area between the views of different cameras is small compared to the whole surface of the mirror, the stereoscopic method of evaluation can not be used effectively on this setup either, which is why the method described above is also used for SWD measurements.

**Figure 5:** Photo of the LWD setup screen and cameras.
The setup was initially designed to use a stereoscopic method to evaluate the data, which was not possible because of the small stereo angle between the two cameras. [1]

The way the setup was constructed ensures that the exact position of the mirror is not important for the measurements. Another advantage is the wide range of radii of curvature that can be measured without changes to the setup itself.
For the evaluation of the data taken with the SWD setup, the assumption that they are part of a perfect sphere is used to provide a first iteration for the surface from where the normals and surface can be calculated properly.

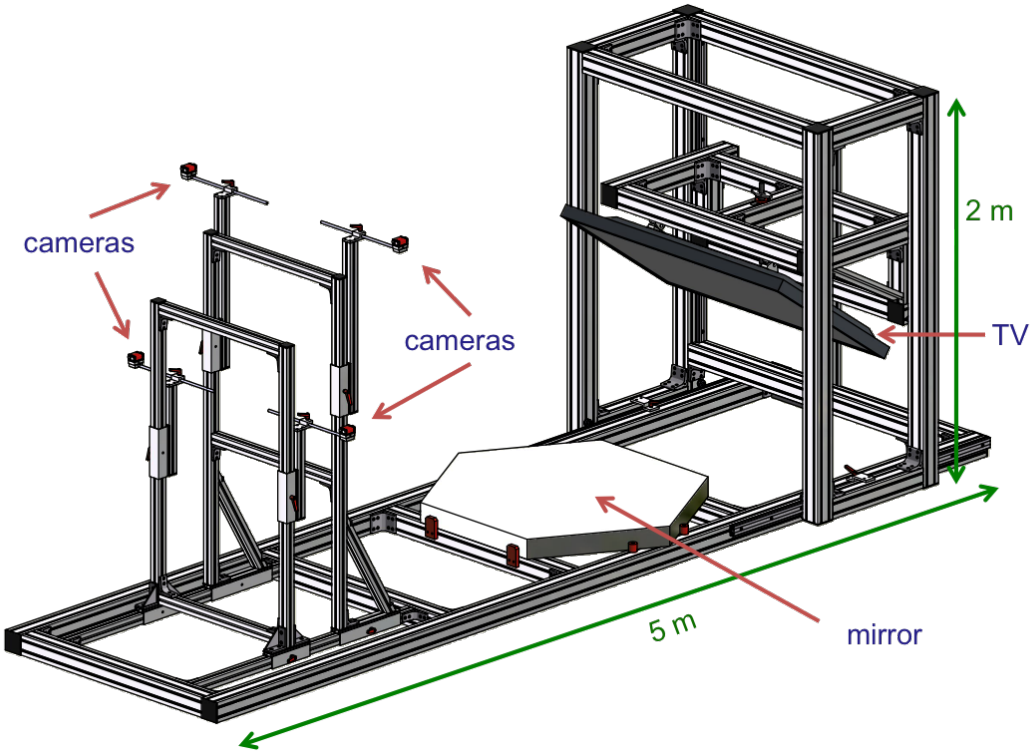



**Figure 6:** Technical drawing of the SDW setup [1]

### 2.1.3 Calibration and calibration parameters

For the results of measurements and evaluation to be reliable, a properly calibrated setup is needed. The different steps needed for the calibration of the SWD setup are listed below, followed by a more detailed description.

- Internal Calibration (of the cameras, Camera Calibration)
- Internal Calibration (of the screen, Screen Calibration)
- External calibration of screen and cameras (Geometry Calibration)

All parameters derived by the calibration are saved in the corresponding configuration files for the cameras and screen, examples of which can be found in the appendix.

## Internal Calibration

Since PMD is based on the assumption that each camera is an ideal pinhole camera, it is important to consider the fact, that the measurements are taken with real, non-ideal cameras. This is done by using an extended pinhole camera model, which allows for calculations to compensate the distortions caused by the real objective lens. When evaluating the measurement, the extended pinhole model can be used to derive substitute positions for each pixel of the chip, which are then used as starting points for the sight paths.
Depending on the kind of screen used in an individual setup and the way it is mounted there, the assumption of flatness does not always hold. Especially in the SWD setup, where the 60" display is mounted facing downward and only held at the back, it is strongly deformed, with an amplitude of two to three millimeters. These deformations have to be taken into account, which is done similar to the cameras, usually using polynomials to describe the surface of the screen. The parameters for either of these models are determined with the internal calibration.
For the calibration, several images of a calibration pattern are taken, that have to cover a wide range of relative orientations of the pattern and camera.

The calibration pattern consists of tags on a contrastingly coloured surface (typically white tags on a black surface or black tags on a white surface). Each of the tags has a number and its position on the plate is precisely known in its own coordinate system.
For the camera calibration, the pattern is placed on a flat plate, while it is simply displayed on the screen for the screen calibration.
These images are then evaluated with a photogrammetric software, where each of the tags is assigned its correct number. There are different methods to help identify the first tags of the pattern, for example additional marks at the edge of the pattern next to certain tags or special marks in the pattern (replacing multiple tags if necessary, as can be seen in figure 7).
With the tags in each image labelled correctly and with the information about the pattern, the parameters of the models for both the camera taking the images and the surface the pattern is on can be calculated simultaneously.

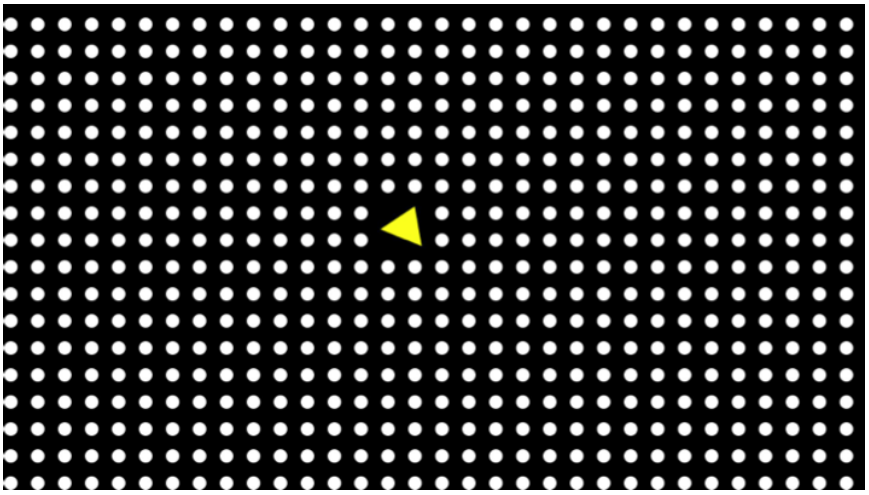


**Figure 7:** Pattern used for the calibration of the screen.
A similar pattern with additional marks on the side instead of the centre is used to calibrate the cameras. [1]

## Geometry Calibration

When both the cameras and the screen are calibrated, the geometry of the whole setup can be determined with the geometry calibration. For this, a set of measurements, following the procedure described in chapter 2.1.4, of a perfectly spherical mirror (or, in case of the SWD setup a H.E.S.S. II mirror, which is the available surface closest to a perfect sphere of the size needed) is created and evaluated with a dedicated calibration software.
To compensate the flatness of the H.E.S.S. II mirror due to its high radius of curvature of $\approx 72\,\mathrm{m}$, there are 9 measurements for calibrating the SWD setup covering a wide range of angles. Smaller setups, especially the ones for eyeglasses or other objects of that size, can be calibrated with much smaller spheres and need a smaller number of images.
The calibration algorithm calculates, what points on the screen would be seen by the cameras after being reflected on a perfect sphere of a fixed radius of curvature equal to the one of the calibration mirror and a position that is not fixed. The positions and orientations of the cameras are then minimised, using the average distance between the calculated points and the ones that were actually observed as a measure to determine how suitable the current constellation is.
This leads to the calibration parameters shown in table 1, that define the coordinate system of each object relative to the shared coordinate system of the whole setup.
In the shared coordinate system, $x$ points to the right when looking from the cameras to the

screen, $y$ points from the cameras in the direction of the screen and $z$ points upward.

| parameter | meaning |
|---|---|
| zerobase | translation of the object's coordinate system relative to the shared coordinate system |
| xbase | $x$ base vector of the object's coordinate system |
| ybase | $y$ base vector of the object's coordinate system |

**Table 1:** Parameters of the geometry calibration.
'Zerobase' points at the position of the screen and cameras, 'xbase' and 'ybase' fix the orientation of the coordinate system in the 3D space.

### 2.1.4 The measurement and evaluation procedure

The process of measuring the PSF of a mirror can roughly be devided into three steps, each of which is done using a separate software. Data is taken using *softPMD*, and used to calculate the surface data of the mirror using an external programme. A third programme is used for ray tracing the PSF of the mirror. A scheme of these three steps is hsown in figure 8.

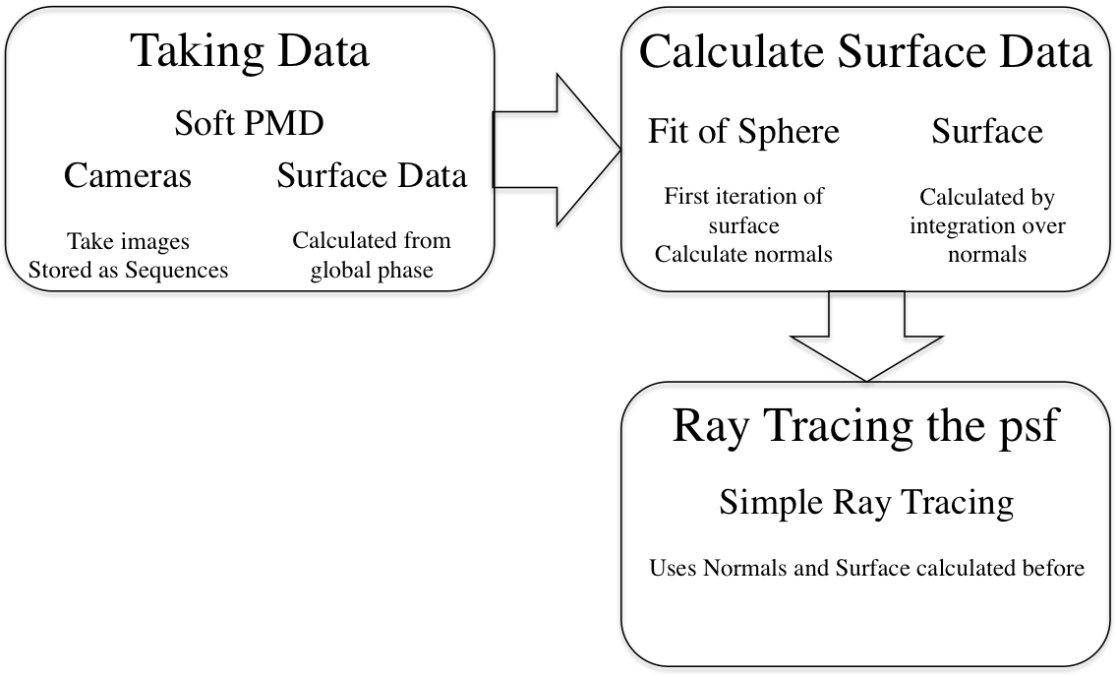



**Figure 8:** Scheme of the measurement and evaluation process.
The cameras are used to take images of the distorted screen pattern. This is done from *softPMD*, where the pattern is automatically translated into projector coordinates and screen data. The screen data is then used for further evaluation with a software that calculates the surface data and from that evaluates the correponding PSF.

## Taking data

The measurement is controlled from the user interface of *softPMD*.
It consists of two steps that are executed in a similar way. The only difference is the use of different stretch factors ($SF$). This parameter defines the pattern and is essential for precise measurements as described below.
During the phase shifting, two coprime sine functions have to be used to allow for unambiguous calculations, as the periodic nature of a single sine function limits all results to phase shifts of multiples of $2\pi$. To ensure that the two sine functions are coprime, they are set to display $P \cdot SF$ and $Q \cdot SF$ full periods over the width of the screen, where $P$ and $Q$ are two coprime numbers. Usually, $P = 7$ and $Q = 8$ are used.
This divides the screen into $SF$ identical areas, that can be identified precisely using another measurement with $SF = 1$.
Due to the way the sinusoidal pattern is generated, the stretch factor also defines the width of the individual stripes. There is an ideal width of the stripes, that corresponds to the size of the spot each camera sees on the screen due to the blurring and leads to the best results. A good indicator for this is the local phase contrast, that should ideally be close to $K = \frac{2}{\pi}$. A detailed derivation of these findings can be found in [3]. The ideal stretch factor for the SWD setup in Erlangen is $SF = 24$.

In practice, a reference measurement with $SF = 1$ is taken first, followed by the measurement with the ideal stretch factor.
A total of 32 different stages of the sinusoidal pattern are used for the measurement, as each measurement with a given stretch factor consists of 8 stages of the phase shifting to determine either of the horizontal and vertical projector coordinates (resulting in a total of 16 images per stretch factor) and two different stretch factors are used.
After setting the stretch factor and starting the measurement, the programme automatically takes images with the screen displaying the correct patterns. From this, it also derives the global phase and the coordinates of each point seen by the camera in the screen's own coordinate system, the so called projector coordinates $px$ and $py$ along the screen's $x$ and $y$ axis. This is then used to calculate the corresponding 3D coordinates (in the global coordinate system) for each point on the screen, the so called screen data.
Both measurement steps with $SF = 1$ and $SF > 1$ together lead to a result that is both unambiguous and precise. The measurement is then saved for further evaluation.

## Calculating the surface data

The first step in the evaluation is to cut out the actual mirror area from the images. This is currently done using the local contrast on the global phase, as it is different for regions on the mirror itself and on the surrounding area. Everything within the set interval (usually $\approx 0.3$ to $\approx 0.8$) is considered to be the surface of the mirror, the data from the other areas is not used for further calculations. Any pixels within the interval, that do not have three adjacent pixels that are also within the interval, are also not considered as part of the mirrors surface.
The translation of the screen data to surface data for the mirror starts with a similar algorithm to the one used for the geometry calibration described in chapter 2.1.3, this time with the radius and position of the perfect sphere as the output instead of the geometry parameters.
This is, in the current state, accompanied by a recalibration of the geometry of the setup, allowing the geometry parameters to be changed, to compensate minor deviations from the calibrated state. The recalibration is either done using the measurement itself along with the fit of the sphere, or on a separate reference measurements of a mirror with known radius of curvature. In the second case, the radius of the reference sphere is not variable but instead, the process is the same as the geometry calibration but on only one dataset.
The surface of this ideal sphere is then used as base for the calculation of the local normals. At each point of a $xy$ grid, the axes identical to the ones from the global coordinate system used by *softPMD*, the height value given by the surface of the sphere is used as attachment point for the normals to the surface, allowing for them to be calculated. Based on the normals, the height (usually called surface) can then be recalculated more precisely. Experience shows, however, that the surface has a negligible influence on the PSF and the d80 due to the small deviation from a sphere that most of the tested mirrors show. Therefore, calculating the surface is not necessary for the mirrors that are currently tested, if this information is not needed otherwise. The normals are not calculated again in this process.
Along with the surface and the normals, the local slope and the slope and surface deviation is calculated. The slope is divided into its components in $x$ ans $y$ direction. The slope and surface deviation is calculated by subtracting the slope (again in $x$ and $y$ components) and the surface of the perfect sphere from the slope and surface data.

## Ray tracing the PSF

The ray tracing algorithm uses the normals and surface for the mirror and performs scans around the specified focal length and radius of curvature, creating a PSF and corresponding d80 value for each distance.
It uses a simplified ray tracing algorithm that generates the incoming ray by connecting each surface point to the light source and then calculates the reflected ray using the normals corresponding to the surface point. The intersection of each reflected ray and the image plane is then used for calculating the d80, which is the diameter of the circle around the centre of gravity that contains 80 percent of these intersection points.
Scans of both the 1f and 2f PSF are performed by the ray tracing algorithm, each around the nominal value of the focal length and radius of curvature. This results in the output of a d80 value at both the nominal and the optimal distance ($d80_{nom}$ and $d80_{opt}$), the first one being the nominal radius of curvature of the mirror (or the focal length in case of the 1f PSF) and the second one the distance that generates the smallest d80, which can be seen as the measured radius of curvature. In the optimal case, these are identical.
This method of measuring the PSF of the mirror facets does not provide a statistically verified error on either the measured radius of curvature, or the d80. The error on the radius of curvature can be estimated by the minimum step size of the scan, which is 2 cm. Similarly, the d80 can be calculated with a precision of $\pm 0.1$ mm for individual measurements.

## 2.2 Ray tracing the SWD PMD setup with Mitsuba

Using images generated by ray tracing instead of the real cameras and implementing them into the usual evaluation process was decided as the ideal way to simulate PMD for testing its capabilities and limits.

### Ray tracing using specialised programmes

While the PSF is determined using ray tracing, it is a rather simple and situational programme that can not be used for more complex tasks like generating the images needed for the simulation of the whole PMD setup. This is instead done with a programme specially developed for ray tracing.
These programmes calculate the path of the light from each pixel of the camera through a 3D environment, usually called scene. At each intersection with an object in the scene, the continuation of the path is calculated based on the shape of the object and the information about it's surface. Shadow rays starting at each intersection point try to connect to light sources and generate good estimations of the shadows at their starting point, which accelerates the calculations.
Specialised ray tracing programmes properly take effects like reflection, absorption and refraction of light into account, sometimes using models specifically developed for the purpose of using them for scientific applications. In the end, the calculation leads to the color for each pixel, resulting in an image of what the camera sees. This concept is shown in figure 9.
Tracing rays backwards from the camera, in contrast to the way light actually travels, greatly reduces the computing time, as every traced ray contributes to the image this way, instead of ending outside of the field of view of the camera.
Using multiple samples per pixel, which means tracking multiple rays and averaging over the results can increase the quality of the simulated images, especially if many objects with diffuse surface are placed in the scene.

