Institute for Computer Science XVII
Robotics

Master's thesis

Development and Calibration of a High Accuracy Structured Light Stereo Camera System for 3D Scanning

Christoph Liebender

September 2024

First reviewer:	Prof. Dr. Andreas Nüchter
Second reviewer:	Prof. Dr. Sergio Montenegro
Advisor:	Dr. Michael Bleier

Abstract

This work presents the development of a 3D scanner using stereo cameras and pattern projection. It begins with an overview of related research and places the work in the context of the “Mixed Media Scanning” research project, which proposes to solve the problem of 3D scanning through a water surface and correcting for refractive errors. This work’s first major part presents the theory behind camera calibration. For this purpose, we describe Zhang’s calibration method, which is suitable for determining the intrinsic parameters and reducing non-linear errors of cameras. The second part contains the definition and reappraisal of the *correspondence problem*, which is central to the triangulation of 3D points from multiple camera images. To solve the correspondence problem, we implement the *BICOS* method by utilizing a graphics card framework, to make structured light usable for solving the correspondence problem. This implementation allows for the real-time application of this principle on large, high resolution images. In the evaulation part, we calibrate two high-resolution cameras using the method of Zhang, to achieve a reprojection error of less than a third of a pixel. Finally, by quantifying the accuracy of the developed scanner based on industry standards, we show that the presented methodology yields a scanner that is able to reconstruct lengths and preserve shapes with sub-millimeter to millimeter accuracy. In addition to that, we compare scans of free-form scenes to reference scans to prove the scanner’s versatility.

Zusammenfassung

Diese Arbeit handelt von der Entwicklung eines 3D-Scanners mit Stereokameras und Musterprojektion. Sie beginnt mit einem Überblick über verwandte Forschung und stellt die Arbeit in den Kontext des Forschungsprojekts “Mixed Media Scanning”, in dem das Problem des 3D-Scannens durch eine Wasseroberfläche und der Korrektur von Brechungsfehlern gelöst werden soll. Der erste Hauptteil dieser Arbeit stellt die Theorie der Kamerakalibrierung vor. Zu diesem Zweck wird die Kalibrierungsmethode von Zhang beschrieben, die für die Bestimmung der intrinsischen Parameter und die Reduzierung nichtlinearer Fehler von Kameras geeignet ist. Der zweite Teil enthält die Definition und Aufarbeitung des *Korrespondenzproblems*, das für die Triangulation von 3D-Punkten aus mehreren Kamerabildern von zentraler Bedeutung ist. Zur Lösung des Korrespondenzproblems wird die *BICOS*-Methode mittels eines Grafikkarten-Frameworks implementiert, um strukturiertes Licht für die Lösung des Korrespondenzproblems nutzbar zu machen. Diese Implementierung ermöglicht die Anwendung dieses Prinzips in Echtzeit auf große, hochauflösende Bilder. Im Evaluierungsteil kalibrieren wir zwei hochauflösende Kameras mit der Methode von Zhang, um den Reprojektionsfehler auf weniger als ein Drittel eines Pixels zu reduzieren. Schließlich quantifizieren wir die Genauigkeit des entwickelten Scanners anhand von Industriestandards. Dies zeigt, dass die vorgestellte Methodik einen Scanner hervorbringt, der Längen und Formen mit einer Genauigkeit im Submillimeter- bis Millimeterbereich rekonstruieren kann. Darüber hinaus vergleichen wir Scans von Freiformszenen mit Referenzscans, um die Vielseitigkeit des Scanners zu beweisen.

Contents

1 Introduction **1**
- 1.1 State of the Art . . . 2
- 1.2 Project Context . . . 3
- 1.3 Contributions of this Work . . . 3

2 Camera Calibration **5**
- 2.1 Pinhole Camera . . . 5
  - 2.1.1 Euclidean Camera . . . 5
  - 2.1.2 Perspective Camera . . . 6
- 2.2 Brown Distortion Model . . . 6
- 2.3 Projection Matrix . . . 7
- 2.4 Stereo Pair . . . 9
  - 2.4.1 Epipolar Geometry . . . 9
  - 2.4.2 Stereo-Normal Case . . . 10
  - 2.4.3 Triangulation . . . 12
- 2.5 Zhang's Method . . . 12
  - 2.5.1 Planar Calibration Pattern . . . 12
  - 2.5.2 Initial Homography Estimation . . . 13
  - 2.5.3 Intrinsic Constraints . . . 13
  - 2.5.4 Solving $K, R, X_{\mathcal{O}}$ for $n \geq 2$ Views . . . 14
  - 2.5.5 Estimating Radial Distortion . . . 15
- 2.6 Determining Relative Orientation . . . 15
- 2.7 Calibration Patterns . . . 15
  - 2.7.1 Circles . . . 16
  - 2.7.2 Checkerboard . . . 16
  - 2.7.3 ArUco . . . 17
  - 2.7.4 ChArUco . . . 17

3 The Correspondence Problem **19**
- 3.1 Single-Shot . . . 19
  - 3.1.1 Block-Matching . . . 20
  - 3.1.2 Semi-Global Matching . . . 23
- 3.2 Multi-Shot . . . 26

3.2.1 Binary Features 27
3.2.2 Subpixel Accuracy 29
3.2.3 CUDA-based Binary Correspondence Search 30

4 Scanner Hardware **37**
4.1 Cameras 37
4.2 Trigger 37
4.3 Projector 37

5 Evaluation **39**
5.1 Calibration 39
5.2 VDI/VDE 2634 39
5.2.1 Sphere Probing Error 40
5.2.2 Plane Flatness Error 41
5.2.3 Sphere Spacing Error 41
5.3 Measurements 42
5.3.1 VDI/VDE Metrics 42
5.3.2 Free-Form Scans 44
5.3.3 Scans through a Water Surface 49

6 Conclusion **55**

List of Figures

2.1 Euclidean Pinhole Camera 6
2.2 Perspective Pinhole Camera: Scale Difference 7
2.3 Barrel and Pincushion Distortion 8
2.4 Epipolar Geometry 9
2.5 Stereo-Normal Case 11
2.6 Calibration Patterns 16

3.1 Principle of Stereo Block Matching 20
3.2 Stereo Block Matching on the New Tsukuba Dataset 21
3.3 Stereo Block Matching with Cross-Correlation 22
3.4 Multiple-Path Aggregation in Semi-Global Matching 26
3.5 Result of Semi-Global Matching 27
3.6 Principle of Multi-Shot 28
3.7 BICOS Binary Descriptor 28

4.1 Experimental Scanner Setup 38
4.2 Projected Speckle and Fringe Patterns 38

5.1 Example Calibration Images 40
5.2 Quiver Plot of Calibration Residuals 41
5.3 Sphere Probing, Plane Flatness and Sphere Spacing Errors 42
5.4 Achieved Probing and Flatness Errors 43
5.5 Achieved Sphere Spacing Errors 43
5.6 The Garden Gnome and his Ground Truth Scan 45
5.7 Scan of the Garden Gnome using a Fringe Pattern 46
5.8 Scan of the Garden Gnome using a Speckle Pattern 47
5.9 Garden Gnome Error Histogram 48
5.10 3D Scene: Adam & Eve 49
5.11 Scans of Adam & Eve using Semi-Global Matching 50
5.12 Scans of Adam & Eve using BICOS and a small Sequence Size 51
5.13 Scans of Adam & Eve using BICOS and a large Sequence Size 52
5.14 Adam & Eve Error Histogram 53
5.15 Scans through a Water Surface 53

Chapter 1

Introduction

Many motivators for 3D scanning exist. Indeed, it is a demanded technology in various industries: manufacturing, healthcare and remote sensing, to name a few. There, a high fidelity 3D scanner producing a high quality 3D model of a given object is required for many processes.