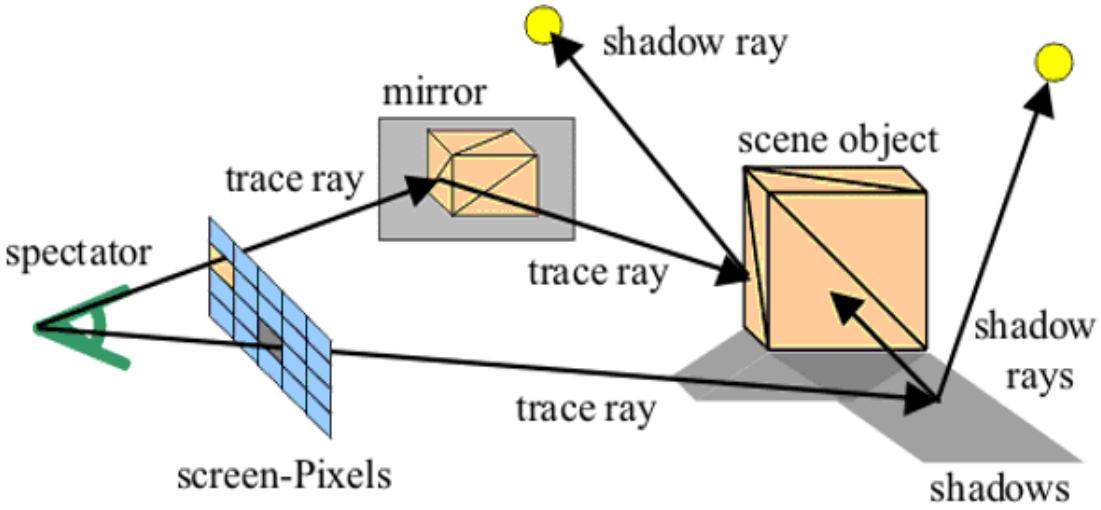



**Figure 9:** Conceptual sketch of ray tracing.
Starting at the camera, one ray hits a mirror in the scene and has only one possile way it can go. the so called shadow rays that start at the intersection of the reflected ray and other rays that do not hit the mirror try to connect to light sources and generate an estimation of the shadows at theit intersection points. [7]

# Mitsuba

The physical renderer[1] Mitsuba was chosen for this simulation, especially because it is an open source programme, which allowed for easy access to the source code. This was necessary as there was no suitable method for simulating the mirror, but Mitsuba contains a class for including analytically calculated spheres to a scene. To simulate the surface geometry of a CTA mirror, some modifications were made to this class to allow for the inclusion of only a part of the sphere to a scene.
This has advantages over adding the mirror using meshes created with 3D modelling software or with the height field class of the ray tracer. Both methods would lead to an imperfect surface description, as meshes are constructed out of polygons, that approximate the surface they represent, while the height field, in case of Mitsuba, is read in from an image file, using the brightness of the pixels to assign each point in a $xy$ grid a $z$ value.
The sphere class was duplicated and the copy was rewritten to not consider intersections of the ray and the sphere that are outside of certain boundaries. The first changes were done to the bounding box of the new object, which is used by the ray tracer to determine if an intersection of the object and any given ray is possible at all. Any ray that does not cross the bounding box simply ignores the shape.
This alone was not sufficient, as the programme automatically chooses the intersection point that is closer to the origin of the ray, which resulted in the intersections being calculated for the outer side of the sphere outside of the bounding box in many cases. Therefore, a comparison of the intersection point's $x$ coordinate in the objects own coordinate system with the desired height of the segment was added as an additional criterion. This is first done for the first intersection point along the traced ray. If it is within the desired boundaries, the point is used for all further calculations, if it is outside the boundaries the same test is done for the second potential intersection point.

Scenes for Mitsuba are described using xml-files with a well-documented and consistent scripting language. All objects, including cameras, are generated, placed and specified independent of all other objects in the scene, and it is possible to include other scenes using the command of the same name. This allows for a modular construction of multiple scenes, where common elements can be kept in their own scene files. That way, changes to these elements can be done swiftly by changing their respective files instead of making the changes in every scene.

[1] A physical renderer is a type of programme that generates images with ray tracing or similar techniques and relies as little as possible on estimations and short cuts during its calculations.

### 2.2.1 Construction and validation of the simulation

### 2.2.2 Reconstructing the setup in the ray tracer

While Mitsuba offers a wide variety of ray tracing techniques to choose from, a simple path tracer was chosen for all simulations. Due to the simplicity of the scenes, other, more complex techniques were not seen as necessary, as they would have increased the computing time without improving the results on an equal level.
The simulation of the mirror was done using the altered sphere shape described above with a totally reflecting surface. In most cases, the radius or the sphere was $r = 32\,\mathrm{m}$, with the height of the segment $h \approx 0.00563\,\mathrm{m}$, resulting in a round mirror with base radius $a = 0.6\,\mathrm{m}$.
In addition to that, a plane was added that serves as a background to the images (which would instead be transparent everywhere but on the mirror).

## The cameras

For each of the cameras, one scene was created that contained all relevant information, that consists of the position and orientation, the field of view and the desired resolution, for which $1280x920$ was chosen in accordance to the resolution of the cameras used in the SWD setup.
The cameras position and orientation was defined using the 'lookat' command. This command contains three components, all of which are cartesian vectors that indicate points in the 3D space or directions:

- origin - the position of the camera,
- target - a point along the sight axis
- up - direction that fixes the orientation of the image around the sight axis

These parameters are correlated with the parameters of the geomery calibration, as 'origin' is identical to 'zerobase', 'target' was chosen as ('zerobase'+4·'ybase') and 'up' was found identical to 'zbase', which can be calculated from 'xbase' and 'ybase' via the cross product.
The field of view limits the area that is seen by the camera to a certain angle centered around the the line of sight. The concrete value depends on the characteristic size of the chip that is chosen for the calculation. The diagonal field of view was calculated using

$$2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right)$$

with the diagonal of the chip $d$, calculated using the pixel size and the resolution, and the focal length $f$ of each individual camera.

## The screen

The screen was simulated using two separate elements. The first one was a rectangular light source that was placed slightly behind the desired image plane. The image plane itself was created using a diffuse transmitting material, the transmittance of which was specified by textures that match the different stages of the phase shifting. Each individual stage had its own scene file, resulting in a total of 32 different scenes for the screen, as simulations were done for stretch factor 1 and stretch factor 24 to cover the whole process of phase shifting.
The textures were created using a simple programme to calculate the brightness of each pixel using the formula

$$b(x) = \lfloor 255 \cdot (0.5 + 0.5 \cdot \cos((x + 0.5) \cdot l \cdot f + \phi)) + 0.5 \rfloor$$

with $x$ being the number of the pixel, counting from 0 to 1920, $l$ being the length of a pixel and

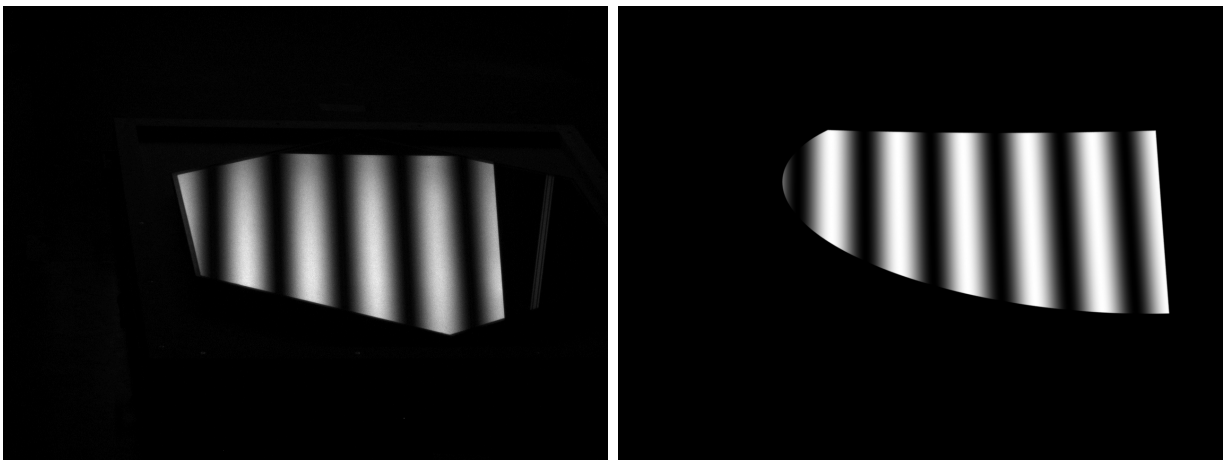
$f$ being the frequency

$$f = \frac{2 \cdot \pi \cdot SF \cdot n_{PQ}}{w}$$

with the stretch factor $SF$, $n_{PQ} \in \{P, Q\}$ with $P = 7$ and $Q = 8$ and the width $w$ (in metres) of the screen.

## Combination of the different elements

These 37 common resource scene files (32 different screens, 4 cameras and the mirror) were then combined in the 128 final scenes, that were used for the actual ray tracing.
Along with these scenes, the corresponding configuration files for *softPMD* files had to be created. All internal calibration parameters apart from the focal length for the cameras were set to 0, which corresponds to the assumption of both the cameras and the screen being perfect. All geometry parameters and the corresponding parameters for the simulation were calculated using an excel sheet.



(a) Image taken in the setup (b) Simulated Image

**Figure 10:** Comparison of an Image taken with camera 1 in the setup to a simulated image of the same camera

## Integrating the simulation in the measurement framework

The ray tracing generates the individual images taken during the phase shifting in a real measurement. A comparison between a real and simulated image can be seen in figure 10 as an example. These images are sorted by camera and stretch factor and each of these subsets is converted into a sequence, which is the format used by *softPMD* to save and work with the images during the measurement. *softPMD* is also capable of reading sequences provided by an external source and treat them equal to images taken during its own measurement. This was used to keep the simulation as close to the real measurements as possible (figure 11).
With these sequences replacing the real phase shifting by a simulated one, the projector coordinates and screen data are calculated as normal. The rest of the evaluations is the same as described in chapter 2.1.4 from there on.

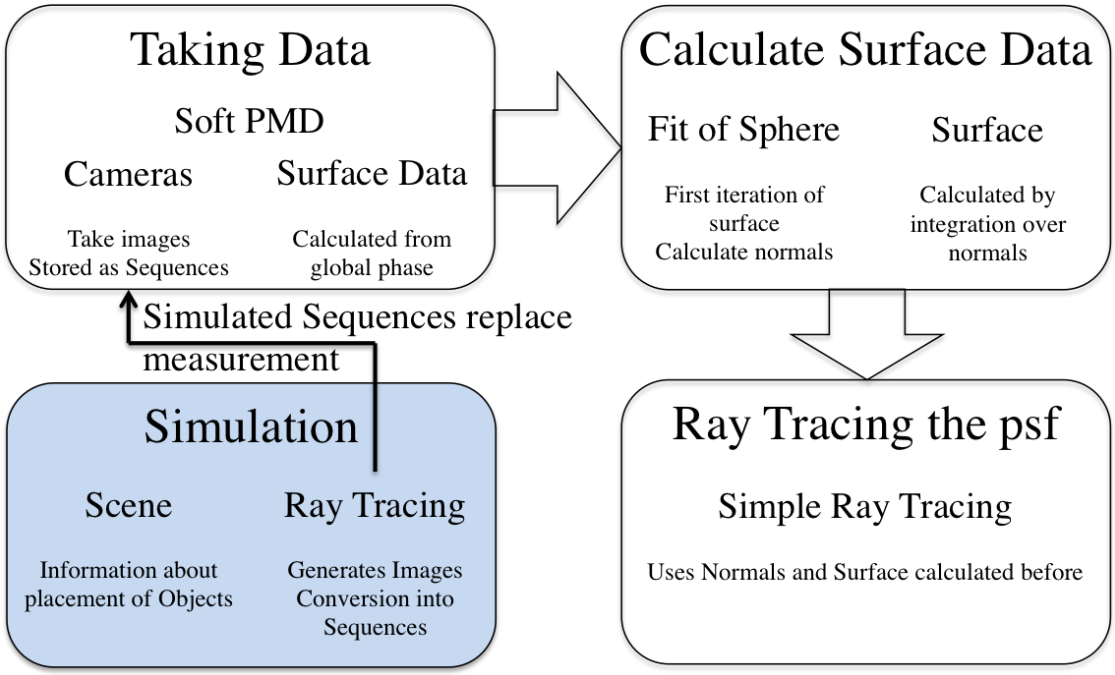


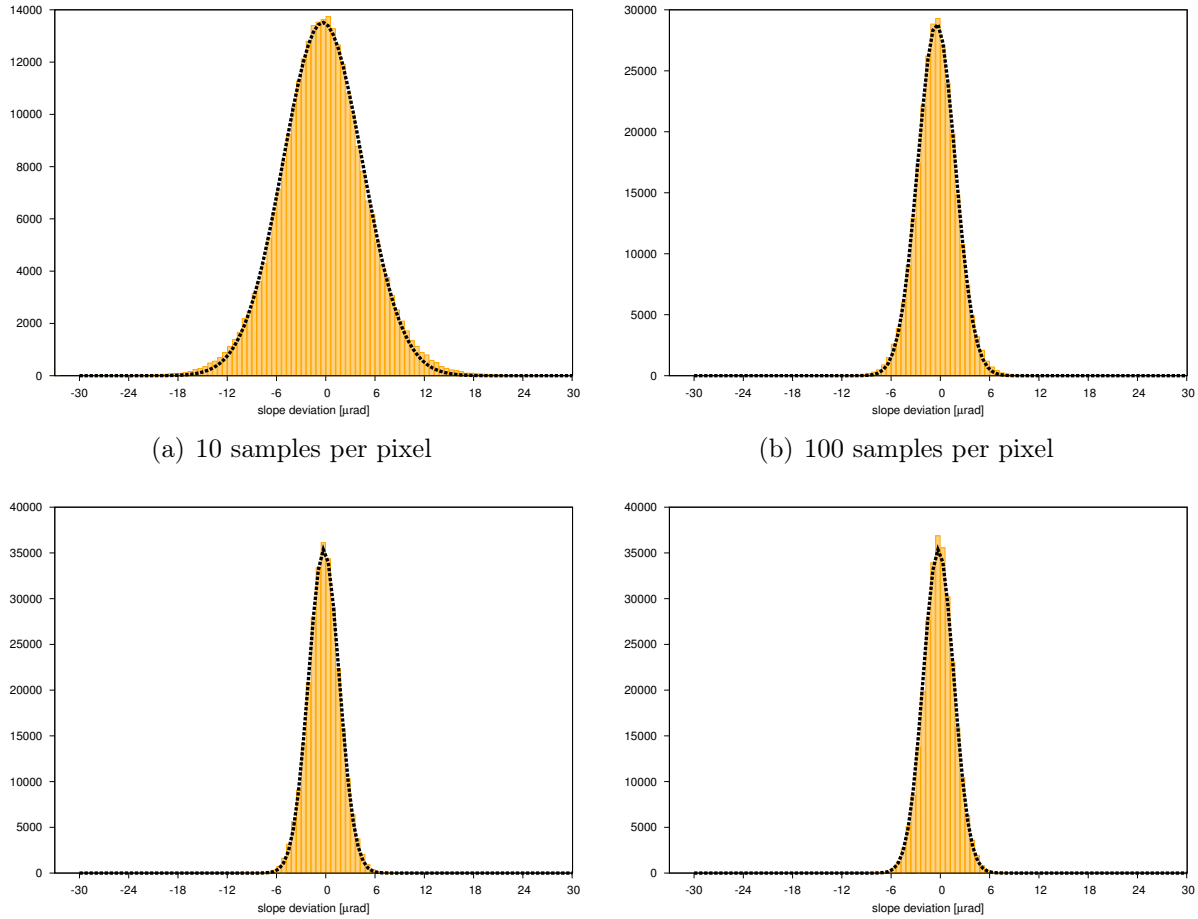

**Figure 11:** Scheme of the idea of the simulation.
Images created using ray tracing are transformed into sequences, a special data format that *softPMD* can read. These sequences are taken as input for *softPMD*, which treats them the same way it would treat images from real cameras. From this point, everything is done exactly as if it was a real measurement.

### 2.2.3 Choosing a reasonable number of samples

Due to the way the scene is simulated, noise effects can not be ruled out completely. A high number of samples per pixel, which means tracing more rays and averaging over the resulting color values, can be used to counter the effects of that noise, which may, however, result in long computing times. Therefore, a set of simulations with different sample counts was made to test for the effects this parameter has on the d80, the optimal radius of curvature and the noise on the surface data.

## Different noise on the $x$ and $y$ axes

Figure 12 and 13 show histograms of the x and y components of the slope deviation, which were used to quantify the noise on the acquired data of the surface, for the different numbers of samples per pixel that were simulated. The slope deviation of a real measurement (figure 14) is noticeable wider spread
The y component can be described by a normal distribution, with the width $\sigma_y$ being a measure for the noise level.
In case of the x component, the noise can not be described by a Gaussian distribution properly, but instead, the sum of two different normal distributions can reproduce the results of the simulations. With the width $\sigma_{x,1}$ roughly corresponding to $\sigma_y$, it is possible that this component of the noise has the same origin. At this point, the exact origin of the second component of the noise could not be identified, though it is likely an effect of the evaluation algorithm.
This second contribution to the $x$ component of the noise might be an explanation for the shape of the simulated PSF, which can be seen in figure 22 (a). The asymmetrical noise on the surface corresponds to a similarly asymmetric noise on the normals, which leads to a wider scattering of the reflected rays along the $x$ axis and thus a widening of the PSF.



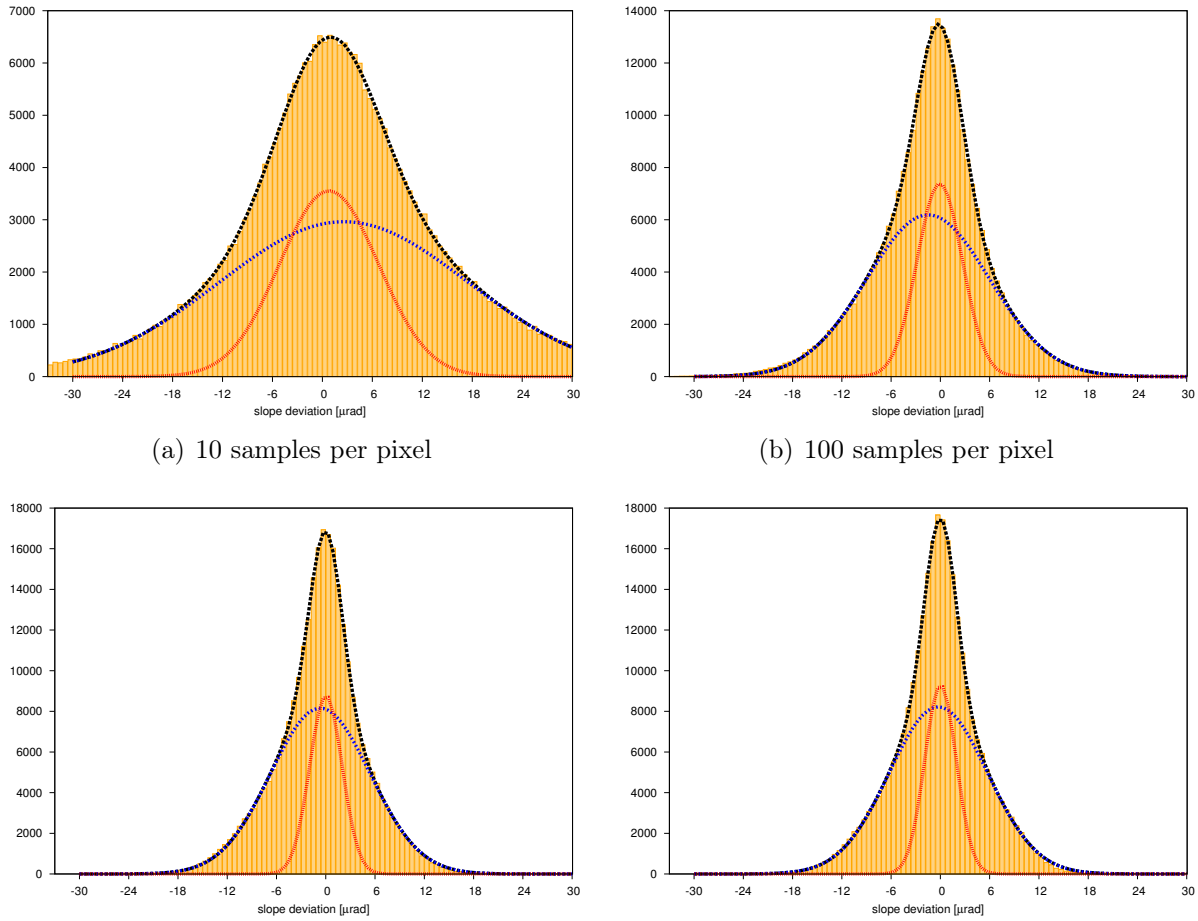

(a) 10 samples per pixel

(b) 100 samples per pixel

(c) 1000 samples per pixel

(d) 10000 samples per pixel

**Figure 12:** Histogram of the y component of the slope deviation




(a) 10 samples per pixel

(b) 100 samples per pixel

(c) 1000 samples per pixel

(d) 10000 samples per pixel

**Figure 13:** Histogram of the x component of the slope deviation
The black line is the sum of the red and blue Gaussian distributions.