Depending on the scanning modality and measurement task, different methods of 3D scanning are possible: time-of-flight (ToF) laser scanning for terrestrial scans, acoustic systems for long-range underwater deployments and camera setups can be implemented. The latter method is possible by exploiting multiple viewpoints of a scene or object, since a change of viewpoints causes a perspective change where nearer objects are displaced further compared to the background of the image. Multiple cameras can be used by themselves, or, additional light can be projected into the scene, which makes them a flexible choice for different types of scanning conditions. In fact, cameras are the measurement tool of choice when it comes to mid- to close-range measurements underwater. There are multiple reasons for this: first of them being their abundance and therefore low cost, thus enabling quick development of new scanning setups. Second, since typical cameras capture light in the visible spectrum, they do not suffer from the pitfalls of infrared ToF scanning underwater, where infrared wavelengths are quickly absorbed by the enclosing medium. This doesn't apply to visible light to the same degree. Lastly, high resolution sensors are able to deliver a data density that is unmatched by other methods.

However, turbidity and bad lighting conditions are a hindrance of camera-based scanning. Therefore, it makes sense to illuminate a scene artificially, possibly with lighting systems directly attached to the scanner. This principle is also taken advantage of in another way: by projecting a light pattern, additional, visual structure is added to the scene. And while the human eyes may not benefit from visual "noise", computer algorithms tasked with creating a 3D reconstruction from multiple images benefit from this.

The issue at hand is to develop computer software that is able to create this 3D reconstruction, given some input images. Yet, ever increasing camera resolutions require ever more performant reconstruction algorithms. This sparks the issue of implementing the reconstruction algorithm in such a way that realtime-measurements are still feasible. Additionally, cameras project a distorted image of reality onto their sensors, to a degree which makes measurement with them unreliable. These distortions need to be corrected for. The work at hand now addresses both problems: Stereo camera calibration and the performance aware implementation of structured light 3D reconstruction via matching correspondences.

1.1 State of the Art

Narrow baseline stereo vision can generally be considered as a well-understood problem of computer- and machine vision. Initially, there is a need for calibrated cameras such that the camera geometry is known and straight lines in the real world also appear straight in a camera's image. This calibration process is typically achieved by the parametric method of Zhang [54], though more recent research indicates that generic camera models [41] can achieve better results. Images of a stereo pair where both cameras are calibrated individually then need to be rectified, for matching algorithms to be able to exploit the stereo geometry efficiently. This rectification requires the relative orientation of both cameras to be known, which can be expressed as the pose of the second camera relative to the first. The determination of this relative orientation as well as the calibration parameters can be achieved with software libraries like OpenCV [4]. Given both rectified images, correspondences need to be searched for in each image row. This correspondence search can be considered as the most important step in this process, as wrong or noisy correspondences will directly degrade the measurement result.

Semi-global Matching (SGM) [18] can be considered a baseline for stereo matching methods. Introduced in 2008, SGM searches multiple pixel paths inside a disparity range, for a minimum cost that is computed as a maximization of mutual information [47] between pixels. Since its proposal, many implementations have been made publicly available; Including a variation of SGM in OpenCV [4] and a GPU accelerated implementation in *libSGM* [31]. Dedicated hardware implementations also exist [33, 48].

According to the *ETH3D* [42] and *KITTI* [12] publications (2017, 2012), there are multiple methods that were considered state of the art when it comes to stereo matching. Yamaguchi *et al.* [49] combine both stereo and video pairs of a scene. Considering their use-case of autonomous driving, a video pair are two consecutive images of one side of the stereo pair. They then utilize the assumption that static scenes mostly consist of piecewise planar segments to create a dense depth map from the semi-dense output of SGM. SGM also appears in the original ETH3D evaluation with a variation based on matching Daisy [44] descriptors. Another approach of Geiger *et al.* [13] shows how well-designed preprocessing reduces the search space for efficient computation. Their preprocessing utilizes *support points* that are determined from robust matches between left and right views. These support points and their disparities compute to a triangle mesh, which then is used for minimizing an energy function based on the fact that disparities need to be in the proximity of an epipolar trace on the mesh.

Latest works referencing challenges like ETH3D and KITTI use learned methods for matching correspondences to achieve state of the art single-shot stereo performance on these datasets [21, 50]. However, Chang *et al.* [7] voice doubts about the apparent successes of learned methods: “due to the scarcity of sufficient labeled realistic training data existing state-of-the-art [...] networks usually are trained on synthetic data [...], which fail to generalize well to unseen realistic domains”.

Instead of enhancing the stereo matching process, other research focuses on working with *active* stereo sensors, where in addition to the camera system, there is some kind of projector to add visual structure into the scene. One approach is to project laser lines into the scene to either match points on these lines in both cameras [22], or to only have one camera with a calibrated laser projector [3]. Others project light patterns: either speckles or point patterns [14, 36, 37, 56] which can be projected at very high speeds; Up to 500 Hz [36] are possible. Vertical lines [16, 26, 40, 53] are another possibility for projection. Those variants are easily set up by placing a rotating wheel between a projector's light source and its optics [16]. Lastly, entirely coded patterns [1, 35, 51] help matching correspondences by assigning a unique code to each pixel. These varying light patterns lead to multi-shot approaches where a time-series of left and right views are used for correspondence search [8, 15].

1.2 Project Context

This work was conducted in the framework of the DFG research project *Mixed Media 3D Scanning* [30]. Its research question is how 3D scanning through media boundaries – the main application being a water surface – can be made possible, given that two cameras in this kind of setting observe a scene point through rays that each are refracted differently. The research proposal notes that means to enable 3D scanning of submerged structures already exist [20, 27, 45], given a planar water surface as the media boundary. Contrary, this problem is further complicated by a water surface that is perturbed by waves. Given this complexity, global matching methods do not provide a reliable solution. Local, pixel-wise matching methods able to match correspondences in real-time are required. Only by matching correspondences in real-time, a possibility to estimate the water surface exists, which in turn is necessary to correct for refraction to produce a metrically accurate 3D reconstruction.

1.3 Contributions of this Work

Initially, this work gives a short overview of camera calibration with a focus on stereo pairs, to then show how the correspondence problem is,

- already addressed by single shot algorithms, and
- solved in a more accurate way by introducing structured light projection.

This gets us to the main contributions of this work; Working with time-series of images increases the amount of data that needs processing. Thus, reducing this complexity is essential for real-time applications. This work reduces this complexity of row-wise exhaustive pixel-wise matching to a minimum by introducing the binary correspondence search (BICOS) of [8].

Additionally, we provide an efficient and versatile GPU-based implementation of said algorithm, that is able to:

- compute dense pointclouds from two 7 megapixel cameras, and
- doing this multiple times per second, while
- using an uncalibrated and unsynchronized aftermarket scanner for pattern projection.

Combined with a well-calibrated stereo setup, this gives us the ability to reconstruct a scene with sub-millimeter accuracy. This work's evaluation proves this by quantifying the scanner's performance based on industry standards. As an outlook, dense scans of a submerged structure prove how our implementation is able to find correspondences on rays that have been altered by refraction, attesting its versatility.

Chapter 2

Camera Calibration

Cameras are at the core of the development of a stereo vision measurement system. To describe the geometry of a camera, we first define the intrinsic parameters including the Brownian model of nonlinear distortions. Afterwards, we introduce the extrinsic parameters, which are necessary for both the formal definition of the projection of an image ray, and the relative orientation of a stereo pair required for triangulation. We then describe Zhang's method for camera calibration, followed by a short discussion on ways to determine the relative orientation of two cameras to then finish this chapter with a subset of possible calibration patterns required for these procedures.

2.1 Pinhole Camera

In its simplest form, a camera is described by the pinhole model. The pinhole model helps to understand how the camera's *intrinsic* parameters are defined.

2.1.1 Euclidean Camera

Figure 2.1 shows a pinhole camera model. Defined by Förstner & Wrobel [10] as the Euclidean camera, it includes three intrinsic parameters:

- The camera constant c,
- $x^s_{\mathcal{H}}$ as the x-, and
- $y^s_{\mathcal{H}}$ as the y-coordinate of the principal point $\mathcal{H}$.

The camera constant c is the distance between the projection center $\mathcal{O}$ and the principal point $\mathcal{H}$. For an euclidean camera, it is defined to be perpendicular to the sensor plane. Since a camera sensor's coordinates are unsigned and start at the top left at $(0, 0)$, the "projection" of $\mathcal{O}$ on the sensor plane has the positive coordinates $(x^s_{\mathcal{H}}, y^s_{\mathcal{H}})$. Units of these values are usually given in sensor pixels if the sensor size is unknown.

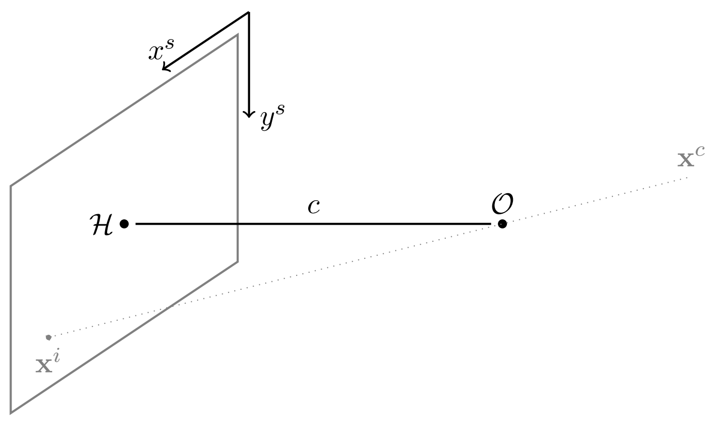


Figure 2.1: The euclidean pinhole camera. The projection center $\mathcal{O}$ lies in front of the principal point $\mathcal{H}$ with a distance of c (camera constant). $\mathcal{H}$ lies on the sensor plane, with sensor coordinates $y^s_{\mathcal{H}}, x^s_{\mathcal{H}} > 0$. A ray $\mathbf{x}^c$ passes through $\mathcal{O}$ and is projected onto the sensor plane as $\mathbf{x}^i$.

2.1.2 Perspective Camera

The two remaining intrinsic parameters stem from the physical properties of the imaging sensor. First, due to the layout of the pixels, the photosensitive area of each pixel needn't be square shaped. Thus, we need to account for a *scale difference*, shown in Figure 2.2: The scale difference m depends on the ratio between row and column pitch of the pixels on the pixel grid:[1]

$$1 + m = \frac{\mathcal{P}_y}{\mathcal{P}_x}. \tag{2.1}$$

This gives us the fourth intrinsic parameter. The fifth intrinsic parameter is the *shear*, denoted as s. It is caused by any deformations of the whole image sensor: namely the deviation from its ideal rectangular shape towards a rhomboid. [10]

As this deviation is negligible in practice, we don't consider s during the calibration process. However, it is included in the following formal definitions of the camera matrix nonetheless.

2.2 Brown Distortion Model

Pinhole cameras are suitable for describing straight-line preserving cameras. Realistically however, lenses, unaligned optics as well as non-planar sensor surfaces cause nonlinear distortions which need to be accounted for in a different way.

The Brown distortion model [5, 6, 10] models these distortions as *radial* and *tangential*

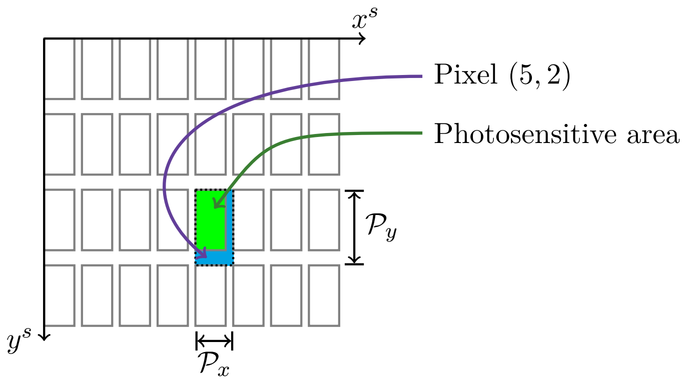


Figure 2.2: Scale difference in a perspective camera. Multiple pixels are arranged in a grid on the image sensor plane. Row pitch is indicated as $\mathcal{P}_y$, column pitch as $\mathcal{P}_x$. The scale difference is computed with their ratio. The photosensitive area of pixel at pixel coordinate $(5, 2)$ is colored green; Its total pixel area is the union of green and blue areas. Figure adapted from [10].

distortions by an additive delta for both image coordinates:[1]

$$\Delta_x = \bar{x}\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots\right) + \left[2p_1\bar{x}\bar{y} + p_2\left(r^2 + 2\bar{x}^2\right)\right]\left[1 + p_3 r^2 + \cdots\right], \quad (2.2)$$

$$\Delta_y = \underbrace{\bar{y}\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots\right)}_{\text{radial}} + \underbrace{\left[p_1\left(r^2 + 2\bar{y}^2\right) + 2p_2\bar{x}\bar{y}\right]\left[1 + p_3 r^2 + \cdots\right]}_{\text{tangential}}, \quad (2.3)$$

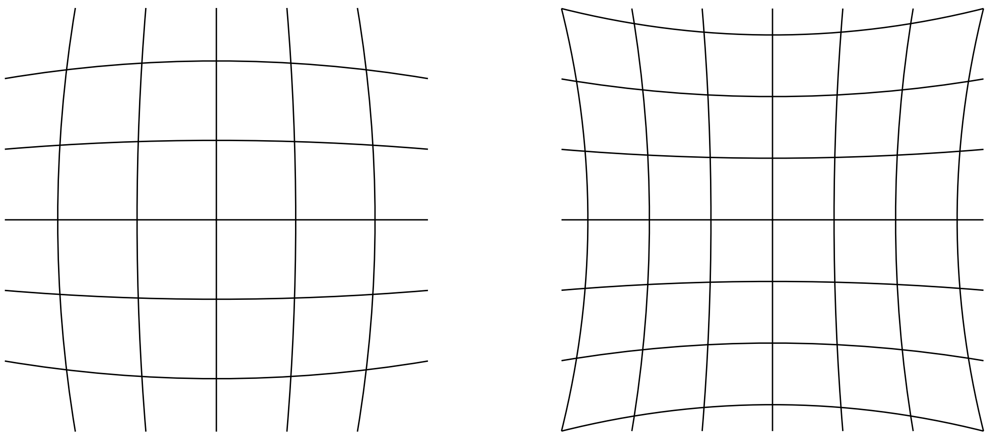
with $\bar{x} = x^s - x^s_{\mathcal{H}}$ ($\bar{y}$ analog.) and the radial distance from the principal point $r = \sqrt{\bar{x}^2 + \bar{y}^2}$. The *distortion coefficients* $k_{1\ldots n}$ and $p_{1\ldots n}$ are to be determined by the calibration process.