## Effects of the noise

As can be seen in table 2, $\sigma$ is reduced by a factor of $\approx 2$ when using 100 instead of 10 samples. Further increasing the number of samples to 1000 or 10000 does not reduce the noise in the same amount, while the computing time is increased by a factor of 10 or 100 respectively.
The deviation of the mean values $x_{0,1}$, $x_{0,2}$ and $y_0$ from 0 can be explained by a global tilt of the surface to the coordinate system due to the noise when starting the integration process to calculate the exact surface. As with the total surface, this tilt has only a negligible effect on the ray tracing to calculate the PSF.
The d80 is influenced in the same way, with a steep step from 10 to 100 samples per pixel and less significant changes for more samples than that.
This leads to the conclusion that using 100 samples per pixel is the best compromise between computing time and noise reduction. All further studies were therefore done with 100 samples per pixel.

| | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| $\sigma_{x,1}$ [$\mu$rad] | $5.950 \pm 0.083$ | $2.743 \pm 0.032$ | $1.975 \pm 0.025$ | $1.913 \pm 0.021$ |
| $\sigma_{x,2}$ [$\mu$rad] | $15.011 \pm 0.187$ | $7.443 \pm 0.060$ | $6.051 \pm 0.046$ | $5.972 \pm 0.041$ |
| $\sigma_y$ [$\mu$rad] | $4.836 \pm 0.023$ | $2.275 \pm 0.012$ | $1.862 \pm 0.012$ | $1.815 \pm 0.013$ |
| $x_{0,1}$ [$\mu$rad] | $0.791 \pm 0.041$ | $-0.077 \pm 0.020$ | $0.018 \pm 0.016$ | $0.016 \pm 0.013$ |
| $x_{0,2}$ [$\mu$rad] | $2.563 \pm 0.093$ | $-1.492 \pm 0.044$ | $-0.671 \pm 0.031$ | $-0.250 \pm 0.026$ |
| $y_0$ [$\mu$rad] | $-0.373 \pm 0.023$ | $-0.461 \pm 0.012$ | $-0.231 \pm 0.012$ | $-0.181 \pm 0.013$ |
| d80 [mm] | 2.26 | 1.13 | 0.95 | 0.92 |
| r [m] | 32.00 | 32.00 | 32.00 | 32.00 |

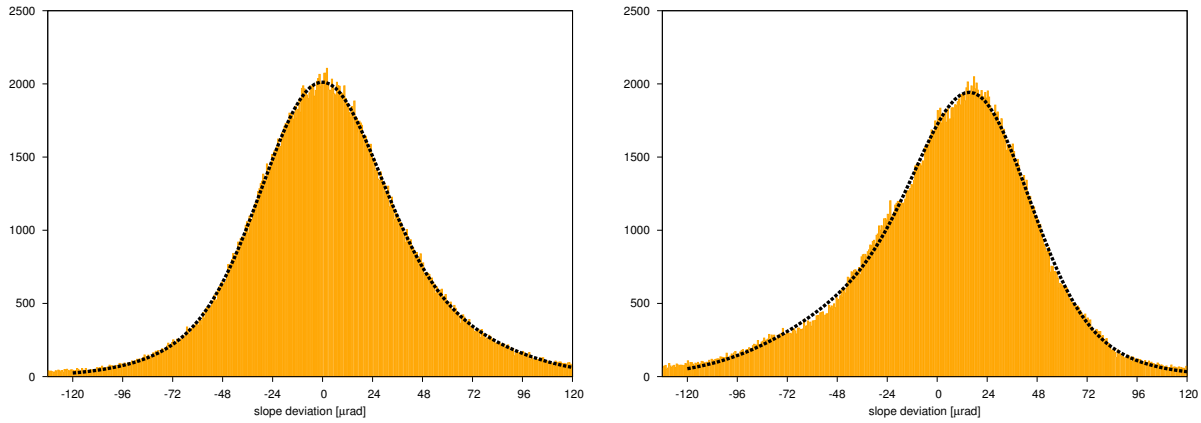
**Table 2:** Fit results, d80 and radius of curvature for the different numbers of samples

### 2.2.4 Comparison to a real measurement

Independent of the number of samples, the simulation leads to smaller deviations from a perfect sphere than a real measurement, as shown in figure 14. The slope deviation of the comparison measurement can, to some extent, be described by the sum of two Gaussians for both the $x$ and $y$ component, with

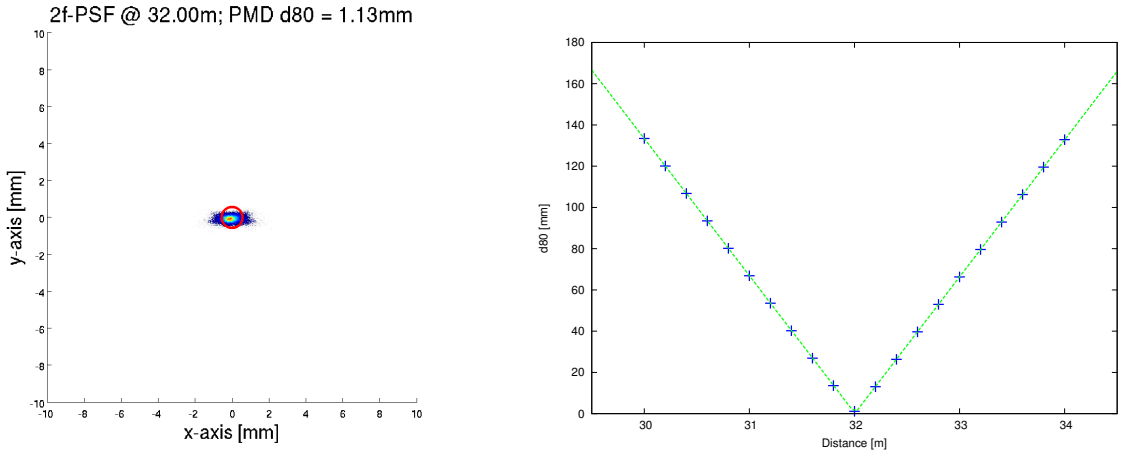
$$\begin{aligned}\sigma_{x,1} &= 50.651 \pm 0.884\\ \sigma_{x,2} &= 26.783 \pm 0.319\\ \sigma_{y,1} &= 50.082 \pm 0.587\\ \sigma_{y,2} &= 25.613 \pm 0.346.\end{aligned}$$

As indicated by the bigger errors, this does not properly describe the slope deviation of a real measurement, that does not only consist of noise but also includes information about real defects on the mirror surface.
The PSF that results from the simulation is very small compared to real measurements, as can be seen in figure 22 (a) and 16 (a), but still bigger than what was expected for the simulation of a perfect mirror. Taking a closer look at the simulated PSF, one can see a distinct widening along the $x$ axis compared to the $y$ axis, that also dominates the d80. One possible reason for this might be a different behaviour of the noise, as discussed in chapter 2.2.3, though this is not confirmed yet.



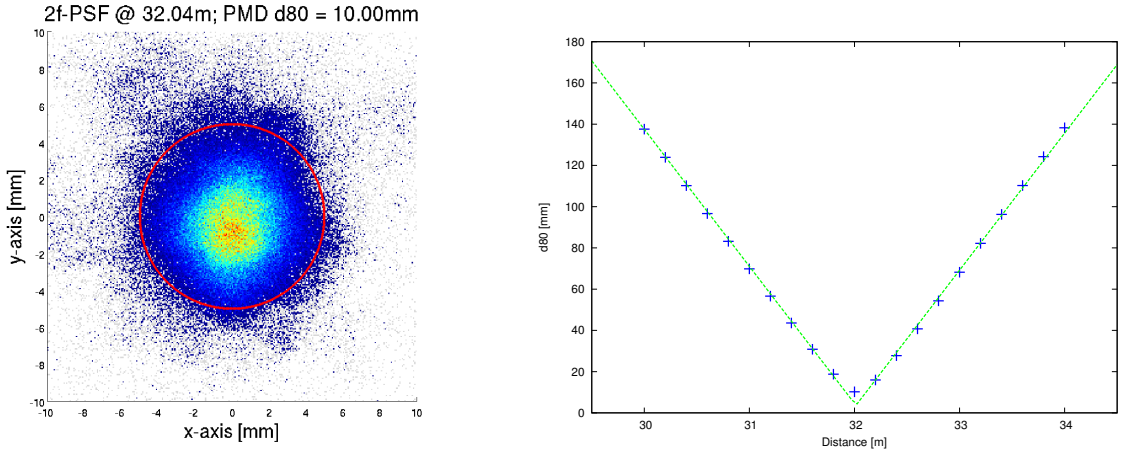

(a) $x$ component (b) $y$ component

**Figure 14:** Histogram of the slope deviation of a real measurement.
The black graphs show the sum of two Gaussian distribution fitted to the data.




(a) optimal PSF (b) d80 as a function of the distance from the mirror

**Figure 15:** Result of the simulation of the symmetrical setup




(a) optimal PSF (b) d80 as a function of the distance from the mirror

**Figure 16:** Results of a real measurement.
The d80 does not fully follow the linear trend close to the radius of curvature and is instead slightly larger than expected.

# 3 Results

## 3.1 Studies on a symmetrical Setup

Using the geometry parameters of the setup in Erlangen as reference, a symmetric setup was simulated.

The screen was rotation 135° around the $x$ axis and 0° around the other axes, its centre placed at $x = 0\,\text{m}$, $y = 1.34\,\text{m}$ $z = 1.10\,\text{m}$. The geometry parameters used to simulate the cameras are shown in table 3.

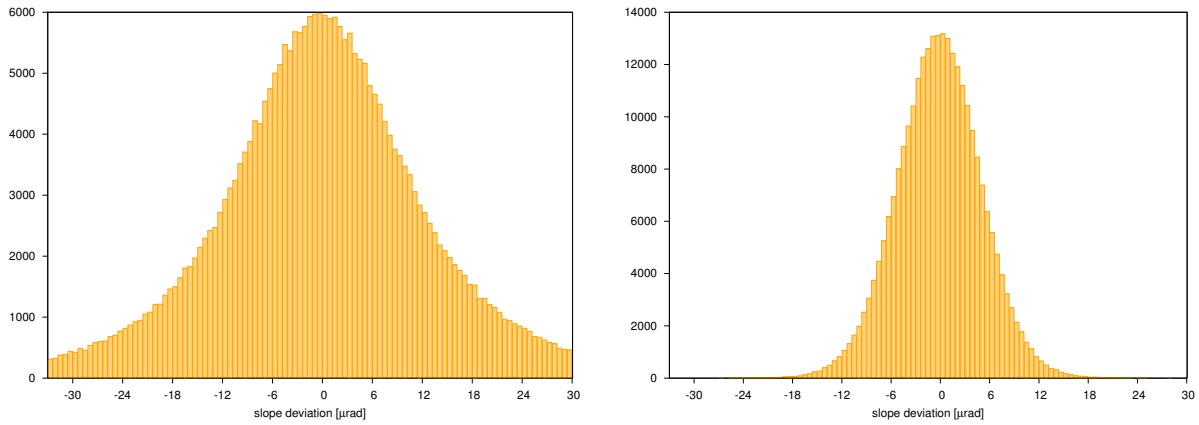
| | camera 0 | camera 1 | camera 2 | camera 3 |
|---|---|---|---|---|
| origin | −1.650 | −0.950 | 0.950 | 1.650 |
| | −2.850 | −4.000 | −4.000 | −2.850 |
| | 2.900 | 1.750 | 1.750 | 2.900 |
| target | −0.406 | −0.341 | 0.341 | 0.406 |
| | −0.064 | −0.334 | −0.334 | −0.064 |
| | 0.314 | 0.269 | 0.269 | 0.314 |
| up | −0.011 | −0.003 | 0.003 | 0.011 |
| | 0.683 | 0.375 | 0.375 | 0.683 |
| | 0.730 | 0.927 | 0.927 | 0.730 |

**Table 3:** Positions of the simulated cameras for the simulation of the setup in Erlangen

The simulation of this setup resulted in a minimal d80 = 1.13 mm at a distance of $r_{opt} = 32.00\,\text{m}$ for the mirror with $r = 32\,\text{m}$. The d80 increases linear with a slope of $(66.46 \pm 0.06) \cdot 10^{-3}$ when deviating from the optimal position, which can be seen in figure 15 (b).
These values are used as reference to determine the influence of different changes to the simulation in the following chapters.

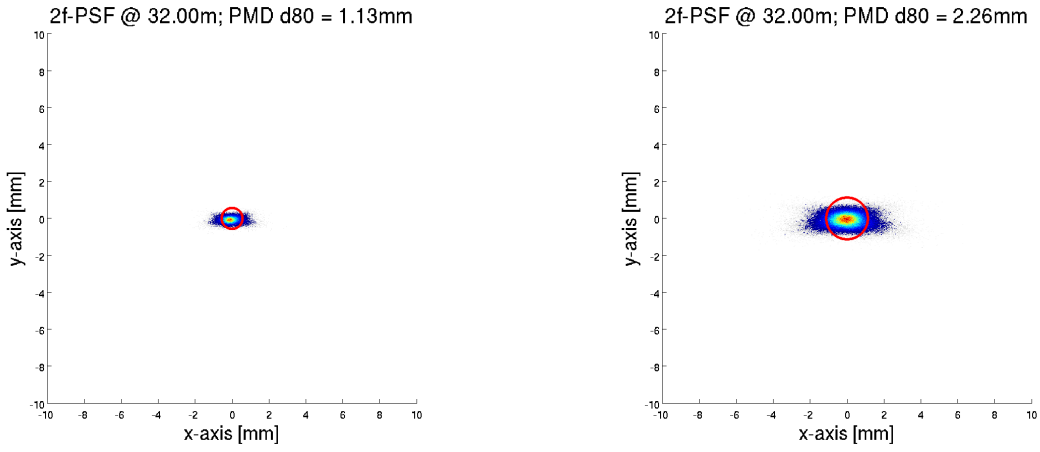
### 3.1.1 Influences of different camera resolutions

The influence of the camera resolution on the precision of the measurements is of interest for the design of possible new setups. Because of this a simulation with half the resolution of the current setup was performed.
This resulted in an increased noise level and a widened psf, which canbe seen in figure 17 and 18. Both are on a similar level to the simulations done with only 10 sample per pixel (compare chapter 2.2.3).
This clearly shows that the camera resolution can have a great influence on the results of the measurements and that using lower resolution cameras leads to less reliable results.
A simulation for cameras with higher resolution will be needed, however, to properly characterise this effect and decide if the higher cost of the corresponding cameras for new setups is justified by a sufficiently increased precision of the measurement results.




(a) $x$ component (b) $y$ component

**Figure 17:** Histogram of slope deviation when using a camera with halved resolution




(a) Normal resolution (b) Halved resolution

**Figure 18:** Comparison of the PSF from the simulations reduced resolution of the cameras

### 3.1.2 Moving the Cameras

To test the boundaries of the recalibration, the displacement of a single camera up to 10 cm from its original position, without changing the corresponding geometry parameters, was tested. This displacement was simulated for both camera 1 and camera 3 and the simulations were evaluated twice, one time using the recalibration as described in chapter 2.1.4 and one time using a version of the evaluation programme without the recalibration.

## Moving camera 1

The first camera that was moved was camera 1, which is the lower camera on the left side of the setup, when facing towards the screen from the cameras.
The results of this are shown in table 4, while figure 19 shows a comparison of the d80 at the nominal distance that results when the camera positions are fixed during the evaluation and when the recalibration is used.
For small displacements $x_{displ} \leq 10^{-6}$ m, the only noticable effect is a widening of the d80, which is not completely corrected by the recalibration. For stronger displacements, the PSF is widened even more and a noticeable deviation of the optimal distance $r_{opt,nR}$ and $e_{opt,nR}$ both without and with the recalibration. As shown in figure 15, the d80 can increase fast when it is not at the optimal distance, which leads to a further increased d80 at the nominal distance in this case.
Not using the recalibration on the simulation with the camera moved 10 cm away from its position lead to a strong deviation of the radius of curvature outside of the borders set for the raytracing and thus no calculation of the d80. The recalibration could not compensate the faulty reconstruction of the radius of curvature and the resulting PSF was still very large with $\mathrm{d80}_{nim,R} = 128$ mm at the measured radius of curvature and $\mathrm{d80}_{nom,R} = 292.36$ mm at the nominal radius of curvature.

| $x_{displ}$ [m] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $\mathrm{d80}_{opt,nR}$ [mm] | $\mathrm{d80}_{opt,R}$ [mm] | $\mathrm{d80}_{nom,nR}$ [mm] | $\mathrm{d80}_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| $10^{-9}$ | 32.00 | 32.00 | 1.35 | 1.14 | 1.35 | 1.14 |
| $10^{-6}$ | 32.00 | 32.00 | 1.36 | 1.14 | 1.36 | 1.14 |
| $10^{-3}$ | 31.86 | 31.98 | 44.04 | 1.14 | 47.54 | 1.62 |
| $10^{-2}$ | 30.66 | 31.74 | 419.54 | 1.66 | 474.52 | 16.24 |
| $10^{-1}$ | 29.80 | 29.80 | *N.A.* | 128.00 | *N.A.* | 292.36 |

**Table 4:** Results for moving camera 1.
Camera 1 was moved by $x_{displ}$ along the x axis. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $\mathrm{d80}_{opt,nR}$ and $\mathrm{d80}_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $\mathrm{d80}_{nom,nR}$ and $\mathrm{d80}_{nom,R}$.

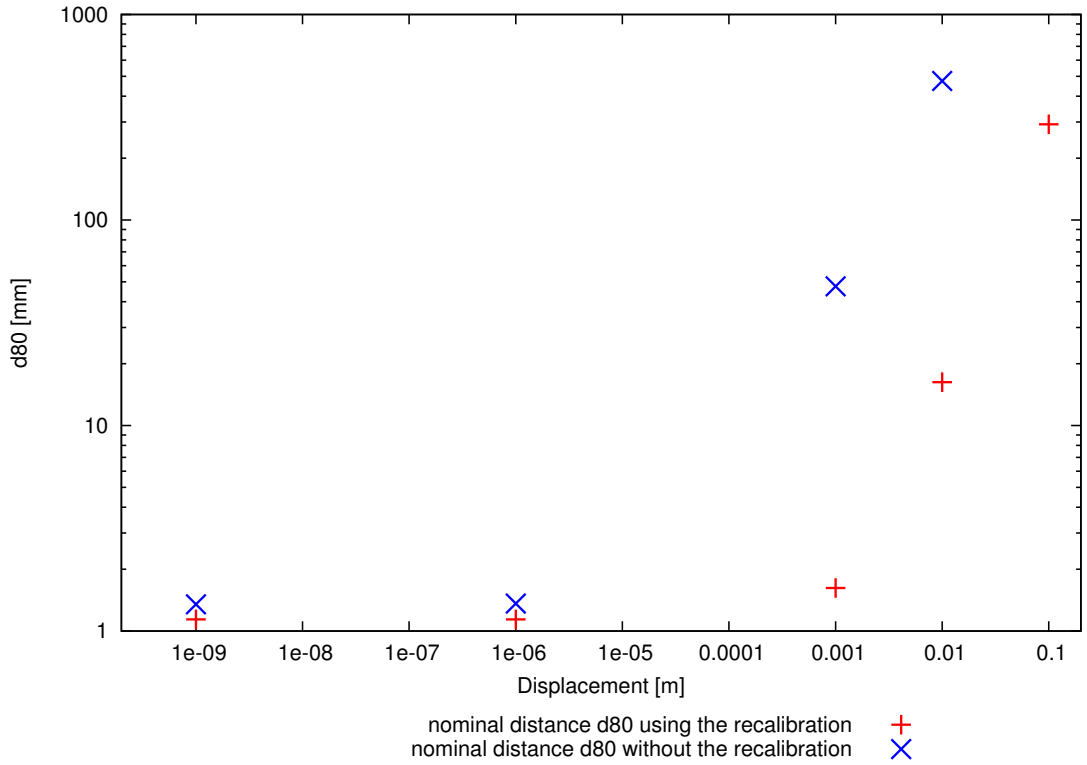



**Figure 19:** d80 at the nominal radius of curvature as a function of the displacement distance for camera 1.
For the strongest displacement, the d80 could not be calculated without the recalibration.

## Moving camera 3

Camera 3, the higher camera on the right side of the setup, was then moved in a similar way. The effects on the reconstructed radius of curvature and the d80 were mostly similar, though the PSF was not widened as strong as it was when moving camera 1, as can be seen in table 5. This might be related to the fact, that camera 3 covers a smaller area of the mirror and therefore only has a smaller influence on the end result.

| $x_{displ}$ [m] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $d80_{opt,nR}$ [mm] | $d80_{opt,R}$ [mm] | $d80_{nom,nR}$ [mm] | $d80_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| $10^{-9}$ | 32.00 | 32.00 | 1.35 | 1.13 | 1.35 | 1.13 |
| $10^{-6}$ | 32.00 | 32.00 | 1.34 | 1.14 | 1.34 | 1.14 |
| $10^{-3}$ | 31.88 | 32.02 | 20.00 | 1.12 | 20.76 | 1.58 |
| $10^{-2}$ | 30.52 | 32.14 | 194.31 | 1.28 | 209.05 | 8.95 |
| $10^{-1}$ | 29.80 | 34.20 | *N.A.* | 18.64 | *N.A.* | 152.29 |

**Table 5:** Results for moving camera 3.
Camera 3 was moved by $x_{displ}$ along the x axis. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $d80_{opt,nR}$ and $d80_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $d80_{nom,nR}$ and $d80_{nom,R}$.

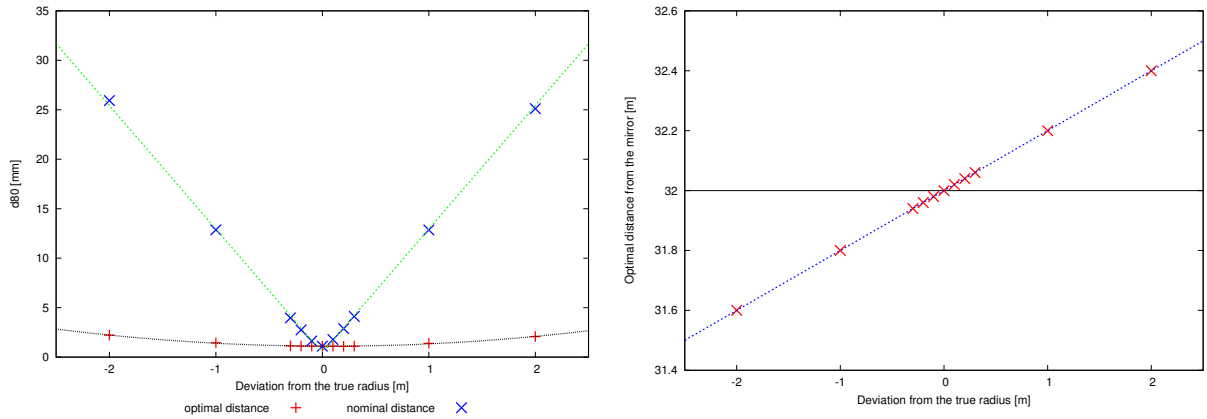
Notably in both cases is the fact that the recalibration did not move the camera to the place it actually was at but instead made changes to all cameras and in all directions. The geometry parameters resulting from the recalibration can be found in the appendix.
Cameras that are displaced by more than 1 mm from their position in the calibration data were found to be the limit of the recalibration. Bigger deviations from the calibrated state cause the recalibration to misplace all cameras and lead to unreliable results.

### 3.1.3 Influence of incorrect reference measurements

To compensate minor deviations from the calibrated state, reference measurements of a mirror with known radius of curvature are taken and used for the recalibration. This increases the reliability of the recalibration, but it also adds another source of uncertainty to the measurement. If the reference mirror has a different radius of curvature than what it is believed to have, which can be caused by a different mounting or orientation compared to its initial measurement and a deformation under its own weight, it causes new errors during the evaluation of the measurements.
The influence an incorrect reference measurement can have on the results of the actual measurements has been tested using a set of simulations with different radii of curvature forthe mirrors that were measured.
A H.E.S.S. II mirror with radius of curvature $r \approx 72\,\mathrm{m}$ was chosen as reference mirror for the setup in Erlangen, corresponding to that, a simulation with a similar mirror was created as reference.

## Results for the mirror with $r = 32\,\mathrm{m}$

Using the simulation of the mirror with $r = 72\dot{\mathrm{m}}$ as reference, but with multiple false values for the radius of curvature, the simulation for the mirror with $r = 32\,\mathrm{m}$ was evaluated.
Using a false reference radius leads to a faulty calculation of the sphere radius during the evaluation of the actual simulation or measurement and subsequently to a wrong reconstruction of the surface data. The deviation of the optimal distance from the nominal radius of curvature is linear in the deviation from the reference radius. The slope of the best fit straight line is $0.200 \pm 0.001$ for the evaluation of the $r = 32$ mirror with the $r = 72$ mirror as reference.
Due to the PSF widening fast when moving away from the optimal point, this leads to incorrect results for the $\mathrm{d80}_{nom}$ at the nominal point.
The faulty surface data also causes the optimal PSF to be widened and deformed, but this effect does not contribute to the error on $\mathrm{d80}_{nom}$ to the same extent.




(a) d80 of the $r = 32$ mirror as a function of the deviation from the true radius of curvature

(b) Optimal distance from the $r = 32$ mirror as a function of the deviation of the reference radius

**Figure 20:** Result of the simulation of the symmetrical setup.
While the d80 at the measured radius of curvature stays relatively small, it deviates away from the nominal distance which leads to a widened PSF there, as can be seen on the left side.
This deviation can be seen on the right side, along with the best fit staight line to these data points.

## Comparing other simulated mirrors

In a similar way to the mirror with $r = 32$, other simulation with different radii were evaluated using the wrong references. The results were very similar, so no in depth discussion of each simulated mirror is necessary.
For each of the simulated mirrors, the optimal distance deviates linear with the error on the reference radius, and the deviation slope $s$ of the best fit straight lines are shown in table 6.
To find a functional correlation, another fit of a straight line was performed to these slopes as

| radius $R$ | slope $s$ |
|---|---|
| 30 | $0.170 \pm 0.003$ |
| 32 | $0.200 \pm 0.001$ |
| 33 | $0.219 \pm 0.001$ |
| 50 | $0.581 \pm 0.001$ |
| 72 | 1 |

**Table 6:** Resulting slopes $s$ for the different radii $R$.
Note that the last point is fixed, as it is derived from the reference measurement itself

a function of the radius of the mirrors, which is shown in figure 21. The resulting slope was $0.020 \pm 0.001$ with an offset of $-0.433 \pm 0.015$. This does not describe the data properly, which is why another fit with a polynomial:

$$s(R) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$$

with the parameters $a = (-7.44 \pm 1.137) \cdot 10^{-6}$, $b = (1.096 \pm 0.174) \cdot 10^{-3}$, $c = -0.031 \pm 0.009$, $d = 0.305 \pm 0.127$. Neither of these are based on a physical model. Additional data points, especially in the range between 50 and 72 and above 72, are necessary to further test the correlation of the deviation slope and the radius of the mirror, especially to test for the behaviour when evaluating mirrors with radii of curvature that are greater than the radius of curvature of the reference mirror itself.
Cross references with other simulations used as reference could further improve the understanding of this correlation and may lead to the development of an universally usable rule to estimate the influence of false reference radii before any measurements have to be performed.

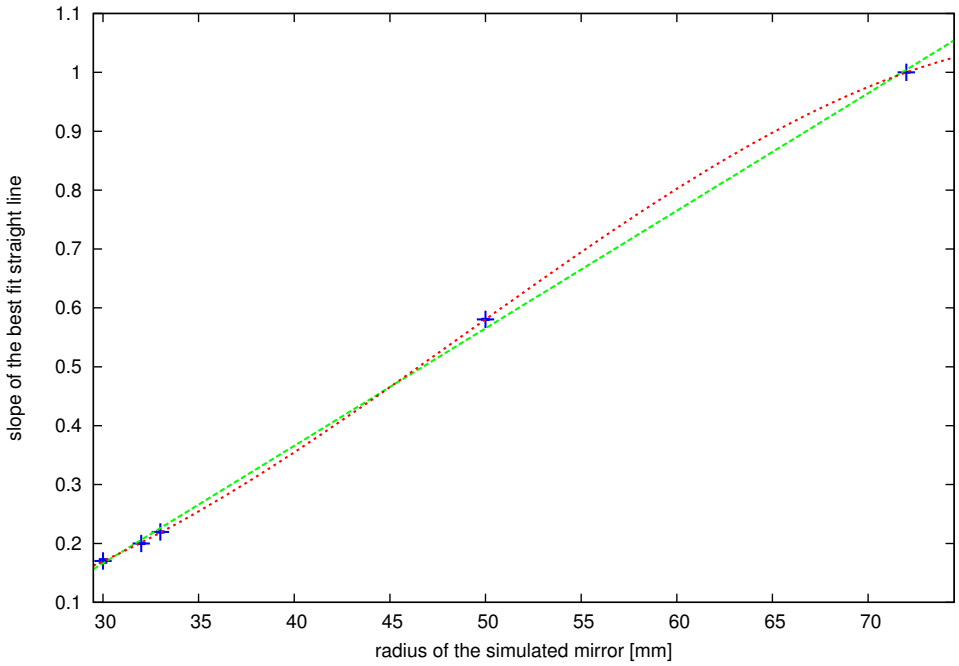



**Figure 21:** Correlation of the slope with the radius of curvature

## 3.2 Simulations of the existing Setup in Erlangen

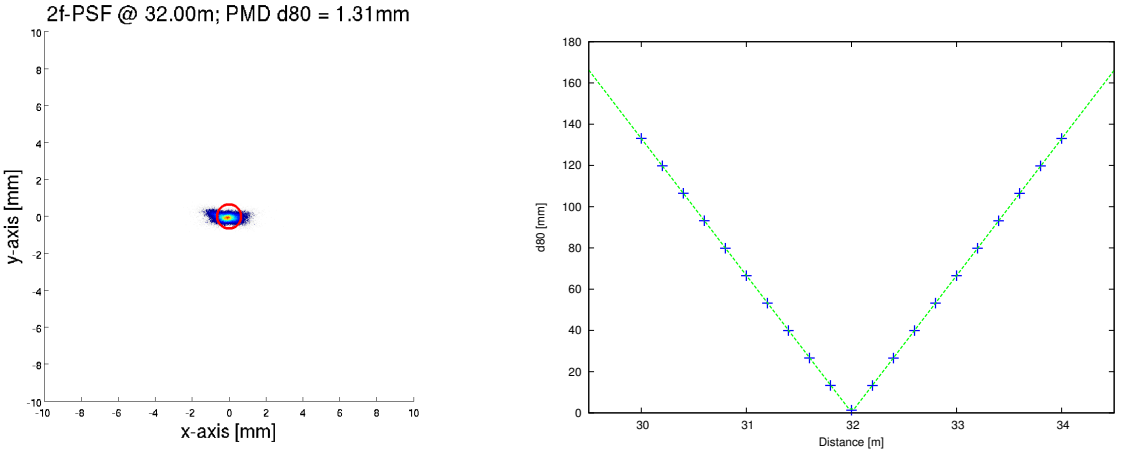
Using the calibration data of the SWD setup in Erlangen, it was reconstructed identically for the simulations. The geometry parameters of the configuration files files were translated directly to the corresponding parameters for the simulation.
The screen was rotation $-0.198°$ around the $z$ axis, $1.268°$ around the $y$ axis and $136.708°$ around the $x$ axis, its centre placed at $x = 0.005\,\mathrm{m}$, $y = 1.345\,\mathrm{m}$ $z = 1.102\,\mathrm{m}$. The geometry parameters used to simulate the cameras are shown in table 7.

| | camera 0 | camera 1 | camera 2 | camera 3 |
|---|---|---|---|---|
| | −1.576 | −0.778 | 1.113 | 1.741 |
| origin | −2.872 | −4.057 | −4.005 | −2.674 |
| | 2.910 | 1.751 | 1.750 | 2.965 |
| | −0.414 | −0.164 | 0.406 | 0.498 |
| target | 0.087 | −0.396 | −0.350 | 0.136 |
| | 0.482 | 0.262 | 0.287 | 0.404 |
| | −0.002 | 0.027 | −0.012 | 0.015 |
| up | 0.635 | 0.381 | 0.370 | 0.677 |
| | 0.773 | 0.924 | 0.929 | 0.736 |

**Table 7:** Positions of the simulated cameras for the simulation of the setup in Erlangen

The results for this simulation without any deliberate errors were very similar to the symmetrical setup, with a slightly larger minimal d80 = 1.31 mm at a distance of $r_{opt} = 32.00\,\mathrm{m}$ for the

simulated mirror with $r = 32\,\mathrm{m}$ and a slope of $(66.39 \pm 0.10) \cdot 10^{-3}$ when deviating from there.




(a) optimal PSF

(b) d80 as a function of the distance from the mirror

**Figure 22:** Result of the simulation of the reconstructed setup

Both the influence of a tilt and deformation of the screen that was not accounted for in the calibration were tested on this setup.

### 3.2.1 Tilt of the Screen

A linear deviation from the calibrated state, which can be seen as a first order error, was simulated by slightly rotating the simulated screen from the original position up to $\theta = 1°$ to test the capabilities of the recalibration. This tilt was simulated for all three axes of the screen's coordinate system individually, while no changes to the geometry parameters in the configuration files were made.

## Rotation around the $x$ axis

For small tilts around the $x$ axis, the only effect that could be seen was a widened PSF that could be reduced slightly by the recalibration.
The reconstructed radius of curvature was reconstructed correctly even without the recalibrations for all small tilts below $\theta_x = 0.1°$, where it actually caused a small deviation itself. For the maximum rotationthat was tested, the existing deviation was enlarged by the recalibration, but the resulting PSF was greatly reduced in size.

| $\theta_x$ [°] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $d80_{opt,nR}$ [mm] | $d80_{opt,R}$ [mm] | $d80_{nom,nR}$ [mm] | $d80_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| $10^{-2}$ | 32.00 | 32.00 | 1.32 | 1.28 | 1.32 | 1.28 |
| $10^{-1}$ | 32.00 | 31.98 | 8.10 | 1.40 | 8.10 | 2.27 |
| $10^{-0}$ | 31.94 | 31.70 | 80.63 | 2.73 | 80.91 | 19.25 |

**Table 8:** Results for rotating the screen around the $x$ axis.
The screen was rotated $\theta_x$ around the $x$ axis. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $d80_{opt,nR}$ and $d80_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $d80_{nom,nR}$ and $d80_{nom,R}$.

## Rotation around the $y$ axis

The effect on both the optimal distance from the mirror and the d80 is stronger for rotations around the $y$ axis compared to the tilt around the $x$ axis.
Like before, the recalibration could not properly compensate the deviation of the radius of curvature and in fact enlarged it again for the strongest rotation. The resulting PSFs, however, were small compared to the ones that resulted from the evaluation without recalibration.

## Rotation around the $z$ axis

The evaluation of the simulations with the screen rotated around the $z$ axis do not show any new effects, though it is notable that the recalibration slightly overcopmpensated the tilt in this case, but did not cause a larger deviation in the radius of curvature compared to the results without recalibration.

Like during the simulations with displaced cameras, the recalibration did show some flaws during this study, notably the overcompensation or actual causation of deviations in the radius of curvature. In general, it was found to be able to correct a tilt of up to 0.01° reliably, but stronger effects count not be compensated properly.

| $\theta_y$ [°] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $d80_{opt,nR}$ [mm] | $d80_{opt,R}$ [mm] | $d80_{nom,nR}$ [mm] | $d80_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| $10^{-2}$ | 32.00 | 32.00 | 1.60 | 1.30 | 1.60 | 1.30 |
| $10^{-1}$ | 32.02 | 32.02 | 11.32 | 1.37 | 11.55 | 2.14 |
| $10^{-0}$ | 32.22 | 32.28 | 113.14 | 1.84 | 114.77 | 18.32 |

**Table 9:** Results for rotating the screen around the $y$ axis.
The screen was rotated $\theta_y$ around the $y$ axis. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $d80_{opt,nR}$ and $d80_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $d80_{nom,nR}$ and $d80_{nom,R}$.

| $\theta_z$ [°] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $d80_{opt,nR}$ [mm] | $d80_{opt,R}$ [mm] | $d80_{nom,nR}$ [mm] | $d80_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| $10^{-2}$ | 32.00 | 32.00 | 4.20 | 1.30 | 4.20 | 1.30 |
| $10^{-1}$ | 32.00 | 32.00 | 40.73 | 1.39 | 40.73 | 1.39 |
| $10^{-0}$ | 31.30 | 32.04 | 402.30 | 1.29 | 406.73 | 2.98 |

**Table 10:** Results for rotating the screen around the $z$ axis.
The screen was rotated $\theta_z$ around the $y$ axis. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $d80_{opt,nR}$ and $d80_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $d80_{nom,nR}$ and $d80_{nom,R}$.