The radial parts of both deltas indicate that distortions are proportional to the distance of a pixel from the image center. Visually, this fact manifests as either *barrel-*, *pincushion-distortion* or mixed forms thereof. Shown in Figure 2.3, the distortion is larger at the edges of the image. It is clear that these distortions make straight lines in the scene appear curved in the resulting image. This radial distortion is introduced by the lens system of the camera in the first place [10]. On the other hand, the reason for tangential distortion is mainly a falsely centered lens system [5]. Given a sufficiently accurate set of distortion parameters, we can correct for these errors and make accurate measurement of the scene possible.

2.3 Projection Matrix

To combine both previous sections into an actual mathematical definition that describes the relation between some 3D point in the scene and its 2D pixel coordinate projection in the sensor coordinate system, we lastly need to define the extrinsic parameters of a camera. These are

[1]These equations are equal to the definitions in [6, 10] with the exception that x, y coordinates have been swapped, to match the coordinate system definition in Figure 2.1. $\Delta_{x,y}$ are thus analogous to the definitions in the OpenCV documentation "Camera Calibration" from `https://docs.opencv.org/4.9.0/dc/dbb/tutorial_py_calibration.html`.



(a) Barrel distortion **(b)** Pincushion distortion

Figure 2.3: Radial distortion. On the left, barrel distortion causes pixels further away from the principal point to be distorted inwards. The opposite applies on the right for pincushion distortion. (Figures courtesy of WolfWings, Wikipedia; Public domain.)

simply given as the six-dimensional pose $(R, X_{\mathcal{O}})$ of the projection center $\mathcal{O}$ and the orientation of the camera coordinate system in the same frame as the 3D points of the scene.

Combining, we get the general projection matrix as defined by Förstner & Wrobel [10]:

$$P = \underbrace{\begin{bmatrix} c & cs & x^s_{\mathcal{H}} + \Delta_x \\ 0 & c(1+m) & y^s_{\mathcal{H}} + \Delta_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Camera Matrix } K} \cdot \underbrace{R\,[I_3 \mid -X_{\mathcal{O}}]}_{\mathbb{R}^{3\times 4}}. \tag{2.4}$$

Note that $\Delta_{x,y}$ implicitly depend on coordinates in the ideal camera system. Because of this, the projection needs to be split into multiple steps [10]. For this, we first transform the 3D scene point into a ray through the camera projection center $\mathcal{O}$:

$$\mathbf{x}^c = R\,[I_3 \mid -X_{\mathcal{O}}]\,\mathbf{X} \tag{2.5}$$

where both $\mathbf{x}, \mathbf{X}$ are homogenous coordinates of their cartesian counterparts x, X via:

$$X = [x\ y\ z]^T \mapsto \mathbf{X} = [X \mid 1]^T, \tag{2.6}$$

$$\mathbf{x} = [u\ v\ w] \mapsto x = \frac{1}{w}[u\ v]^T. \tag{2.7}$$

This mapping is necessary to obtain the cartesian coordinates of $\mathbf{x}^i$ that are used in the determination of the distortion $\Delta_{x,y}$:

$$\mathbf{x}^i = \operatorname{diag}(c, c, 1)\mathbf{x}^c. \tag{2.8}$$

With an initialized K from Equation 2.4, we simply project the ray onto the sensor plane:

$$\mathbf{x} = K\mathbf{x}^c, \tag{2.9}$$

which again can be mapped to cartesian sensor pixel coordinates using Relation 2.7.

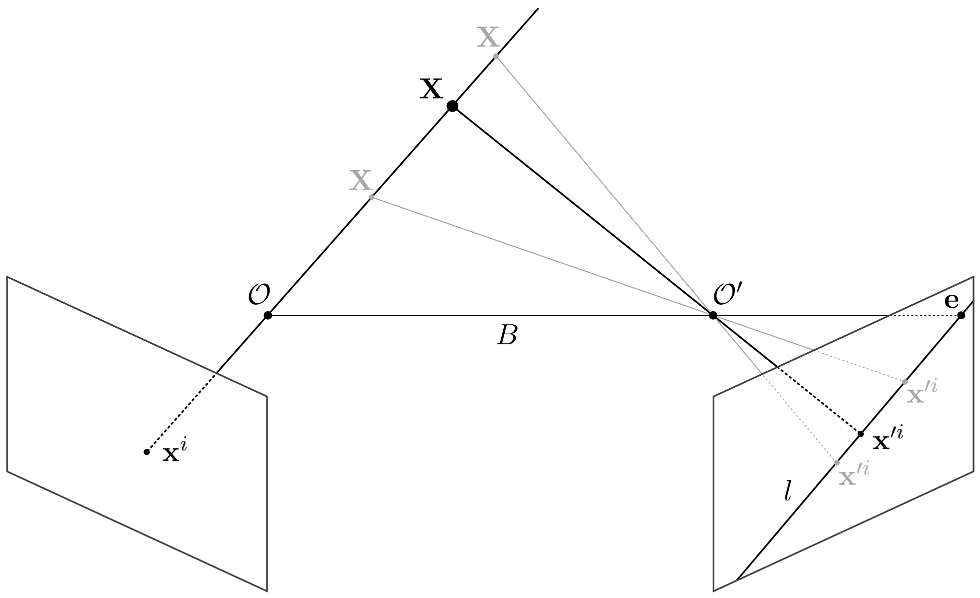


Figure 2.4: Epipolar geometry. Two cameras are placed side by side, with their projection centers $\mathcal{O}$ (left) and $\mathcal{O}'$ (right). Both projection centers are separated by their baseline B. The 3D scene point $\mathbf{X}$ gets projected onto the left plane as $\mathbf{x}^i$. The projection center $\mathcal{O}$ of the left view gets projected onto the right image plane as its epipole $\mathbf{e}$. Together with the projection of $\mathbf{X}$ onto the right plane, $\mathbf{e}$ forms the epipolar line l.

2.4 Stereo Pair

The projection with P of Equation 2.4 maps a 3D point onto a 2D plane. This mapping is not fully invertible. Given a point on the sensor plane of a camera, we may only reconstruct its ray, but not the position on it. To actually be able to reconstruct a 3D scene from images, we need additional views of the scene. This can either be achieved by moving the camera to a different viewpoint and capturing the static scene again, or by adding a second camera which view is offset to the first by some baseline.

2.4.1 Epipolar Geometry

Figure 2.4 shows the epipolar geometry of a stereo system. Starting with the left view, we have the case of Figure 2.1 where some scene point is projected onto the sensor plane. Now, given only the projected point $\mathbf{x}^i$ in the left frame, we can utilize the second view from the right to determine the actual position of $\mathbf{X}$ on the ray. For this, we need to find the correspondence in the right image. Figure 2.4 indicates that this correspondence must lie on the epipolar line l through the epipole $\mathbf{e}$ in the right image. This is possible because of the *coplanarity constraint* where both camera centers and the scene point $\mathbf{X}$ form a plane. [10]

We now define the relation between the projected point on the left frame and it's corresponding epipolar line in the right frame.

Fundamental- & Essential Matrix

As of [10], the coplanarity constraint can be expressed as the scalar triple product of three lines between $x^i, x'^i, \mathcal{O}$ and $\mathcal{O}'$:

$$\det \begin{bmatrix} \mathcal{O}x^i & \mathcal{O}\mathcal{O}' & \mathcal{O}'x'^i \end{bmatrix} = 0, \tag{2.10}$$

which is equivalent to:

$$(x^i)^T \underbrace{K^{-T} \overbrace{R S_b (R')^T}^{E} (K')^{-1}}_{F} x'^i = 0, \tag{2.11}$$

with S_b as the skew-symmetric matrix of $b = B = \mathcal{O}' - \mathcal{O}$:

$$S_b = \begin{bmatrix} 0 & -b_z & b_y \\ b_z & 0 & -b_x \\ -b_y & b_x & 0 \end{bmatrix}. \tag{2.12}$$

Equation 2.11 yields the *fundamental matrix* F that, premultiplied with a left image point x^i, gives us a line through the epipole $\mathbf{e}$:

$$(x^i)^T F = l^T. \tag{2.13}$$

Similarly, the *essential matrix* E in Equation 2.11 expresses the coplanarity constraint, albeit assuming calibrated cameras. Both of these matrices are determined during the calibration process.