The resulting geometry parameters of the recalibration can be found in the appendix.

### 3.2.2 Deformation of the Screen

As the surface of the screen is described by polynomials, the calibration can differ from its actual shape, especially at the borders where the polynomial extrapolates from the pattern that was used instead of interpolating between its tags. Due to their size, the measurements of the mirrors often only cover the whole surface when the data from these areas is used as well instead of discarding it.
The weight of the screen of the setup in Erlangen also causes a strong deformation, with differences of some mm between the deepest and highest areas. In the worst case, the deviation from the calibration could be of a similar amplitude.
By replacing the diffuse transmitting plane that serves as the surface of the screen in the simulation with a height field of the same size, these deviations were simulated using six different patterns.
These patterns were created in a similar way to the images that had to be displayed by the screen described in chapter 2.2.1, though only one half term of the sine function was used in either direction. For the first two patterns the maxima of the half term were aligned with the borders of the screen, leading to protruding left and right or upper and lower borders. Using the sum over the previous two, the third pattern made the whole border of the screen, and especially the corners, stand out.
Similarly, the other three patterns were created by aligning the minima of the half term of the sinosuidal function with the borders, creating a bulge along either of the screens axes and the centre of the screen, which is very reminiscent of a feature of the real screens' deformation.
Each of the patterns has been used for simulations with three different amplitudes, $1 \cdot 10^{-9}$ m, $1 \cdot 10^{-6}$ m and $1 \cdot 10^{-3}$ m.

#### Raised borders

The first set of simulations added a deviation from the calibrated state at the borders of the screen. The most notable effect is a decreased d80 for small amplitudes below $10^{-6}$ m and especially a slight decrease in of the d80 when the amplitude of the combined pattern is increased. The deformation of the screen might counter the effect that widenes the PSF along the $x$ axis and thus result ina smaller overall d80. The recalibration actually increases the d80 for these small deformations.
The deviation of the radius of curvature, that only occured for the strongest deformation that was simulated, could not be compensated properly by the recalibration.

#### Bulged centre

The overall behavior of the second set of simulations with bulges along the axes or in the centre of the screen was similar to the first set of deformations that were tested, as can be seen in table 12.

The recalibration being unable to properly compensate the deformation of the screen was not unexpected, as the parameters that would have to be changed can not be accessed.
The reduction of the d80 for small deformation, however, was not expected. More different patterns may have to be tested to fully understand this effect, as well as more data points for each individual pattern.

| $h$ [m] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | $d80_{opt,nR}$ [mm] | $d80_{opt,R}$ [mm] | $d80_{nom,nR}$ [mm] | $d80_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| Raised left and right borders | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.11 | 1.29 | 1.11 | 1.29 |
| $10^{-6}$ | 32.00 | 32.00 | 1.11 | 1.29 | 1.11 | 1.29 |
| $10^{-3}$ | 32.04 | 31.98 | 4.33 | 1.29 | 5.47 | 4.57 |
| Raised upper and lower borders | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.11 | 1.27 | 1.11 | 1.27 |
| $10^{-6}$ | 32.00 | 32.00 | 1.11 | 1.27 | 1.11 | 1.27 |
| $10^{-3}$ | 32.10 | 32.04 | 9.92 | 4.21 | 12.87 | 5.37 |
| Combination with completely raised border and corners | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.11 | 1.28 | 1.11 | 1.28 |
| $10^{-6}$ | 32.00 | 32.00 | 1.10 | 1.29 | 1.10 | 1.29 |
| $10^{-3}$ | 32.04 | 32.02 | 4.49 | 2.86 | 5.57 | 3.01 |

**Table 11:** Results for the deformation of the screen with raised borders.
The height of the $h$ pattern gives the maximum height distance on the surface of the deformed screen. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. $d80_{opt,nR}$ and $d80_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, $d80_{nom,nR}$ and $d80_{nom,R}$.

| $h$ [m] | $r_{opt,nR}$ [m] | $r_{opt,R}$ [m] | d80$_{opt,nR}$ [mm] | d80$_{opt,R}$ [mm] | d80$_{nom,nR}$ [mm] | d80$_{nom,R}$ [mm] |
|---|---|---|---|---|---|---|
| Bulge along the $x$ axis | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.11 | 1.32 | 1.11 | 1.32 |
| $10^{-6}$ | 32.00 | 32.00 | 1.11 | 1.29 | 1.11 | 1.29 |
| $10^{-3}$ | 32.12 | 32.00 | 10.44 | 3.99 | 13.70 | 3.99 |
| Bulge along the $y$ axis | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.11 | 1.30 | 1.11 | 1.30 |
| $10^{-6}$ | 32.00 | 32.00 | 1.11 | 1.28 | 1.11 | 1.28 |
| $10^{-3}$ | 32.18 | 32.12 | 11.75 | 5.39 | 17.65 | 9.51 |
| Combination with bulge in the centre | | | | | | |
| $10^{-9}$ | 32.00 | 32.00 | 1.10 | 1.33 | 1.10 | 1.33 |
| $10^{-6}$ | 32.00 | 32.00 | 1.12 | 1.36 | 1.12 | 1.36 |
| $10^{-3}$ | 32.10 | 32.04 | 8.42 | 2.26 | 11.95 | 4.54 |

**Table 12:** Results for the deformation of the screen with a central bulge.
The height of the $h$ pattern gives the maximum height distance on the surface of the deformed screen. $r_{opt,nR}$ and $r_{opt,R}$ are the two optimal distances derived without and with the recalibration respectively. d80$_{opt,nR}$ and d80$_{opt,R}$ are the optimal values of d80, again without and with the recalibration. The same applies for the d80 at the nominal distance, d80$_{nom,nR}$ and d80$_{nom,R}$.

# 4 Conclusion and outlook

Using ray tracing to simulate the images that would usually be taken by the cameras of the measurement setup, the measurement method PMD, in combination with the current methods of evaluation, was tested regarding their capability to measure mirror facets for CTA.
Assuming a properly calibrated setup, the method was found to be suitable for measuring the mirrors with the required precision.
Over the course of this master thesis, a foundation for executing additional studies on potential sources of errors in a PMD setup.

Multiple studies of potential sources for errors were performed and used to test the limits of the recalibration performed during the evaluation of the measured data. It was found to be an unreliable tool on its own, as it made changes to the geometry of the whole setup to compensate small errors in only one of its components.

To prevent this behaviour, the recalibration can be preformed on a reference measurement of a mirror with a known radius of curvature. The effect of a false assumption for this radius of curvature was tested and was shown to have a great effect on the results.

The studies done for tor this master thesis systematically investigated some potential sources for errors. As it is unlikely for any of the tested errors to occur on their own, their interactions with each other as well as the recalibration, either with or without reference measurement, will have to be tested too.
The combination of deliberate errors and the reference method is especially important to ensure that it is sufficient to prevent the overcompensation of geometry errors.
Other potential future studies include tests of the internal calibration of the screen and the geometry calibration of the setup, which can be performed with minor modifications to the existing scene files.

The simulation was also used outside of the original intention to test the field of view and find the optimal positions for the individual cameras of a new setup planned to be constructed at the manufacturing place of the mirrors.

# Appendix

## The spherical segment class

This class was used to simulate the mirror. It is based on the sphere class included in mitsuba with only minor modifications.
The bounding box AABB was modified to only include the part of the sphere that is needed. An additional check if either of the intersection points are within the desired range was added by transforming them into the coordinate system of the object and compare their $x$ components with the desired borders of the mirror.

```
/*
    This file is part of Mitsuba, a physically based rendering system.

    Copyright (c) 2007-2012 by Wenzel Jakob and others.

    Mitsuba is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License Version 3
    as published by the Free Software Foundation.

    Mitsuba is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

#include <mitsuba/render/shape.h>
#include <mitsuba/render/bsdf.h>
#include <mitsuba/render/emitter.h>
#include <mitsuba/render/sensor.h>
#include <mitsuba/render/subsurface.h>
#include <mitsuba/render/trimesh.h>
#include <mitsuba/render/medium.h>
#include <mitsuba/core/properties.h>
#include <mitsuba/core/warp.h>/

MTS_NAMESPACE_BEGIN

class Sphericalsegment : public Shape {
public:
 Sphericalsegment(const Properties &props) : Shape(props) {
  m_objectToWorld =
  Transform::translate(Vector(props.getPoint("center", Point(0.0f))));
  m_radius = props.getFloat("radius", 1.0f);
  m_height1 = props.getFloat("height1", m_radius);        //
  m_height2 = props.getFloat("height2", 0.0f);          //

  if (props.hasProperty("toWorld")) {
   Transform objectToWorld = props.getTransform("toWorld");
   Float radius = objectToWorld(Vector(1,0,0)).length();
   // Remove the scale from the object-to-world transform
   m_objectToWorld =
   objectToWorld
   * Transform::scale(Vector(1/radius))
   * m_objectToWorld;
```

```
    m_radius *= radius;
49  }

51  /// Are the sphere normals pointing inwards? default: no
    m_flipNormals = props.getBoolean("flipNormals", false);
53  m_center = m_objectToWorld(Point(0,0,0));
    m_worldToObject = m_objectToWorld.inverse();
55  m_invSurfaceArea = 1/(4*M_PI*m_radius*m_radius);

57  if (m_radius <= 0)
     Log(EError, "Cannot create spheres of radius <= 0");
59  if (m_radius <= m_height1)
     Log(EError, "height1 cannot be equal or greater than radius");
61  if (m_height1 <= m_height2)
     Log(EError, "height2 cannot be equal or greater than height1");
63  if (m_height1 <= 0 || m_height1 <= 0)
     Log(EError, "Cannot create shperical segments with height-values <= 0");
65 }

67


69 Sphericalsegment(Stream *stream, InstanceManager *manager)
   : Shape(stream, manager) {
71  m_objectToWorld = Transform(stream);
    m_radius = stream->readFloat();
73  m_height1 = stream->readFloat();            //
    m_height2 = stream->readFloat();            //
75  m_center = Point(stream);
    m_flipNormals = stream->readBool();
77  m_worldToObject = m_objectToWorld.inverse();
    m_invSurfaceArea = 1/(4*M_PI*m_radius*m_radius);
79 }

81 void serialize(Stream *stream, InstanceManager *manager) const {
    Shape::serialize(stream, manager);
83  m_objectToWorld.serialize(stream);
    stream->writeFloat(m_radius);
85  m_center.serialize(stream);
    stream->writeBool(m_flipNormals);
87 }

89


91 AABB getAABB() const {
    float maxradiush;
93  float maxradiusl;
    if (-m_radius + m_height1 >0 && -m_radius + m_height2 < 0) {
95   maxradiush = m_radius;
     maxradiusl = m_radius;
97  }
    else {
99   maxradiush = sqrt(2*m_height1*m_radius - m_height1*m_height1);
     maxradiusl = sqrt(2*m_height2*m_radius - m_height2*m_height2);
101 }
    AABB aabb;
103

    //cut on x-axis instead of z-axis
105 aabb.expandBy(m_objectToWorld(Point(-m_radius + m_height2, -maxradiusl, -
      maxradiusl)));
    aabb.expandBy(m_objectToWorld(Point(-m_radius + m_height2, maxradiusl,
```

```
    maxradiusl)));
107
   aabb.expandBy(m_objectToWorld(Point(-m_radius + m_height1, -maxradiush, -
     maxradiush)));
109  aabb.expandBy(m_objectToWorld(Point(-m_radius + m_height1, maxradiush,
     maxradiush)));

111
//   aabb.min = m_center - Vector(m_radius);  //original boundig box of the
    sphere pligin
113 //   aabb.max = m_center + Vector(m_radius);
   return aabb;
115 }

117 Float getSurfaceArea() const {
   return 4*M_PI*m_radius*m_radius;
119 }

121 bool rayIntersect(const Ray &ray, Float mint, Float maxt, Float &t, void *tmp)
     const {
   Vector3d o = Vector3d(ray.o) - Vector3d(m_center);
123  Vector3d d(ray.d);

125  double A = d.lengthSquared();
   double B = 2 * dot(o, d);
127  double C = o.lengthSquared() - m_radius*m_radius;

129  double nearT, farT;
   if (!solveQuadraticDouble(A, B, C, nearT, farT))
131   return false;

133  if (!(nearT <= maxt && farT >= mint)) /* NaN-aware conditionals */
    return false;
135
        //Calculation of the centre and intersection points in object
     coordinates
137  Vector3d nearC =  Vector3d(ray.o) + Vector3d(ray.d) * nearT;
   Vector nearTest = m_worldToObject(Vector(nearC.x, nearC.y, nearC.z));
139  Vector3d farC =  Vector3d(ray.o) + Vector3d(ray.d) * farT;
   Vector farTest = m_worldToObject(Vector(farC.x, farC.y, farC.z));
141  Vector centertest = m_worldToObject(Vector(m_center.x, m_center.y, m_center.z)
     );

143  //additional check if either of the intersections is inside the bounding box
   if (nearT < mint || nearTest.x > (centertest.x - m_radius + m_height1)      ||
     nearTest.x < (centertest.x - m_radius + m_height2)  ) {
145   if (farT > maxt || farTest.x > (centertest.x - m_radius + m_height1)  ||
     nearTest.x < (centertest.x - m_radius + m_height2)  )
     return false;
147   t = (Float) farT;
   } else {
149   t = (Float) nearT;

151  }

153  return true;
  }
155
  bool rayIntersect(const Ray &ray, Float mint, Float maxt) const {
157  Vector3d o = Vector3d(ray.o) - Vector3d(m_center);
```

```
  Vector3d d(ray.d);

  double A = d.lengthSquared();
  double B = 2 * dot(o, d);
  double C = o.lengthSquared() - m_radius*m_radius;

  double nearT, farT;
  if (!solveQuadraticDouble(A, B, C, nearT, farT))
   return false;

  if (nearT > maxt || farT < mint)
   return false;
  if (nearT < mint && farT > maxt)
   return false;

  return true;
 }

 void fillIntersectionRecord(const Ray &ray,
   const void *temp, Intersection &its) const {
  its.p = ray(its.t);

  #if defined(SINGLE_PRECISION)
   /* Re-project onto the sphere to limit cancellation effects */
   its.p = m_center + normalize(its.p - m_center) * m_radius;
  #endif

  Vector local = m_worldToObject(its.p - m_center);
  Float theta = math::safe_acos(local.z/m_radius);
  Float phi = std::atan2(local.y, local.x);

  if (phi < 0)
   phi += 2*M_PI;

  its.uv.x = phi * (0.5f * INV_PI);
  its.uv.y = theta * INV_PI;
  its.dpdu = m_objectToWorld(Vector(-local.y, local.x, 0) * (2*M_PI));
  its.geoFrame.n = normalize(its.p - m_center);
  Float zrad = std::sqrt(local.x*local.x + local.y*local.y);
  its.shape = this;

  if (zrad > 0) {
   Float invZRad = 1.0f / zrad,
      cosPhi = local.x * invZRad,
      sinPhi = local.y * invZRad;
   its.dpdv = m_objectToWorld(Vector(local.z * cosPhi, local.z * sinPhi,
     -std::sin(theta)*m_radius) * M_PI);
      its.geoFrame.s = normalize(its.dpdu);
   its.geoFrame.t = normalize(its.dpdv);
  } else {
   // avoid a singularity
   const Float cosPhi = 0, sinPhi = 1;
   its.dpdv = m_objectToWorld(Vector(local.z * cosPhi, local.z * sinPhi,
     -std::sin(theta)*m_radius) * M_PI);
   coordinateSystem(its.geoFrame.n, its.geoFrame.s, its.geoFrame.t);
  }

  if (m_flipNormals)
   its.geoFrame.n *= -1;

```

```
    its.shFrame = its.geoFrame;
    its.wi = its.toLocal(-ray.d);
    its.hasUVPartials = false;
   its.instance = NULL;
   its.time = ray.time;
  }

  void samplePosition(PositionSamplingRecord &pRec, const Point2 &sample) const {
   Vector v = Warp::squareToUniformSphere(sample);

   pRec.p = Point(v * m_radius) + m_center;
   pRec.n = Normal(v);

   if (m_flipNormals)
    pRec.n *= -1;

   pRec.pdf = m_invSurfaceArea;
   pRec.measure = EArea;
  }

  Float pdfPosition(const PositionSamplingRecord &pRec) const {
   return m_invSurfaceArea;
  }

  void getNormalDerivative(const Intersection &its,
    Vector &dndu, Vector &dndv, bool shadingFrame) const {
   Float invRadius = (m_flipNormals ? -1.0f : 1.0f) / m_radius;
   dndu = its.dpdu * invRadius;
   dndv = its.dpdv * invRadius;
  }

  /**
   * Improved sampling strategy given in
   * "Monte Carlo techniques for direct lighting calculations" by
   * Shirley, P. and Wang, C. and Zimmerman, K. (TOG 1996)
   */
  void sampleDirect(DirectSamplingRecord &dRec, const Point2 &sample) const {
   const Vector refToCenter = m_center - dRec.ref;
   const Float refDist2 = refToCenter.lengthSquared();
   const Float invRefDist = static_cast<Float>(1) / std::sqrt(refDist2);

   /* Sine of the angle of the cone containing the
      sphere as seen from 'dRec.ref' */
   const Float sinAlpha = m_radius * invRefDist;

   if (sinAlpha < 1-Epsilon) {
    /* The reference point lies outside of the sphere.
       => sample based on the projected cone. */

    Float cosAlpha = math::safe_sqrt(1.0f - sinAlpha * sinAlpha);

    dRec.d = Frame(refToCenter * invRefDist).toWorld(
     Warp::squareToUniformCone(cosAlpha, sample));
    dRec.pdf = Warp::squareToUniformConePdf(cosAlpha);

    /* Distance to the projection of the sphere center
       onto the ray (dRec.ref, dRec.d) */
    const Float projDist = dot(refToCenter, dRec.d);

    /* To avoid numerical problems move the query point to the
```

```
      intersection of the of the original direction ray and a plane
279   with normal refToCenter which goes through the sphere's center */
    const Float baseT = refDist2 / projDist;
281 const Point query = dRec.ref + dRec.d * baseT;

283 const Vector queryToCenter = m_center - query;
    const Float queryDist2    = queryToCenter.lengthSquared();
285 const Float queryProjDist  = dot(queryToCenter, dRec.d);

287 /* Try to find the intersection point between the
      sampled ray and the sphere. */
289 Float A = 1.0f, B = -2*queryProjDist,
      C = queryDist2 - m_radius*m_radius;
291
    Float nearT, farT;
293 if (!solveQuadratic(A, B, C, nearT, farT)) {
     /* The intersection couldn't be found due to roundoff errors..
295      Don't give up -- one workaround is to project the closest
        ray position onto the sphere */
297  nearT = queryProjDist;
    }
299
    dRec.dist = baseT + nearT;
301 dRec.n = normalize(dRec.d*nearT - queryToCenter);
    dRec.p = m_center + dRec.n * m_radius;
303 } else {
    /* The reference point lies inside the sphere
305    => use uniform sphere sampling. */
    Vector d = Warp::squareToUniformSphere(sample);
307
    dRec.p = m_center + d * m_radius;
309 dRec.n = Normal(d);
    dRec.d = dRec.p - dRec.ref;
311
    Float dist2 = dRec.d.lengthSquared();
313 dRec.dist = std::sqrt(dist2);
    dRec.d /= dRec.dist;
315 dRec.pdf = m_invSurfaceArea * dist2
     / absDot(dRec.d, dRec.n);
317 }

319 if (m_flipNormals)
    dRec.n *= -1;
321
  dRec.measure = ESolidAngle;
323 }

325 Float pdfDirect(const DirectSamplingRecord &dRec) const {
    const Vector refToCenter = m_center - dRec.ref;
327 const Float invRefDist = (Float) 1.0f / refToCenter.length();

329 /* Sine of the angle of the cone containing the
      sphere as seen from 'dRec.ref' */
331 const Float sinAlpha = m_radius * invRefDist;

333 if (sinAlpha < 1-Epsilon) {
    /* The reference point lies outside the sphere */
335 Float cosAlpha = math::safe_sqrt(1 - sinAlpha*sinAlpha);
    Float pdfSA = Warp::squareToUniformConePdf(cosAlpha);
337
```

```
    if (dRec.measure == ESolidAngle)
339  return pdfSA;
    else if (dRec.measure == EArea)
341  return pdfSA * absDot(dRec.d, dRec.n)
      / (dRec.dist*dRec.dist);
343  else
     return 0.0f;
345 } else {
    /* The reference point lies inside the sphere */
347  if (dRec.measure == ESolidAngle)
     return m_invSurfaceArea * dRec.dist * dRec.dist
349   / absDot(dRec.d, dRec.n);
    else if (dRec.measure == EArea)
351  return m_invSurfaceArea;
    else
353  return 0.0f;
 }
355}

357ref<TriMesh> createTriMesh() {
 /// Choice of discretization
359 const uint32_t thetaSteps = 30;
 const uint32_t phiSteps = thetaSteps * 2;
361 const Float dTheta = M_PI / (thetaSteps-1);
 const Float dPhi   = (2*M_PI) / (phiSteps-1);
363
 /// Precompute cosine and sine tables
365 Float *cosPhi = new Float[phiSteps];
 Float *sinPhi = new Float[phiSteps];
367 for (uint32_t i=0; i<phiSteps; ++i) {
  sinPhi[i] = std::sin(i*dPhi);
369  cosPhi[i] = std::cos(i*dPhi);
 }
371
 /// Define the starting- and ending-values
373 Float height_start = (- m_radius + m_height1)/m_radius;  // higher edge of the
    spherical segmnt, in units of m_radius
 Float height_end   = (- m_radius + m_height2)/m_radius;  // lower edge of the
   spherical segment (-m_radius for spherical cap), in units of m_radius
375 Float theta_high = std::acos(height_start);                // exact theat-
   value for the higher edge
 Float theta_low  = std::acos(height_end);                  // exact theta-
   value for the lower edge
377
 uint32_t theta_start = floor( theta_high/dTheta + 0.5);
379 uint32_t theta_end;
 if (-m_radius + m_height2 > -m_radius) {
381  theta_end = floor( theta_low/dTheta + 0.5);
 }
383 else {
  theta_end = thetaSteps;
385 }

387 size_t numTris = 2 * (phiSteps-1) * (thetaSteps-1);
 size_t numVertices = thetaSteps * phiSteps;
389
 ref<TriMesh> mesh = new TriMesh("Sphere approximation",
391  numTris, numVertices, true, true, false);

393 Point *vertices = mesh->getVertexPositions();
```

```
  Normal *normals = mesh->getVertexNormals();
  Point2 *texcoords = mesh->getVertexTexcoords();
  Triangle *triangles = mesh->getTriangles();
  uint32_t vertexIdx = 0;
  for (uint32_t theta=0; theta<theta_start; ++theta) {
   for (uint32_t phi=0; phi<phiSteps; ++phi) {
    vertexIdx++;
   }
  }
  for (uint32_t theta=theta_start; theta<theta_end; ++theta) { //adepted number
   of used theta Steps to represent the correct part of the sphere
//  for (uint32_t theta=0; theta<thetaSteps; ++theta) { //original code in the
   sphere class
   Float sinTheta = std::sin(theta * dTheta);
   Float cosTheta = std::cos(theta * dTheta);

   for (uint32_t phi=0; phi<phiSteps; ++phi) {
    Vector v(
     sinTheta * cosPhi[phi],
     sinTheta * sinPhi[phi],
     cosTheta
    );
    texcoords[vertexIdx] = Point2(phi * dPhi * INV_TWOPI, theta * dTheta *
   INV_PI);
    vertices[vertexIdx] = m_objectToWorld(Point(v*m_radius));
    normals[vertexIdx++] = m_objectToWorld(Normal(v));
   }
  }
  for (uint32_t theta=theta_end; theta<thetaSteps; ++theta) {
   for (uint32_t phi=0; phi<phiSteps; ++phi) {
    vertexIdx++;
   }
  }
  Assert(vertexIdx == numVertices);

  uint32_t triangleIdx = 0;
  for (uint32_t theta=1; theta<thetaSteps; ++theta) {
   for (uint32_t phi=0; phi<phiSteps-1; ++phi) {
    uint32_t nextPhi = phi + 1;
    uint32_t idx0 = phiSteps*theta + phi;
    uint32_t idx1 = phiSteps*theta + nextPhi;
    uint32_t idx2 = phiSteps*(theta-1) + phi;
    uint32_t idx3 = phiSteps*(theta-1) + nextPhi;

    triangles[triangleIdx].idx[0] = idx0;
    triangles[triangleIdx].idx[1] = idx2;
    triangles[triangleIdx].idx[2] = idx1;
    triangleIdx++;
    triangles[triangleIdx].idx[0] = idx1;
    triangles[triangleIdx].idx[1] = idx2;
    triangles[triangleIdx].idx[2] = idx3;
    triangleIdx++;
   }
  }
//  Assert(triangleIdx == numTris);
  delete[] cosPhi;
  delete[] sinPhi;
  mesh->copyAttachments(this);
  mesh->configure();
```

```
451   return mesh.get();
  }
453
  size_t getPrimitiveCount() const {
455   return 1;
  }
457
  size_t getEffectivePrimitiveCount() const {
459   return 1;
  }
461
  std::string toString() const {
463   std::ostringstream oss;
    oss << "Sphere[" << endl
465     << "  radius = " << m_radius << "," << endl
      << "  center = " << m_center.toString() << "," << endl
467     << "  bsdf = " << indent(m_bsdf.toString()) << "," << endl;
    if (isMediumTransition())
469     oss << "  interiorMedium = " << indent(m_interiorMedium.toString()) << "," <<
        endl
        << "  exteriorMedium = " << indent(m_exteriorMedium.toString()) << "," <<
        endl;
471   oss << "  emitter = " << indent(m_emitter.toString()) << "," << endl
      << "  sensor = " << indent(m_sensor.toString()) << "," << endl
473     << "  subsurface = " << indent(m_subsurface.toString()) << endl
      << "]";
475   return oss.str();
  }
477
 MTS_DECLARE_CLASS()
479 private:
  Transform m_objectToWorld;
481 Transform m_worldToObject;
  Point m_center;
483 Float m_radius;
  Float m_height1;    //
485 Float m_height2;    //
  Float m_invSurfaceArea;
487 bool m_flipNormals;
};
489
MTS_IMPLEMENT_CLASS_S(Sphericalsegment, false, Shape)
491 MTS_EXPORT_PLUGIN(Sphericalsegment, "Sphere intersection primitive");
MTS_NAMESPACE_END
```

# The configuration files for PMD

## The screen

'CompleteSizeX' and 'CompleteSizeY' give the size of the screen in metres, while 'OffsetX' and 'OffsetY' define the point on the screen 'zerobase' points at, which is the centre in this case.
The geometry parameters 'zerobase', 'xbase' and 'ybase' define the sceen's own coordinate system.
The parameters of the model used to describe the surface were all set to 0, therefore the specific model that is used is not relevant for the simulation.

```
[Pattern]
CompleteSizeX=1.32828
CompleteSizeY=0.74716
OffsetX=-0.6641
OffsetY=-0.3736
AspectRatio=1.77777777778
GlobalScaleFactor=1
[ScreenPosition]
zerobase=(0|1.34|1.10)
xbase=(1|0|0)
ybase=(0|-0,70710678118655|0,70710678118655)
[Flatscreen]
glasdicke=0
brechungsindex=1
[FlatscreenZParams]
p0=0
p1=0
p2=0
p3=0
p4=0
p5=0
p6=0
p7=0
p8=0
p9=0
p10=0
p11=0
p12=0
p13=0
p14=0
p15=0
p16=0
p17=0
p18=0
p19=0
p20=0
```

# The cameras

The general properties of the camera mainly consist of the resolution, size of the pixels and bit depth of the image.
The internal calibration parameters were set to 0, except for $c$, which is the focal length of the objective lens in metres.
The parameters of the geometry calibration are listed at the end, consisting of the vectors 'zerobase', 'xbase' and 'ybase', as described in chapeter 2.1.3.

```
[Properties]
SizeX=1280
SizeY=960
PixelSizeX=3.75e-006
PixelSizeY=3.75e-006
BitsPerPixel=8
CameraType=
LensType=Pentax
CameraName=
[CalibParams]
c=0.025
xp=0
yp=0
k1=0
k2=0
k3=0
p1=0
p2=0
b1=0
b2=0
[CamPosition]
zerobase=(-1.65000000000000|-2.85000000000000|2.90000000000000)
xbase=(0.95033267466439|-0.22007704044860|0.22007704044860)
ybase=(0.31104180956730|0.69653814657658|-0.64659694019131)
```

# The scene files for the ray tracer

The following scene files are examples of the ones used for the simulation of the PMD measurement.

## The miror

The ground simply ensures that the background of the image is black instead of transparent and serves no other purpose.
The position of the mirror is defined by the centre of the sphere it is part of, while the parameter 'height1' gives the higher border of the mirror itself.
As the segment of the sphere was defined along the $x$ axis of its own coordinate system, the object was created at the origin of the shared coordinate system and rotated around the $y$ axis.

```
<?xml version='1.0' encoding='utf-8'?>

<scene version="0.5.0">


    <!-- Ground -->
    <shape type="rectangle">
  <transform name="toWorld">
        <scale x="200" y="200"/>
        <translate x="0" y="0" z="0.28"/>
  </transform>
        <bsdf type="diffuse">
            <texture name="reflectance" type="gridtexture">
                <float name="uvscale" value="400"/>
                <spectrum name="color0" value="0"/>
                <spectrum name="color1" value="0"/>
            </texture>
        </bsdf>
 </shape>


 <!-- Mirror -->
 <shape type="sphericalsegment">
  <point name="center" x="0" y="0" z="0"/>
  <float name="radius" value="32"/>
  <float name="height1" value="0.00562549447169"/>

        <transform name="toWorld">
        <rotate y="1" angle="-90"/>
        <translate x="0" y="-0.2" z="32.3"/>
  </transform>

        <boolean name="flipNormals" value="true"/>

        <bsdf type="conductor">
            <string name="material" value="none"/>
        </bsdf>
 </shape>

</scene>
```

## The screen

The rotations and translations of the objects in this scene file were done in accordance with the geometry parameters in the corresponding configuration file for the screen. The example shown here was taken from the symmetrical setup.
The actual light source was first moved to the position of the screen itself and then translated slightly backwards, while a diffuse transmitting plane was used as the screen itself, its transmittance defined by an image showing the sinusoidal brightness pattern.
For each of the different stages of the phase shifting, an individual scene file was written. The different stages of the pattern were generated by different images used for the 'transmittance' parameter.

```
<?xml version='1.0' encoding='utf-8'?>

<scene version="0.5.0">


    <!-- rectangle as lightsource -->
 <shape type="rectangle">
  <transform name="toWorld">
        <scale x="0.66414000000000" y="0.37358000000000"/>
   <rotate z="1" angle="0"/>
            <rotate y="1" angle="0"/>
            <rotate x="1" angle="135"/>
  <translate x="0" y="1.34" z="1.10"/>
        <translate x="0" y="0.00070710678119" z="0.00070710678119"/>
  </transform>

  <emitter type="area">
   <spectrum name="radiance" value="1"/>
  </emitter>
 </shape>

    <!-- diffuse transmitting material as screen -->
 <shape type="rectangle">
  <transform name="toWorld">
        <scale x="0.66414000000000" y="0.37358000000000"/>
   <rotate z="1" angle="0"/>
            <rotate y="1" angle="0"/>
            <rotate x="1" angle="135"/>
  <translate x="0" y="1.34" z="1.10"/>
  </transform>
        <bsdf type="difftrans">
            <texture name="transmittance" type="bitmap">
            <string name="filename" value="SF1_Horizontal_P_Shift0.bmp"/>
        </texture>
        </bsdf>
 </shape>

</scene>
```

## The cameras

A similar scene file was create for each of the cameras. The parameter 'fow' defines the angle the camera can see, while the position and orientation is set by the 'lookat' command.
The argument 'origin' defines the position of the camera using a cartesian vector and is identical to the 'zerobase' parameter of the geometry calibration, 'target' is a point along the sight axis of the camera and 'up' is a vector that fixes the orientation of the camera, as it points from the centre of the image plane to the centre of the images upper border.
The resolution of the output images is set with the parameters 'width' and 'height'.

```
<?xml version='1.0' encoding='utf-8'?>

<scene version="0.5.0">

 <sensor type="perspective">
  <float name="farClip" value="41.2071"/>
  <float name="fov" value="13.68554682526190"/>
        <string name="fovAxis" value="diagonal"/>
  <transform name="toWorld">
   <lookat
            target="-0.40583276173079, -0.06384741369370, 0.31361223923475"
            origin="-1.65000000000000, -2.85000000000000, 2.90000000000000"
            up="-0.01099091289770, 0.68293536050717, 0.73039612074724"/>
  </transform>

  <sampler type="independent">
            <integer name="sampleCount" value="100"/>
        </sampler>

  <film type="ldrfilm">
            <string name="pixelFormat" value="rgba"/>
   <integer name="width" value="1280"/>
   <integer name="height" value="960"/>
            <boolean name="banner" value="false"/>
   <rfilter type="gaussian"/>
  </film>
 </sensor>
</scene>
```

## The combined files

Scene files similar to the one shown below were used for the ray tracing itself. Each of them contained commands to include the mirror as well as the screen that displays the correct stage of the phase shifting and the camera.
With 32 stages of the phase shifting and 4 cameras, is resulted in 128 scene files.

```
<?xml version='1.0' encoding='utf-8'?>

<scene version="0.5.0">
 <integrator type="path"/>

    <include filename="Mirror.xml"/>
    <include filename="Screen_SF1_Horizontal_P_Shift0.xml"/>
    <include filename="TCam_0.xml"/>
</scene>
```

# Geometry parameters

What follows are excerpts of the configuration files that include the resulting geometry parameters, beginning with the initial values of the optimal calibrated setups, followed by the results generated during the different studies on deliberate errors in the setup.
The geometry parameters of the screen are not changed during the recalibration, it is therefore only listed once for the main calibration.

## Symmetric setup

### Camera 0

```
zerobase=(-1.65|-2.85|2.9)
xbase=(0.950332674664|-0.220077040449|0.220077040449)
ybase=(0.311041809567|0.696538146577|-0.646596940191)
```

### Camera 1

```
zerobase=(-0.95|-4|1.75)
xbase=(0.98834741198|-0.139907621514|0.0599604092203)
ybase=(0.152182245003|0.916405691942|-0.370190723929)
```

### Camera 2

```
zerobase=(0.95|-4|1.75)
xbase=(0.98834741198|0.139907621514|-0.0599604092203)
ybase=(-0.152182245003|0.916405691942|-0.370190723929)
```

### Camera 3

```
zerobase=(1.65|-2.85|2.9)
xbase=(0.950332674664|0.220077040449|-0.220077040449)
ybase=(-0.311041809567|0.696538146577|-0.646596940191)
```

### Screen

```
zerobase=(0|1.34|1.1)
xbase=(1|0|0)
ybase=(0|-0.707106781187|0.707106781187)
```

## Simulation of the setup in Erlangen

### Camera 0

```
zerobase=(-1.57571479692|-2.87192586054|2.91047380992)
xbase=(0.956899986334|-0.223207330486|0.185798018749)
ybase=(0.290411456975|0.739613078164|-0.607152106368)
```

### Camera 1

```
zerobase=(-0.778230874843|-4.05698603532|1.75057868306)
xbase=(0.987741517462|-0.131800876609|0.0836374533738)
ybase=(0.153657453001|0.915335478107|-0.372223521094)
```

### Camera 2

```
zerobase=(1.11327145636|-4.00502322561|1.74959009479)
xbase=(0.984173664391|0.168543981438|-0.0547277319167)
ybase=(-0.176824354122|0.913767875615|-0.365734080014)
```

### Camera 3

```
zerobase=(1.74145255008|-2.674274476|2.96489682915)
xbase=(0.950298385674|0.21897592857|-0.221319951409)
ybase=(-0.310964122341|0.702515859355|-0.640134971682)
```

### Screen

```
zerobase=(0.00487273395123|1.34515892776|1.10213452452)
xbase=(0.999749116896|0.0176957278023|0.0137318783253)
ybase=(0.003463435832|-0.727820865298|0.685758552735)
```

# Moving the Cameras

The following geometry parameters resulted from the recalibration during the study on displaced cameras, that used the symmetric setup as starting point.