2.4.2 Stereo-Normal Case

As seen in Figure 2.4, the epipolar lines computed by Equation 2.13 are oblique with respect to the sensor's x^s-axis. Even though it is possible to search for matches on this oblique line, it is ideally horizontal. One motivation behind this is computational efficiency: since images are usually laid out as a contiguous array in memory with concatenated image rows, cache misses are reduced by only accessing horizontally neighboring pixels.

We can *rectify* both views into the *stereo normal* case, as depicted in Figure 2.5. There, both images I, I' are transformed into parallel, stereo normal views, depicted in gray. Both viewing directions are made equal, only separated by a baseline. This transformation from the image coordinate system x, y to the respective normalized one x^n, y^n is possible via homographies computed as

$$H_m^{-1} = K^n R^n R^T K^{-1}, \tag{2.14}$$

$$H_m'^{-1} = K^n R^n R'^T K'^{-1}, \tag{2.15}$$

for each view, such that:

$$\mathbf{x}^n = H_m^{-1} \mathbf{x}^i. \tag{2.16}$$

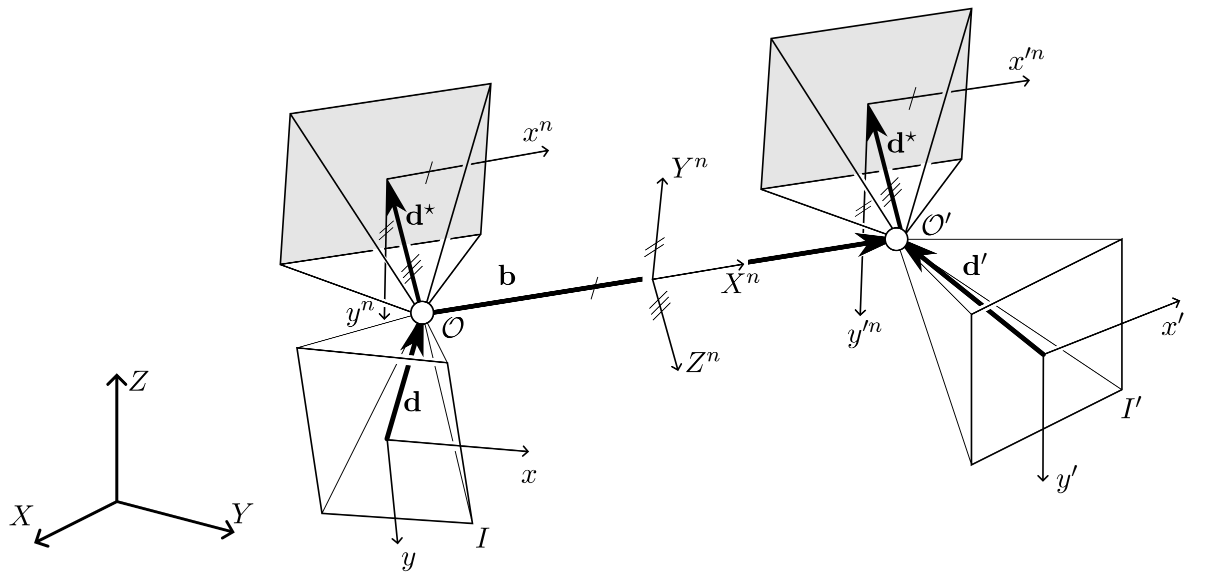


Figure 2.5: Stereo-normal case. Both images I and I' with their image coordinate systems x, y separated by a baseline $\mathbf{b}$ are rectified into a the stereo normal case x^n, y^n, such that the viewing directions $\mathbf{d}^\star$ are equal. In the rectified position, the camera constant is chosen to be negative, which causes the images to be placed in front of the projection center $\mathcal{O}$. The x^n-axes are parallel to the baseline vector $\mathbf{b}$. Figure adapted from [10].

K^n, R^n are common calibration and rotation matrices of the stereo normal pair, which guarantee that the resulting image content is only different w.r.t. the image x-coordinates. These matrices are obtained via:

$$K^n = \mathrm{diag}(c, c, 1) \tag{2.17}$$

for the calibration matrix, with c to be chosen as one of the camera's constant, and

$$R^n = \Big[\|\mathbf{b}\| \quad \|\mathbf{b} \times \mathbf{d}^\star\| \quad \|\mathbf{b} \times (\mathbf{b} \times \mathbf{d}^\star)\, \| \Big]^T \tag{2.18}$$

as the rotation matrix deducted by the geometric constraints shown in Figure 2.5. We choose the viewing direction $\mathbf{d}^\star$ as the average between the individual viewing directions $\mathbf{d}, \mathbf{d}'$. [10]

2.4.3 Triangulation

Given two rectified images, we can *triangulate* a 3D point given its coordinates in both left and right images. The canonical solution as of [10] computes the 3D coordinates via:[1]

$$Z = c\frac{|\mathbf{b}|}{-p_x} \tag{2.19}$$

$$Y = -\frac{y^n + y'^n}{2}\frac{Z}{c} \tag{2.20}$$

$$X = x^n \frac{Z}{c} \tag{2.21}$$

with p_x as the x-*parallax* or *disparity* between a point in both images as $p_x = x'^n - x^n$. Because we eliminate the y-parallax by rectification, we effectively have $y^n = y'^n$, and thus, the Y-coordinate is computed based on solely the left view's y^n. Another approach is to reformulate these equations as a matrix:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -x^s_{\mathcal{H}} \\ 0 & 1 & 0 & -y^s_{\mathcal{H}} \\ 0 & 0 & 0 & c \\ 0 & 0 & -\frac{1}{|\mathbf{b}|} & \frac{x^s_{\mathcal{H}} - x'^s_{\mathcal{H}}}{|\mathbf{b}|} \end{bmatrix}. \tag{2.22}$$

This is a disparity-to-depth mapping matrix computed by OpenCV [4], used via:

$$\mathbf{X} = Q\,[x \; y \; p_x \; 1]^T . \tag{2.23}$$

Note that here, x, y are upper-left centered coordinates in the rectified image. The cartesian coordinates of the 3D point can be computed analogous to Equation 2.7.

2.5 Zhang's Method

Now that we know about the unknowns which are to be known, we need to introduce a way to determine them. A popular method introduced by Zhang [54] uses planar patterns. It is regarded as the standard method to calibrate a camera's intrinsic parameters and radial distortion [55].

2.5.1 Planar Calibration Pattern

Referring to Equation 2.4, we have the mapping of a homogenous 3D coordinate $\mathbf{X}$ to its image coordinate $\mathbf{x}$ via the projection matrix P. This matrix is a 3×4 homography. Since the calibration pattern is affixed – or ideally: printed – on a planar surface, the Z-coordinate of each feature

[1]The sign of the Y-coordinate is flipped w.r.t. [10], to stay consistent with Figure 2.5.

point can be set to zero:

$$\mathbf{x} = \underbrace{K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{X}}, \tag{2.24}$$

with r_i as the i-th columns of R and $t = -RX_{\mathcal{O}}$. This reduces P to a 3×3 homography:

$$H = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}. \tag{2.25}$$

This "removal" of the Z-coordinate effectively defines the planar pattern as the origin of the world coordinate system for each view.