## Displacement of camera 1

Camera 1 displaced by $10^{-1}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.623562045004878|-2.756242370954894|3.099259310430632)
xbase=(0.953518748463077|-0.196171631002089|0.228732786279100)
ybase=(0.301050106768319|0.653091395928721|-0.694867226006997)
```

Camera 1

```
zerobase=(-0.968840938055655|-3.841854840787461|2.121931844503793)
xbase=(0.985343086647496|-0.149635978560730|0.081902841924395)
ybase=(0.170552682804271|0.873422465684061|-0.456119478675892)
```

Camera 2

```
zerobase=(1.090386037447493|-3.822606663645163|2.129608839653571)
xbase=(0.984734975699499|0.153048026980677|-0.082905543062950)
ybase=(-0.174046635527303|0.871783177014600|-0.457932157570183)
```

Camera 3

```
zerobase=(1.622250118085800|-2.754442122485029|3.101312908457387)
xbase=(0.953666732146331|0.196375753179319|-0.227939306746789)
ybase=(-0.300614694335397|0.652843006479627|-0.695289015043584)
```

Camera 1 displaced by $10^{-2}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.647719042086967|-2.845631599444630|2.911164700547272)
xbase=(0.950559565455434|-0.218688242355521|0.220481212751207)
ybase=(0.310343904144647|0.694343777631892|-0.649286823849161)
```

Camera 1

```
zerobase=(-0.947516030928096|-3.991643615080097|1.770676301788704)
xbase=(0.988162758808876|-0.140693934420722|0.061152096608232)
ybase=(0.153376688225824|0.914265814409685|-0.374958147146585)
```

Camera 2

```
zerobase=(0.957575184708992|-3.991444010777325|1.770793138197186)
xbase=(0.988163305405287|0.140702621897549|-0.061123269225297)
ybase=(-0.153373855572309|0.914256175156636|-0.374982808426958)
```

Camera 3

```
zerobase=(1.647774202179348|-2.845693132158599|2.911165154933479)
xbase=(0.950563686050829|0.218697317788559|-0.220454444167416)
ybase=(-0.310332082577212|0.694351104440244|-0.649284638880286)
```

Camera 1 displaced by $10^{-3}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.649693459826722|-2.849607849784356|2.900932174541761)
xbase=(0.950354697994921|-0.219942074571380|0.220116859490209)
ybase=(0.310973585957827|0.696359613327516|-0.646822013975161)
```

Camera 1

```
zerobase=(-0.950079422485477|-3.999098615616460|1.751699272819472)
xbase=(0.988318672319967|-0.140043713545300|0.060116222762866)
ybase=(0.152368246277636|0.916214919772584|-0.370586209016635)
```

Camera 2

```
zerobase=(0.950670532044555|-3.999166915347750|1.751763920418406)
xbase=(0.988332171730053|0.139985651712253|-0.060029456412561)
ybase=(-0.152281871309480|0.916225596905893|-0.370595314657173)
```

Camera 3

```
zerobase=(1.649719312991970|-2.849608663383146|2.900987365568909)
xbase=(0.950361371377260|0.219953601820776|-0.220076524962602)
ybase=(-0.310954527030224|0.696359139444284|-0.646831686787085)
```

Camera 1 displaced by $10^{-6}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.649974521415557|-2.849777718229872|2.900083421601449)
xbase=(0.950330792590853|-0.220078722818887|0.220083485104316)
ybase=(0.311047766039654|0.696516559413437|-0.646617328645473)
```

Camera 1

```
zerobase=(-0.950091025587154|-3.999902424065224|1.749974236454366)
xbase=(0.988343960911447|-0.139932384577797|0.059959508631895)
ybase=(0.152204847736816|0.916404322478815|-0.370184821499318)
```

Camera 2

```
zerobase=(0.950040956414468|-3.999909105109484|1.749976463349618)
xbase=(0.988346809157505|0.139912232810736|-0.059959585866039)
ybase=(-0.152186193477215|0.916407214481040|-0.370185331642062)
```

Camera 3

```
zerobase=(1.650000365799186|-2.849982730144145|2.899984020362801)
xbase=(0.950334141035901|0.220071939867327|-0.220075808902753)
ybase=(-0.311037168354683|0.696543844285057|-0.646593034984521)
```

Camera 1 displaced by $10^{-9}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.649965062481681|-2.849753059694590|2.900113798579127)
xbase=(0.950331233814334|-0.220074475291649|0.220085827262105)
ybase=(0.311046369794140|0.696510290893702|-0.646624752477863)
```

Camera 1

```
zerobase=(-0.950048747327898|-3.999875190361369|1.750023122701000)
xbase=(0.988345001037703|-0.139924305432670|0.059961218067200)
ybase=(0.152198035877313|0.916400929523404|-0.370196021378750)
```

Camera 2

```
zerobase=(0.950071431645488|-3.999875770442293|1.750024107173508)
xbase=(0.988346055161001|0.139916844753182|-0.059961252498052)
ybase=(-0.152191131959811|0.916401985003359|-0.370196246922658)
```

Camera 3

```
zerobase=(1.650017848318398|-2.849911773863354|2.900033818716544)
xbase=(0.950333179330167|0.220072248839051|-0.220079652750438)
ybase=(-0.311040212263560|0.696531187045085|-0.646605205537818)
```

## Displacement of camera 3

Camera 3 displaced by $10^{-1}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.671753860366407|-2.884550783214373|2.799785523459516)
xbase=(0.948131068271923|-0.232447668320922|0.216830714773813)
ybase=(0.317710268508731|0.715211194370941|-0.622521592180225)
```

Camera 1

```
zerobase=(-0.891041181066902|-4.065792393182208|1.566006055173291)
xbase=(0.989723065463846|-0.133687882947510|0.050752375717738)
ybase=(0.142953090429937|0.933836634073840|-0.327892596438372)
```

Camera 2

```
zerobase=(0.885872227837866|-4.067381052728622|1.566367520380992)
xbase=(0.989866099896918|0.133009682926749|-0.049734580743427)
ybase=(-0.141975567556346|0.933979752868034|-0.327909681849354)
```

Camera 3

```
zerobase=(1.767516968765799|-2.881798947367209|2.802775312830748)
xbase=(0.948358961535069|0.232178847277122|-0.216120945196911)
ybase=(-0.317046871570418|0.714940359951652|-0.623170572908905)
```

Camera 3 displaced by $10^{-2}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.651041753784660|-2.852030894363571|2.894287608775765)
xbase=(0.950217783166211|-0.220782455175459|0.219866486852822)
ybase=(0.311394492961605|0.697655898054901|-0.645220673616704)
```

Camera 1

```
zerobase=(-0.946461676226048|-4.004075612933773|1.739224473036692)
xbase=(0.988431495465999|-0.139561769706111|0.059377531158519)
ybase=(0.151635055940281|0.917494924732577|-0.367708951347225)
```

Camera 2

```
zerobase=(0.946321750612797|-4.004125954131023|1.739225539927218)
xbase=(0.988436880743548|0.139531463678874|-0.059359105693482)
ybase=(-0.151600057575044|0.917501084273270|-0.367708013104886)
```

Camera 3

```
zerobase=(1.661314871264742|-2.852001071390928|2.894237228958305)
xbase=(0.950206467707228|0.220809202822169|-0.219888527840822)
ybase=(-0.311429054718047|0.697649805286257|-0.645210580401059)
```

Camera 3 displaced by $10^{-3}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.650010788597482|-2.850061307939665|2.899499892916396)
xbase=(0.950324315864605|-0.220142814814827|0.220047348908114)
ybase=(0.311067849471521|0.696639252408740|-0.646475479062087)
```

Camera 1

```
zerobase=(-0.949755712227074|-4.000289877099873|1.748908115604924)
xbase=(0.988352110231901|-0.139899597470622|0.059901659633438)
ybase=(0.152151921745507|0.916512269431926|-0.369939255406636)
```

Camera 2

```
zerobase=(0.949799287642932|-4.000277129788750|1.748904498969264)
xbase=(0.988352279793875|0.139893245303056|-0.059913695796236)
ybase=(-0.152150542552014|0.916512518250754|-0.369939206210581)
```

Camera 3

```
zerobase=(1.650851634342694|-2.850587744122781|2.899285054091983)
xbase=(0.950339213856818|0.220112898461161|-0.220012932657710)
ybase=(-0.311021524991034|0.696710070287694|-0.646421448400320)
```

Camera 3 displaced by $10^{-6}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.649979763490840|-2.849772558833961|2.900083425162796)
xbase=(0.950330503570601|-0.220079977829423|0.220083478120149)
ybase=(0.311048681079248|0.696516150546350|-0.646617328894729)
```

Camera 1

```
zerobase=(-0.950047806024073|-3.999907506356660|1.749970394142255)
xbase=(0.988345111409070|-0.139924316688320|0.059959372513677)
ybase=(0.152197313047397|0.916405929512973|-0.370183941109743)
```

Camera 2

```
zerobase=(0.950024403504466|-3.999919572072906|1.749968758621258)
xbase=(0.988347218649137|0.139909454393311|-0.059959319200741)
ybase=(-0.152183511268712|0.916408372874455|-0.370183566659090)
```

Camera 3

```
zerobase=(1.649945902299691|-2.850063176896787|2.899953398156059)
xbase=(0.950337401131424|0.220060222454229|-0.220073447975395)
ybase=(-0.311026849387799|0.696555398987204|-0.646585551262690)
```

Camera 3 displaced by $10^{-9}$ m along the $x$ axis
Camera 0

```
zerobase=(-1.649961291596970|-2.849815945184465|2.900047849402069)
xbase=(0.950331664442895|-0.220077860203562|0.220080582980490)
ybase=(0.311045013919483|0.696525871657946|-0.646608621522298)
```

Camera 1

```
zerobase=(-0.949989575342526|-3.999930102040264|1.749928040697010)
xbase=(0.988346528680156|-0.139914927521589|0.059957921091782)
ybase=(0.152188029915359|0.916411390009975|-0.370174239799845)
```

Camera 2

```
zerobase=(0.950110608211430|-3.999875182414200|1.749948622738198)
xbase=(0.988345056608275|0.139925004099654|-0.059958671639991)
ybase=(-0.152197670615916|0.916407883397615|-0.370178957136569)
```

Camera 3

```
zerobase=(1.650001572853382|-2.849980584763718|2.899969494107121)
xbase=(0.950334480873967|0.220071569457340|-0.220074711809885)
ybase=(-0.311036091598705|0.696547618738770|-0.646589486886658)
```

# Tilt of the Screen

The following sets of parameters were generated by the recalibration during the study on a tilted screen, that used the reconstruction of setup in Erlangen as starting point.

## Rotation around the $x$ axis

Screen rotated by $\theta = 10^{-0\,\circ}$ around the $x$ axis
Camera 0

```
zerobase=(-1.567519177164890|-2.833032898697348|2.983699439546786)
xbase=(0.957566271130083|-0.217005726274314|0.189671693085539)
ybase=(0.288208477030411|0.724732188463741|-0.625858713107868)
```

Camera 1

```
zerobase=(-0.785847129324464|-4.029703064183450|1.869617332719977)
xbase=(0.987588957298041|-0.130326596437001|0.087652893186735)
ybase=(0.154686544266100|0.903767534733255|-0.399094370024092)
```

Camera 2

```
zerobase=(1.127817465572184|-3.974091901460922|1.867725138517558)
xbase=(0.983782331243203|0.168917304333248|-0.060326354365282)
ybase=(-0.178969683073427|0.902063466521564|-0.392748462642434)
```

Camera 3

```
zerobase=(1.732919644779004|-2.635364053099303|3.032734613696680)
xbase=(0.951068446750338|0.211717534641855|-0.225043318319964)
ybase=(-0.308597566344765|0.687117642308485|-0.657751387435583)
```

Screen rotated by $\theta = 10^{-1\,\circ}$ around the $x$ axis
Camera 0

```
zerobase=(-1.574592100576183|-2.868460222171970|2.917715230102006)
xbase=(0.956984256374379|-0.222554182807265|0.186147169643582)
ybase=(0.290133755661770|0.738205945219872|-0.608994569982606)
```

Camera 1

```
zerobase=(-0.778815752525534|-4.054413678575163|1.762418781892743)
xbase=(0.987729858019063|-0.131629310357600|0.084044346819493)
ybase=(0.153733846575796|0.914231534273693|-0.374895460303696)
```

Camera 2

```
zerobase=(1.115333382931906|-4.001858067225181|1.761380577150373)
xbase=(0.984116110413647|0.168694016283685|-0.055297469154759)
ybase=(-0.177140946147113|0.912622909695061|-0.368429789645361)
```

Camera 3

```
zerobase=(1.739095808414376|-2.671169639512834|2.971861456302326)
xbase=(0.950466105211021|0.218074750444461|-0.221489471675210)
ybase=(-0.310451128014466|0.701120604643897|-0.641911204807363)
```

Screen rotated by $\theta = 10^{-2\,\circ}$ around the $x$ axis
Camera 0

```
zerobase=(-1.575434092389137|-2.871622667400447|2.911271425792065)
xbase=(0.956919071070218|-0.223125734515707|0.185797734159185)
ybase=(0.290348700679345|0.739474515223589|-0.607350865109553)
```

Camera 1

```
zerobase=(-0.778451953442703|-4.056684114327316|1.751846662701866)
xbase=(0.987737564249823|-0.131813285652295|0.083664579693341)
ybase=(0.153687800697773|0.915214118250941|-0.372509298770614)
```

Camera 2

```
zerobase=(1.113468674279438|-4.004697715025926|1.750811746413691)
xbase=(0.984168038731964|0.168549746914904|-0.054811078751740)
ybase=(-0.176856987206771|0.913648991955011|-0.366015198558796)
```

Camera 3

```
zerobase=(1.739874851830042|-2.674595365761304|2.965733979133924)
xbase=(0.950393741709020|0.218703571367238|-0.221179754025838)
ybase=(-0.310671390216780|0.702470453574695|-0.640326908035461)
```

## Rotation around the $y$ axis

Screen rotated by $\theta = 10^{-0\,\circ}$ around the $y$ axis
Camera 0

```
zerobase=(-1.605829553919925|-2.857341943790462|2.918418853558296)
xbase=(0.956183287481238|-0.235988953067376|0.173270697959829)
ybase=(0.292357712966775|0.738309513018003|-0.607804270021711)
```

Camera 1

```
zerobase=(-0.830046885699889|-4.054470429100471|1.738491735849452)
xbase=(0.987179834110958|-0.143409874397550|0.070068416921113)
ybase=(0.159218087493885|0.915604133144441|-0.369213585859658)
```

Camera 2

```
zerobase=(1.050791354310696|-4.025819892959341|1.713716038880153)
xbase=(0.985523521294689|0.156179310522568|-0.065965232813497)
ybase=(-0.169477263001860|0.918014534929038|-0.358506305362568)
```

Camera 3

```
zerobase=(1.718005162272664|-2.699815184956228|2.931699433543301)
xbase=(0.949789267688766|0.208969488761499|-0.232877864446970)
ybase=(-0.311181382716070|0.708551973503377|-0.633340546542144)
```

Screen rotated by $\theta = 10^{-1\,\circ}$ around the $y$ axis
Camera 0

```
zerobase=(-1.578432547148951|-2.870691505688920|2.911262688514166)
xbase=(0.956862672099472|-0.224465936578305|0.184469157472010)
ybase=(0.290540551589936|0.739517484319244|-0.607206783780140)
```

Camera 1

```
zerobase=(-0.784018190483673|-4.056542221806176|1.749372021981612)
xbase=(0.987686247587617|-0.133075991447716|0.082259691382777)
ybase=(0.154305259377062|0.915348933743867|-0.371922325791667)
```

Camera 2

```
zerobase=(1.107763681323859|-4.006891823085053|1.745913129993002)
xbase=(0.984299657362690|0.167407952960438|-0.055935335896139)
ybase=(-0.176223757014519|0.914178459011386|-0.364997167856895)
```

Camera 3

```
zerobase=(1.737796376733304|-2.677683790690860|2.961557402691157)
xbase=(0.950336770930217|0.217790758795496|-0.222322304775290)
ybase=(-0.310754023501019|0.703251501548485|-0.639428856438812)
```

Screen rotated by $\theta = 10^{-2\,\circ}$ around the $y$ axis
Camera 0

```
zerobase=(-1.575897268735941|-2.872122898261553|2.910367840977691)
xbase=(0.956899674173043|-0.223295785745971|0.185693310690947)
ybase=(0.290413227483346|0.739643249351251|-0.607114504020863)
```

Camera 1

```
zerobase=(-0.779510064585466|-4.056661536495806|1.750501704165726)
xbase=(0.987713917695627|-0.132014245149430|0.083626884838983)
ybase=(0.153850628176918|0.915310162598069|-0.372205978530710)
```

Camera 2

```
zerobase=(1.112369758017045|-4.005329786256535|1.749314856537063)
xbase=(0.984198265241652|0.168394210172546|-0.054746366785512)
ybase=(-0.176692932961762|0.913817523380735|-0.365673544851099)
```

Camera 3

```
zerobase=(1.739449491326343|-2.675204995682535|2.964897520173963)
xbase=(0.950404620344082|0.218676247536154|-0.221160024397529)
ybase=(-0.310633309031623|0.702681568545266|-0.640113709074994)
```

## Rotation around the $z$ axis

Screen rotated by $\theta = 10^{-0\,\circ}$ around the $z$ axis
Camera 0

```
zerobase=(-1.500707865397184|-2.891693296273342|2.928204376764659)
xbase=(0.961775006204843|-0.211106976524363|0.174420990428807)
ybase=(0.273839753674762|0.743281288693410|-0.610364411795345)
```

Camera 1

```
zerobase=(-0.702030882573967|-4.066822880041794|1.758258354287753)
xbase=(0.990252656247979|-0.119290554308880|0.071898820897087)
ybase=(0.137633132423437|0.917282212462199|-0.373698359053805)
```

Camera 2

```
zerobase=(1.188117154824192|-3.991197389370060|1.734779237557605)
xbase=(0.981252736539856|0.180965698824361|-0.066290895925332)
ybase=(-0.192652148919906|0.911548433047021|-0.363269326707283)
```

Camera 3

```
zerobase=(1.813331280421004|-2.652130279298250|2.943602028171156)
xbase=(0.944793870642795|0.230949536756626|-0.232436773052567)
ybase=(-0.327301743765994|0.698590386557233|-0.636274343610107)
```

Screen rotated by $\theta = 10^{-1\,\circ}$ around the $z$ axis
Camera 0

```
zerobase=(-1.567957355058568|-2.873918274362971|2.912284480816316)
xbase=(0.957419405786511|-0.222035170889435|0.184522259663306)
ybase=(0.288695835039211|0.740025964586323|-0.607467107397400)
```

Camera 1

```
zerobase=(-0.770214438352151|-4.058257122189811|1.750944182390352)
xbase=(0.988023565501644|-0.130537454110849|0.082276406627694)
ybase=(0.151969153836899|0.915595838844106|-0.372276290103459)
```

Camera 2

```
zerobase=(1.121732727905491|-4.003524068723827|1.747517709845239)
xbase=(0.983863003502911|0.169917248185185|-0.056051040203838)
ybase=(-0.178592589674275|0.913570873387282|-0.365366865235661)
```

Camera 3

```
zerobase=(1.747758229250311|-2.672962329344478|2.962369785923432)
xbase=(0.949804193871612|0.220041418607057|-0.222382030300736)
ybase=(-0.312464188172120|0.702299856482204|-0.639641339106481)
```

Screen rotated by $\theta = 10^{-2\,\circ}$ around the $z$ axis
Camera 0

```
zerobase=(-1.575041525507744|-2.872134247502555|2.910638719929220)
xbase=(0.956943921804225|-0.223069390384428|0.185737388791857)
ybase=(0.290266410220872|0.739641597240043|-0.607186724764594)
```

Camera 1

```
zerobase=(-0.778276734903148|-4.056737108528969|1.750830381473642)
xbase=(0.987743724725798|-0.131778903894320|0.083646008590417)
ybase=(0.153642044786853|0.915314966194144|-0.372280317414805)
```

Camera 2

```
zerobase=(1.114166119671560|-4.004856629844104|1.749624540819610)
xbase=(0.984142657354199|0.168706221116579|-0.054785408029029)
ybase=(-0.176996139700924|0.913729786038288|-0.365746147809782)
```

Camera 3

```
zerobase=(1.740595205337269|-2.674538493983890|2.965076598092749)
xbase=(0.950341845470584|0.218906229648245|-0.221202258958042)
ybase=(-0.310833129894454|0.702546971055587|-0.640164446700655)
```

# Deformation of the Screen

The last test of the recalibration were performed on the study of a deformed screen, which resulted in the following geometry parameters.

## Pattern 1

Pattern 1 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.565624195735637|-2.880504851378695|2.906372322794872)
xbase=(0.957439625748679|-0.222094322128636|0.184346074339001)
ybase=(0.288626208361992|0.741275348829172|-0.605975056471335)
```

Camera 1

```
zerobase=(-0.771632532873416|-4.059608363910744|1.744773971454056)
xbase=(0.987900350822400|-0.130910581245846|0.083158382396150)
ybase=(0.152610531687801|0.916044526877663|-0.370907603582328)
```

Camera 2

```
zerobase=(1.097064037690354|-4.011496212523253|1.744310701049343)
xbase=(0.984659092380884|0.166379471024332|-0.052577023630525)
ybase=(-0.174038948015831|0.914840854388117|-0.364385312157697)
```

Camera 3

```
zerobase=(1.728755833820597|-2.682314389759152|2.963361067116625)
xbase=(0.951077953714563|0.217843493817058|-0.219077470678943)
ybase=(-0.308588025078530|0.704187010536297|-0.639448266061457)
```

Pattern 1 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575589974619931|-2.871929601473837|2.910592122047989)
xbase=(0.956908426952151|-0.223192685411736|0.185772138940177)
ybase=(0.290383762668119|0.739602016998410|-0.607178826072894)
```

Camera 1

```
zerobase=(-0.778597957917036|-4.056784636774389|1.750794464613509)
xbase=(0.987733135923557|-0.131858214148564|0.083646061234565)
ybase=(0.153715319329572|0.915305948708384|-0.372272240251933)
```

Camera 2

```
zerobase=(1.114027088692416|-4.004703122091760|1.749757315907631)
xbase=(0.984150453654468|0.168657865306764|-0.054794242785153)
ybase=(-0.176954125088903|0.913725700006904|-0.365776684277734)
```

Camera 3

```
zerobase=(1.740024955214742|-2.674901339666130|2.965100675957225)
xbase=(0.950381972739412|0.218779286524235|-0.221155442346048)
ybase=(-0.310707426031989|0.702602768819795|-0.640164232563997)
```

Pattern 1 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575445790302574|-2.872135887914839|2.910447225041154)
xbase=(0.956917286448272|-0.223188827403992|0.185731134218949)
ybase=(0.290354634897052|0.739646694641500|-0.607138331107806)
```

Camera 1

```
zerobase=(-0.778849325383395|-4.056732937500927|1.750599398300870)
xbase=(0.987727029806168|-0.131905166380775|0.083644137106165)
ybase=(0.153756790055006|0.915316843289360|-0.372228324959994)
```

Camera 2

```
zerobase=(1.113777187717037|-4.004831961372911|1.749523674063356)
xbase=(0.984157769714482|0.168617640967492|-0.054786635831215)
ybase=(-0.176914633090776|0.913754617966548|-0.365723544147029)
```

Camera 3

```
zerobase=(1.739975858096254|-2.675079886816731|2.964939325507787)
xbase=(0.950384670560641|0.218782029372487|-0.221141134993889)
ybase=(-0.310698833059969|0.702645318606855|-0.640121700439817)
```

## Pattern 2

Pattern 2 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.563832566006765|-2.898763033146115|2.891251026972534)
xbase=(0.957579977950272|-0.222862659163819|0.182682295198349)
ybase=(0.288161594196870|0.744661011409663|-0.602032286274519)
```

Camera 1

```
zerobase=(-0.770978633614088|-4.065441137808236|1.728687613817932)
xbase=(0.987920396706211|-0.131124546198724|0.082581130749769)
ybase=(0.152482464376908|0.917527356438252|-0.367277345128607)
```

Camera 2

```
zerobase=(1.101939658475315|-4.015930605423455|1.727671061413589)
xbase=(0.984511530809156|0.167049371600791|-0.053212340224066)
ybase=(-0.174942042597556|0.916132380941330|-0.360689260060325)
```

Camera 3

```
zerobase=(1.728669985580827|-2.701650816875086|2.947771957159751)
xbase=(0.951094699331355|0.218744872761085|-0.218104455580172)
ybase=(-0.308515284125187|0.707876151723218|-0.635397256274845)
```

Pattern 2 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575686539618894|-2.871869394578661|2.910566090921686)
xbase=(0.956902041368725|-0.223199647842451|0.185796664198046)
ybase=(0.290404714521547|0.739596964198478|-0.607174960231993)
```

Camera 1

```
zerobase=(-0.778284496842051|-4.056885046709024|1.750737777191497)
xbase=(0.987740270359441|-0.131806426470472|0.083643435195077)
ybase=(0.153665946223064|0.915319447606536|-0.372259433465932)
```

Camera 2

```
zerobase=(1.113326929218579|-4.004996407402794|1.749700426539275)
xbase=(0.984172200355872|0.168549633612946|-0.054736651853686)
ybase=(-0.176832577562961|0.913756080313225|-0.365759572948337)
```

Camera 3

```
zerobase=(1.740012497739939|-2.674882930654526|2.965117437441925)
xbase=(0.950383865904262|0.218784398345909|-0.221142249397699)
ybase=(-0.310702279737881|0.702602811483488|-0.640166683497430)
```

Pattern 2 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575562278117679|-2.871974940050835|2.910553640090303)
xbase=(0.956909118606066|-0.223181555209699|0.185781947838026)
ybase=(0.290381370310412|0.739609968853796|-0.607170283980863)
```

Camera 1

```
zerobase=(-0.778377591649996|-4.056846836165488|1.750753132376731)
xbase=(0.987737237127850|-0.131816019619376|0.083664134266935)
ybase=(0.153682849011801|0.915315022946342|-0.372263335138035)
```

Camera 2

```
zerobase=(1.113338181428575|-4.004919082314031|1.749759232983824)
xbase=(0.984171637613570|0.168553946335732|-0.054733489676825)
ybase=(-0.176835250849633|0.913750361165440|-0.365772568035486)
```

Camera 3

```
zerobase=(1.739997663922718|-2.674841640626856|2.965156156691150)
xbase=(0.950385309211579|0.218781308368091|-0.221139103604368)
ybase=(-0.310698049098173|0.702596186155523|-0.640176008208071)
```

## Pattern 3

Pattern 3 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.565580607143472|-2.887829452855216|2.900278810139399)
xbase=(0.957467233531808|-0.222545871467357|0.183656831633157)
ybase=(0.288535881142253|0.742608489900989|-0.604383715880155)
```

Camera 1

```
zerobase=(-0.769427630631925|-4.062750531131957|1.738319393688736)
xbase=(0.987965678128467|-0.130728007034995|0.082668053180816)
ybase=(0.152211659239392|0.916701560283430|-0.369445341241782)
```

Camera 2

```
zerobase=(1.099618604658640|-4.013322350976206|1.737461194517856)
xbase=(0.984580905838328|0.166694797090658|-0.053040404240630)
ybase=(-0.174521875962207|0.915347480895824|-0.362878910972518)
```

Camera 3

```
zerobase=(1.729525173816146|-2.690735998703662|2.956427516981902)
xbase=(0.951030263131233|0.218261937240021|-0.218867917612787)
ybase=(-0.308719755843709|0.705734880194366|-0.637675772809162)
```

Pattern 3 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575640709568084|-2.871763142969595|2.910716819027031)
xbase=(0.956905594210283|-0.223199160558980|0.185778950622997)
ybase=(0.290393171705198|0.739572210310251|-0.607210631959351)
```

Camera 1

```
zerobase=(-0.778804104045255|-4.056720459927028|1.750909284299721)
xbase=(0.987728514803611|-0.131892485538470|0.083646597672751)
ybase=(0.153748049889374|0.915289942702973|-0.372298076736561)
```

Camera 2

```
zerobase=(1.113917547302135|-4.004790054149205|1.749821896288035)
xbase=(0.984154066873241|0.168631386274183|-0.054810840356958)
ybase=(-0.176935475386359|0.913722997124683|-0.365792457652682)
```

Camera 3

```
zerobase=(1.740145423541604|-2.674704338334041|2.965184274585309)
xbase=(0.950373059015784|0.218783009702169|-0.221190061625808)
ybase=(-0.310734269335448|0.702567495180836|-0.640189916020870)
```

Pattern 3 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575152555302692|-2.872095581745920|2.910612437138770)
xbase=(0.956938097731521|-0.223193389354701|0.185618393641603)
ybase=(0.290286617220454|0.739649484973600|-0.607167455683901)
```

Camera 1

```
zerobase=(-0.777850911204086|-4.057071583488879|1.750534797971885)
xbase=(0.987759265894899|-0.131787148113971|0.083449267406286)
ybase=(0.153572534719104|0.915355399161789|-0.372209577798655)
```

Camera 2

```
zerobase=(1.114658676136228|-4.004553580406213|1.749346000467134)
xbase=(0.984129464167338|0.168728091891004|-0.054954788372420)
ybase=(-0.177079633438142|0.913733618919783|-0.365696154037023)
```

Camera 3

```
zerobase=(1.740112647490903|-2.675434308893310|2.964586663935938)
xbase=(0.950370497914856|0.218763824902821|-0.221220038889662)
ybase=(-0.310737419831611|0.702699484888274|-0.640043506218965)
```

## Pattern 4

Pattern 4 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.585848501728198|-2.851609198290272|2.921761372357662)
xbase=(0.956300620857648|-0.223835727378217|0.188113502163163)
ybase=(0.292378911553675|0.736310374187351|-0.610214392605944)
```

Camera 1

```
zerobase=(-0.783674251709707|-4.051725402196079|1.760777788831858)
xbase=(0.987604122784911|-0.132435392506103|0.084255346832655)
ybase=(0.154550189252962|0.914243309787553|-0.374530919298145)
```

Camera 2

```
zerobase=(1.120489330062766|-3.998304438943845|1.759896853280713)
xbase=(0.983952551145908|0.169569858709475|-0.055528732303526)
ybase=(-0.178043226355728|0.912575070606639|-0.368113229939416)
```

Camera 3

```
zerobase=(1.750299948988758|-2.653742289914540|2.975311546409380)
xbase=(0.949708651422215|0.219305667295432|-0.223513985480305)
ybase=(-0.312764764484078|0.699043626852119|-0.643052260592847)
```

Pattern 4 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575542279842748|-2.871947871095999|2.910607121282523)
xbase=(0.956911218890604|-0.223185854894378|0.185765963879592)
ybase=(0.290374568028756|0.739603099987968|-0.607181904153089)
```

Camera 1

```
zerobase=(-0.778527948507075|-4.056759888684860|1.750828827460189)
xbase=(0.987734677135721|-0.131846491059664|0.083646341095930)
ybase=(0.153704792565998|0.915304570561226|-0.372279975088257)
```

Camera 2

```
zerobase=(1.114113905414708|-4.004603206820651|1.749811717856245)
xbase=(0.984147564847819|0.168671840940001|-0.054803108292749)
ybase=(-0.176970214289924|0.913717418068663|-0.365789588660495)
```

Camera 3

```
zerobase=(1.740052516501423|-2.674861830514449|2.965121250048743)
xbase=(0.950380549439749|0.218780693645429|-0.221160166698283)
ybase=(-0.310711801850253|0.702595538087180|-0.640170044636480)
```

Pattern 4 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575516621317521|-2.871959604548021|2.910594158562664)
xbase=(0.956912637638641|-0.223182667738111|0.185762484771574)
ybase=(0.290369885752105|0.739607895781772|-0.607178301610192)
```

Camera 1

```
zerobase=(-0.778777694523384|-4.056702866474636|1.750799407330046)
xbase=(0.987727904447975|-0.131882632866414|0.083669336814180)
ybase=(0.153746875722553|0.915299991545906|-0.372273855758578)
```

Camera 2

```
zerobase=(1.113682729882490|-4.004856158781900|1.749732064385321)
xbase=(0.984161329200136|0.168604906984907|-0.054761879516315)
ybase=(-0.176893102347471|0.913740759852732|-0.365768580014656)
```

Camera 3

```
zerobase=(1.739930503956779|-2.674865079704802|2.965158853503936)
xbase=(0.950388439482940|0.218768534853742|-0.221138287626806)
ybase=(-0.310688062610238|0.702600083360438|-0.640176577681752)
```

## Pattern 5

Pattern 5 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.585832472207345|-2.866448857326962|2.909578348053429)
xbase=(0.956338242137716|-0.224673831261922|0.186919330650369)
ybase=(0.292256974994442|0.738970566754749|-0.607048895920198)
```

Camera 1

```
zerobase=(-0.783791607008224|-4.056001462965594|1.748816502641015)
xbase=(0.987607882868626|-0.132738467299248|0.083732723559474)
ybase=(0.154551020109830|0.915345288346254|-0.371829242105922)
```

Camera 2

```
zerobase=(1.126244337157909|-4.001266647125484|1.747193881352745)
xbase=(0.983773010156121|0.170341576906578|-0.056341917484885)
ybase=(-0.179094599903696|0.913493020444451|-0.365315789263081)
```

Camera 3

```
zerobase=(1.752315923367318|-2.670919131531274|2.961658205558672)
xbase=(0.949604943383272|0.220302586798119|-0.222973589807171)
ybase=(-0.313057699994980|0.702126320684214|-0.639541637639973)
```

Pattern 5 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575683748542538|-2.871954486326752|2.910506967558487)
xbase=(0.956901933792450|-0.223197206639589|0.185800150840367)
ybase=(0.290405007317683|0.739608639754650|-0.607160597968743)
```

Camera 1

```
zerobase=(-0.778264034515384|-4.056929910696059|1.750696734806911)
xbase=(0.987740369627286|-0.131801288485670|0.083650359006758)
ybase=(0.153663556237867|0.915323542118847|-0.372250352218880)
```

Camera 2

```
zerobase=(1.113591515372344|-4.004904429281182|1.749704421910152)
xbase=(0.984164119691175|0.168596405204871|-0.054737899713251)
ybase=(-0.176876451196932|0.913747191436458|-0.365760565335115)
```

Camera 3

```
zerobase=(1.740011274144683|-2.674909887828305|2.965101343347264)
xbase=(0.950384237731992|0.218784285028856|-0.221140763529428)
ybase=(-0.310701096503998|0.702607207315230|-0.640162433183035)
```

Pattern 5 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575590536212983|-2.871877500708485|2.910626965535510)
xbase=(0.956908453465834|-0.223193562973610|0.185770948031496)
ybase=(0.290383705191242|0.739595014937671|-0.607187382641817)
```

Camera 1

```
zerobase=(-0.778480166686018|-4.056805302076969|1.750804995286245)
xbase=(0.987736100712992|-0.131839062708253|0.083641239184938)
ybase=(0.153695799788926|0.915308305708264|-0.372274504402933)
```

Camera 2

```
zerobase=(1.113231011723154|-4.005031593271610|1.749718838109988)
xbase=(0.984174901153192|0.168527919758429|-0.054754946831574)
ybase=(-0.176819106491128|0.913756480747451|-0.365765085091526)
```

Camera 3

```
zerobase=(1.739987371910207|-2.674874746515183|2.965129244739283)
xbase=(0.950383979990178|0.218772374276729|-0.221153654346335)
ybase=(-0.310701244226778|0.702599168631517|-0.640171184194626)
```

## Pattern 6

Pattern 6 with a maximum height of $10^{-3}$ mm
Camera 0

```
zerobase=(-1.586331430159645|-2.857697715276852|2.916863774458554)
xbase=(0.956296175611980|-0.224285169039869|0.187600073183373)
ybase=(0.292394513049564|0.737360164528470|-0.608937957846055)
```

Camera 1

```
zerobase=(-0.782301534881911|-4.054036016022997|1.756144178406239)
xbase=(0.987650832363647|-0.132377792716237|0.083797096172017)
ybase=(0.154290079480831|0.914717541153773|-0.373478769515933)
```

Camera 2

```
zerobase=(1.122446496993097|-3.999878432616416|1.754704867953680)
xbase=(0.983890099268688|0.169774725686028|-0.056007277025513)
ybase=(-0.178424948822941|0.912956546467432|-0.366980765027392)
```

Camera 3

```
zerobase=(1.751835727666755|-2.661511117434528|2.969111716070670)
xbase=(0.949617053158377|0.219744654475000|-0.223472009835347)
ybase=(-0.313026642428526|0.700365050954407|-0.641485086756093)
```

Pattern 6 with a maximum height of $10^{-6}$ mm
Camera 0

```
zerobase=(-1.575461962867955|-2.872199153928111|2.910390277389884)
xbase=(0.956915182695966|-0.223181290323650|0.185751029001634)
ybase=(0.290361417013456|0.739653936485855|-0.607126265081313)
```

Camera 1

```
zerobase=(-0.779216065479044|-4.056669454782327|1.750589406296889)
xbase=(0.987716551943935|-0.131957207967099|0.083685771080884)
ybase=(0.153820859068775|0.915306625991666|-0.372226978778859)
```

Camera 2

```
zerobase=(1.113088386815019|-4.005182777716766|1.749493477844768)
xbase=(0.984179749439386|0.168512818726217|-0.054714264309229)
ybase=(-0.176790731044650|0.913783485635266|-0.365711332607101)
```

Camera 3

```
zerobase=(1.739854423880928|-2.675097632726601|2.964998601216697)
xbase=(0.950393536348837|0.218774781300611|-0.221110201330939)
ybase=(-0.310672653056543|0.702647078400969|-0.640132475239546)
```

Pattern 6 with a maximum height of $10^{-9}$ mm
Camera 0

```
zerobase=(-1.575535739296647|-2.872047907651080|2.910513907379245)
xbase=(0.956912115927366|-0.223193667615473|0.185751955918407)
ybase=(0.290371648067824|0.739624546745431|-0.607157175573723)
```

Camera 1

```
zerobase=(-0.778910461142952|-4.056725447785156|1.750710292745737)
xbase=(0.987725623131050|-0.131912821159465|0.083648676164179)
ybase=(0.153766359118888|0.915305052373955|-0.372253365198722)
```

Camera 2

```
zerobase=(1.113538014629599|-4.004841319591987|1.749663077691398)
xbase=(0.984164982466490|0.168579486330639|-0.054774483801851)
ybase=(-0.176874338913479|0.913750227860913|-0.365754001096842)
```

Camera 3

```
zerobase=(1.739984836476170|-2.674982696395241|2.965033692243543)
xbase=(0.950384191117588|0.218776134769452|-0.221149026967094)
ybase=(-0.310700361701420|0.702622038301029|-0.640146511769947)
```

# List of Figures

# List of Tables

# Bibliography

[1] A. Wörnlein *Methods to measure optical properties of mirror facets for CTA.* Diploma Thesis, University of Erlangen-Nürnberg, 2012

[2] A. Schulz *Methods to measure characteristics of mirror facets of Imaging Atmospheric Cherenkov Telescopes.* Diploma Thesis, University of Erlangen-Nürnberg, 2010.

[3] Markus C. Knauer. *Absolute Phasenmessende Deflektometrie.* PhD thesis, Universität Erlangen-Nürnberg, 2006.

[4] E.Olesch, G.Häusler, A.Wörnlein, F.Stinzing, C. van Eldik. *Deflectometric measurements of large morrors.* Advanced Optical Technologies 3 (2014) 335-343.

[5] https://portal.cta-observatory.org/Pages/Home.aspx

[6] http://www.mitsuba-renderer.org/index.html

[7] http://www.ice.rwth-aachen.de/research/tools-projects/grace/ray-traycing/

# Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Erlangen, den 19. Dezember, 2014

Stefan Pickel