2.5.2 Initial Homography Estimation

Given the 3D points on the plane and their 2D image coordinates, it is possible to estimate H by minimizing:

$$\min_H \sum_i \|x_i - \hat{x}_i\|^2 \tag{2.26}$$

with x_i as the i-th sensor coordinate of a feature point and:

$$\hat{x}_i = \frac{1}{h_3^T \mathbf{X}_i} \begin{bmatrix} h_1^T \mathbf{X}_i \\ h_2^T \mathbf{X}_i \end{bmatrix} \tag{2.27}$$

as the reprojection of the 3D coordinate with h_i as the i-th row of H. However, reprojection and iterating Expression 2.26 requires an initial guess. Reformulating Equation 2.25 with Z-less homogeneous coordinates as:

$$\begin{bmatrix} \mathbf{X}^T & 0_{1\times 3} & -x^s \mathbf{X}^T \\ 0_{1\times 3} & \mathbf{X}^T & -y^s \mathbf{X}^T \end{bmatrix} \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}}_{\mathbf{z}} = 0 \tag{2.28}$$

yields the initial guess for H via singular value decomposition.

2.5.3 Intrinsic Constraints

Until now, we can estimate one homography per view of the calibration pattern. This homography implicitly contains the intrinsic parameters of the camera. To obtain the intrinsic parameters, Zhang first takes advantage of two constraints deduced from the fact that the reduced 3×3

homography of Equation 2.25, written as:[2]

$$\begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{2.29}$$

has orthonormal rotation matrix columns r_i:

$$\|r_i\| = 1, \tag{2.30}$$

$$r_1^T r_2 = 0. \tag{2.31}$$

Explicitly, we get for 2.30:

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \tag{2.32}$$

and for 2.31:

$$h_1^T K^{-T} K^{-1} h_2 = 0 \tag{2.33}$$

the two basic intrinsic constrains which can be used to setup a closed-form solution of the calibration parameters.

2.5.4 Solving $K, R, X_{\mathcal{O}}$ for $n \geq 2$ Views

We can rewrite $K^{-T}K^{-1}$ as a symmetric matrix:

$$V = K^{-T} K^{-1} \equiv \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{12} & V_{22} & V_{23} \\ V_{13} & V_{23} & V_{33} \end{bmatrix}, \tag{2.34}$$

with its vector representation:

$$v = \begin{bmatrix} V_{11} & V_{12} & V_{22} & V_{13} & V_{23} & V_{33} \end{bmatrix}^T. \tag{2.35}$$

Similarly, H can be vectorized as:

$$u_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{h2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}, \tag{2.36}$$

with $[h_{i1}\ h_{i2}\ h_{i3}]^T = h_i$. Thus, we can express $h_i^T K^{-T} K^{-1} h_j$ as:

$$h_i^T V h_j = u_{ij}^T v \tag{2.37}$$

[2]The right side introduces an additional λ; This is because homographies are defined only up to a scalar.

to reformulate Equations 2.32 and 2.33 via:

$$\underbrace{\begin{bmatrix} u_{12}^T \\ (u_{11} - u_{22})^T \end{bmatrix}}_{U} v = 0, \tag{2.38}$$

which, again, can be solved via singular value decomposition, given at least two distinct views of the calibration pattern (with $s = 0$). The intrinsic parameters defined in Section 2.3, as well as the extrinsic parameters for each individual view of the calibration pattern can be extracted from the singular vector of U with smallest singular value. This however does not apply to the camera system's radial distortion.

2.5.5 Estimating Radial Distortion

To estimate radial distortion, Zhang first notes that most of the radial distortion parameters defined in Section 2.2 are negligible because of sensor quantization, and not helpful from the perspective of numerical stability. Because of this, his method described in [54] only accounts for the first two terms of distortion: $k_{1,2}$.

Starting with n images, m points on the calibration plane, and initial guesses of $K, R, X_{\mathcal{O}}$ from the previous section, the complete set of parameters including the radial distortion terms can be found by minimizing:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \|x_{ij} - \hat{x}(K, k_1, k_2, R_i, X_{\mathcal{O}_i}, \mathbf{X}_j)\|^2 , \tag{2.39}$$

with $\hat{x}$ as the reprojection of $\mathbf{X}_j$ depending on all parameters, and initial guesses of $k_{1,2}$ as zero.

2.6 Determining Relative Orientation

To determine the relative orientation of two cameras introduced in Section 2.4.1, we can make use of the same 3D points on the plane of our calibration pattern under the assumption that they are visible in both left and right images. Said section describes the essential matrix E as the epipolar constraint for two calibrated cameras. Since we can already calibrate each camera individually, we are able to undistort the detected set of image points as they are detected in sensor coordinates. Given a set of minimum 5 points, ways to solve for E exist; OpenCV [4] implements the five point algorithm of Nistér [28]. Another approach implemented in OpenCV – the one used in our evaluation – minimizes the total reprojection error of the 3D points, with an option to further optimize each cameras intrinsic parameters. This minimization makes sense, because in practice, a high number of input images for calibration yield many more points than just five, limiting the usability of the five point algorithm.

2.7 Calibration Patterns

Zhang's method described in Section 2.5 requires a planar calibration pattern. In practice, they are ideally printed onto glass for the pattern to fulfill the property of being planar. Common

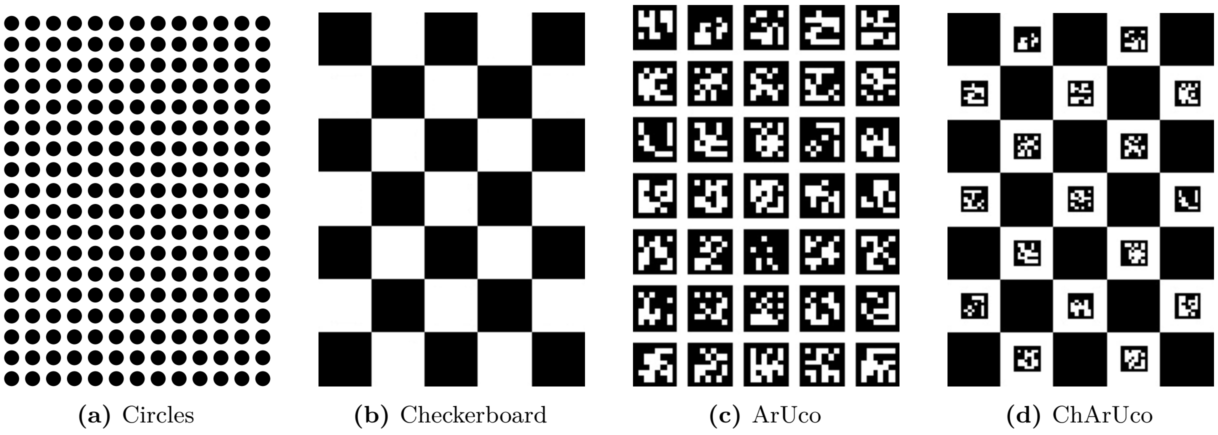


Figure 2.6: Different calibration patterns. Circles and checkerboard patterns arrange circles and black squares in a grid. ArUco patterns employ markers, where each marker has a code associated with them. ChArUco patterns combine checkerboard and ArUco patterns.

calibration patterns are shown in Figure 2.6.

OpenCV [4] comes with functions to automatically extract these patterns. Generally, these extraction algorithms first binarize the input image for it to consist only of black and white pixels. Then, a pattern-specific method is used to extract coordinates of the pattern's features. We now briefly review some of these pattern-specific methods.

2.7.1 Circles

Figure 2.6a shows a circle grid. The center of each circle is defined as the pattern's 3D coordinate. In OpenCV [4], these circles are detected using an algorithm to extract blobs, which extracts the circles' contours to then compute the center point of each contour.

Mallon and Whelan [24] show that circle patterns are unfit for bias-free calibration, due to the perspective transform of the camera system and especially its nonlinear distortion. Their findings on these patterns stand in contrast to those on checkerboards. However, ways of correcting for these biases exist [34].

2.7.2 Checkerboard

Figure 2.6b shows a checkerboard pattern, widely used for camera calibration. Its features are the inner corners of the checkerboard. These corners are first detected using corner detectors and then refined for subpixel accuracy. In [24], two methods are described:

The first refinement method is to calculate derivatives in a small region around an initial guess, to then fit lines into these derivatives. This method still suffers from nonlinear distortion, as lines are affected by nonlinear distortion. A second variant blurs the region of interest to then fit a quadratic function. The corner point is then computed as the function's saddle point [23], being invariant to perspective projections and distortions.

Evolutions of the checkerboard patterns are star patterns, as used in [41].

2.7.3 ArUco

A downside to the two previously mentioned patterns is that their detection is hindered by obstructions. This is due to the fact that a single circle – or corner – cannot be uniquely associated without its position relative to its neighbors.

Figure 2.6c shows a grid of ArUco markers. These markers solve this issue by assigning a unique binary code to each of the squares. Introduced by Garrido-Jurado *et al.* [11], ArUco markers are made to be robustly detected and identified, even when neighboring markers are occluded. This is especially useful when calibrating a stereo pair. There, the calibration pattern may only be partially visible in either left and right views. Being able to identify each marker enables matching of pairs between both views.

2.7.4 ChArUco

Compared to checkerboard corners, corners of ArUco markers cannot be refined as accurately. To benefit from both high accuracy and corners being identifiable, checkerboard patterns and ArUco markers can be combined to form ChArUco patterns, shown in Figure 2.6d. Two neighboring ArUco markers identify a checkerboard corner that lies between them.

Chapter 3

The Correspondence Problem

In the previous chapter, we described how a given pixel of a camera's sensor maps onto a ray in 3D space, and how we can triangulate 3D points from their 1D disparity between two rectified images. This chapter will introduce the *correspondence* or *matching* problem, which needs to be solved to actually obtain the disparity used in triangulation.

Over the years, many approaches have been developed. Scharstein and Szeliski [38, 39] define a stereo vision taxonomy as follows: Starting with a region in the reference image, its matching cost is computed against every possible match in the match image, based on some similarity constraint. Then, the initial 3D $w^2 \times h$ cost volume of 2D $w \times h$ reference and match images is optionally aggregated or modified to consider a spatial or temporal region around a reference pixel. The third step is categorized into either local or global disparity optimization. Local methods choose the disparity with lowest cost at a given pixel. Global methods optimize an energy function to incorporate smoothness assumptions. The fourth and last step is motivated by the fact that pixel-based computations are integer only, yielding jagged steps in the resulting disparity image. Thus, it makes sense to fit a curve into the costs and compute the best match based on values on that curve.

The aforementioned distinction between spatial and temporal match regions actually motivates different ways to do stereo vision. Spatial methods work well on *single-shot* stereo vision, where only one image each is recorded and matched. Temporal methods require *multi-shot* stereo vision. Here, multiple images per camera are used to consider the change of a single pixel over time.

We now look at either of these ways, starting with single-shot methods.

3.1 Single-Shot

Single-shot stereo vision uses a single pair of images, which is why algorithms can only utilize the spatial properties of images, not regarding any temporal changes. Since single-shot stereo is realizable with a simple synchronized camera setup, it can be considered the main focus of stereo vision research [39]. We now introduce block-matching to give an understanding on how simple, block- and epipolar-line based matching is realized. Then, we review semi-global-matching as a more advanced pixel-based method, which we will use in the remainder of this work as a single-shot implementation.

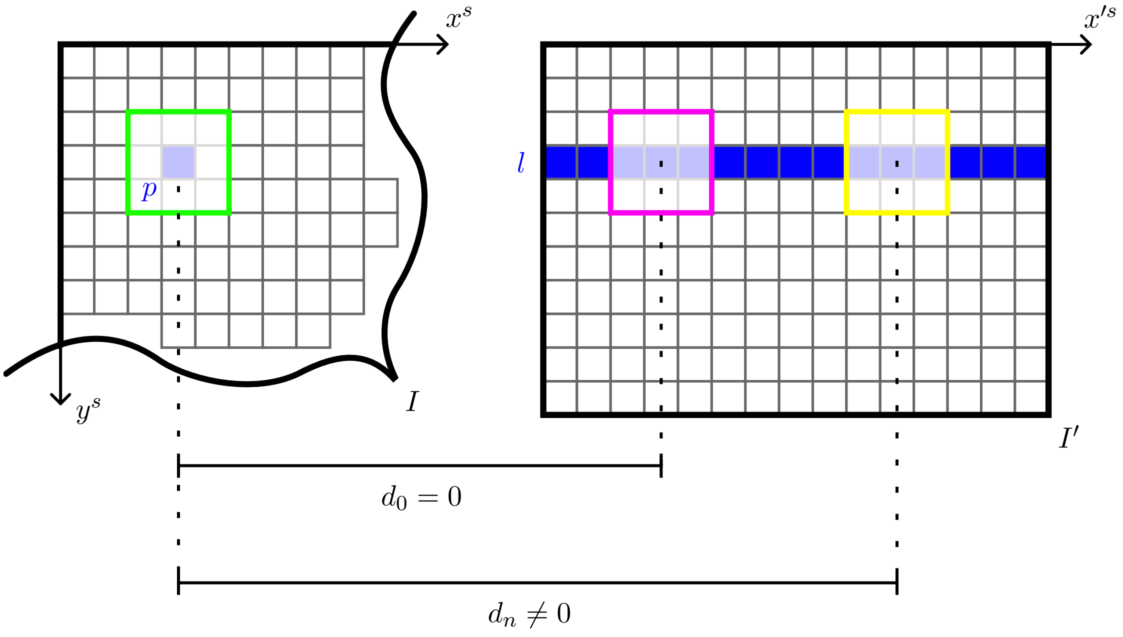


Figure 3.1: Stereo block matching. A reference image I with its pixel grid on the left highlights a pixel p (in blue) and its 3×3 block (in green). On the right, its match image I' shows the corresponding epipolar line l (blue) of p. On that line, the block of p is compared to any block on l. In case $x^s = x'^s$ (pink), the disparity d computes to zero. In any other case (yellow), the disparity computes as the difference of the two.

3.1.1 Block-Matching

One way to do single-shot stereo vision is by matching square sized *blocks*. Figure 3.1 shows the principle of *stereo block matching* (BM). Square blocks around reference pixels in the left image are compared to blocks around candidate pixels of the right image. The square size n of these blocks must be odd for a central pixel to be defined. To compare two blocks, different metrics can be used.

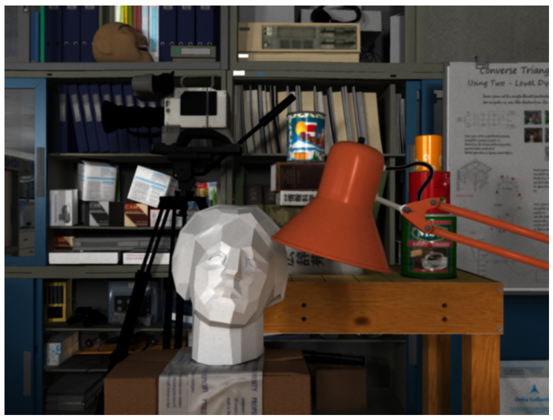
Sum of Absolute Differences

One simple approach is the *sum of absolute differences*:

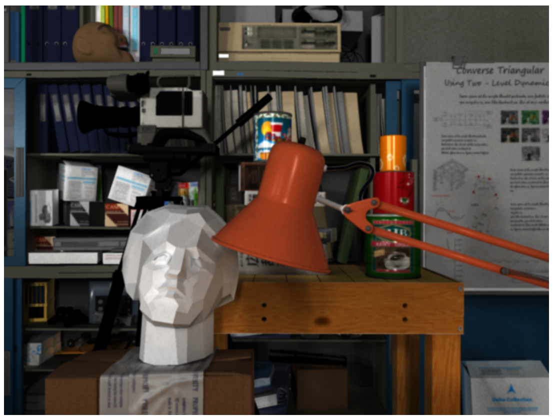
$$\mathrm{SAD}(x, y, d) = \sum_{x,y \in b} |I(x, y) - I'(x + d, y)| \tag{3.1}$$

where each pixel at coordinates x, y in a reference block b is compared to its corresponding pixel in the candidate block at disparity d such that a good match minimizes the SAD. Then, the disparity of pixel p can be set to:

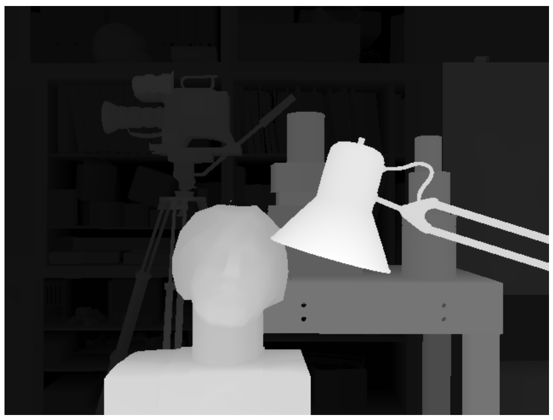
$$d(x, y) = \arg\min_d \mathrm{SAD}(x, y, d). \tag{3.2}$$



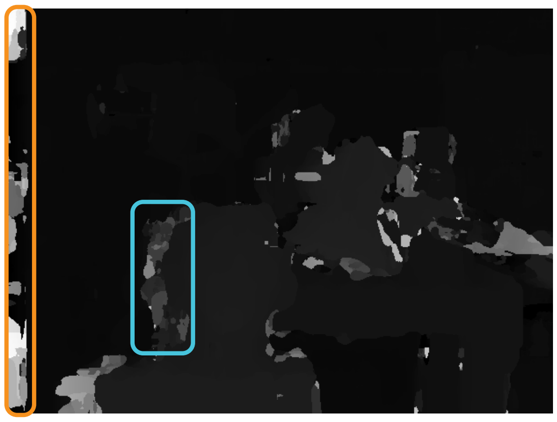
(a) Left view



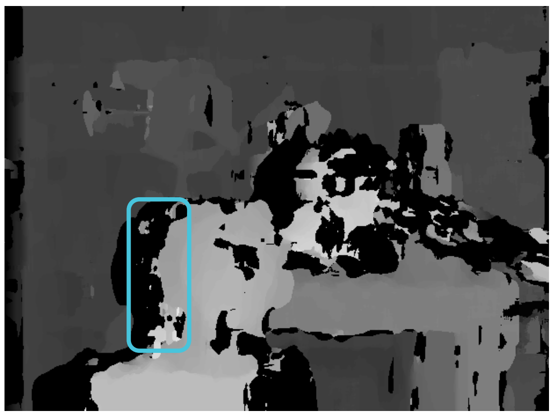
(b) Right view



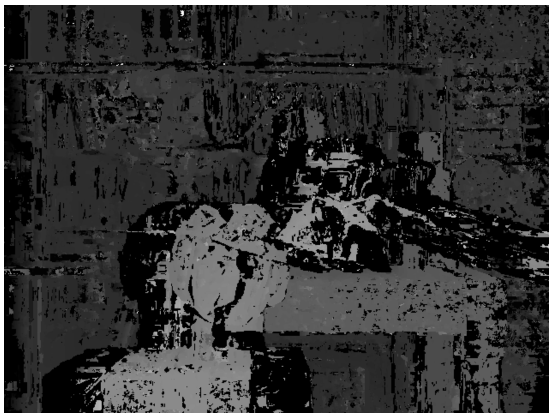
(c) Ground truth disparity



(d) Naive SAD implementation



(e) SAD with left-right check and disparity limit



(f) SAD of (e) with a blocksize of 5

Figure 3.2: Stereo block matching on the New Tsukuba dataset. Figures (a, b) show the stereo view of the scene. Figure (c) displays the ground truth disparity map, where brighter pixels indicate higher disparities. Figure (d) is the result of a naive SAD implementation, without any pre- or postprocessing, with a blocksize of 21. Figure (e) is the postprocessed result of (d) with $\theta_c = 22$ and $\max(d) = 128$, while Figure (f) varies the result of (e) by reducing the blocksize. For display, disparity ranges are normalized to $[0, 255]$.

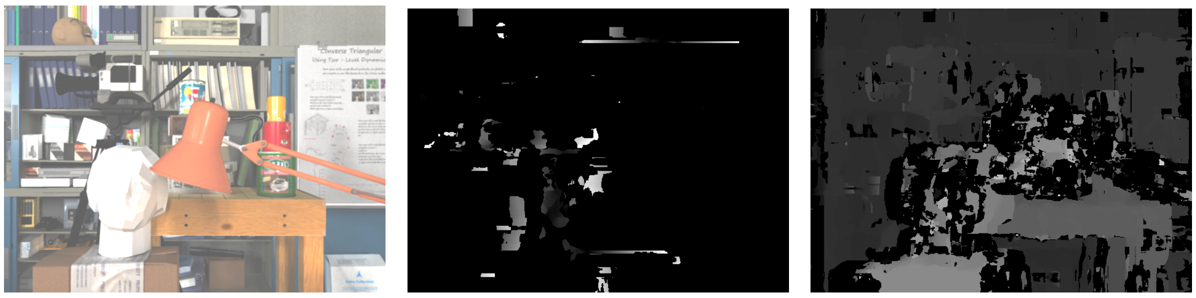

(a) Increased contrast & brightness **(b)** SAD **(c)** Normalized cross correlation

Figure 3.3: Cross-correlation based block matching. Figure (a) shows right view of Figure 3.2b with contrast & brightness change of $\alpha = 1.2$ and $\beta = 100$. Cross-correlation based block matching (c) manages to recover some depth from the modified pair, while SAD based matching (b) fails.

Figure 3.2 compares the ground truth disparity map (c) of two views (a, b) of the New Tsukuba dataset [25, 32] against a naive implementation of Equations 3.1 and 3.2 in Figure 3.2d. Without difficulty, we see that (d) differs greatly from (c). The main reason for this are occlusions and the subsequent normalization to the 8-bit range for display. Because (a) contains regions that are not visible – or: *occluded* – in (b), the SAD minimization chooses some other patch in the match image that cannot be a sensible choice. This causes noise in those occluded areas, highlighted in (d) in orange and blue.

Because of this, pre- and/or postprocessing becomes necessary to obtain a usable result. One can, for example, run the disparity computation with the right view as the reference image, to then compare the disparity outputs against each other:

$$d'(x,y) = \begin{cases} d_l(x,y) & \text{if } |d_l(x,y) - d_r(x,y)| < \theta_c \\ -1 & \text{else.} \end{cases} \tag{3.3}$$

This introduces a left-right consistency check against some fixed threshold θ_c which invalidates occluded regions [9]. Additionally, the disparity range can be limited to a range, which will create a volume of interest. Two ways to implement this are:

- filter $d'(x,y)$ against a range such that any disparities exceeding it are invalidated, or
- limit the disparity search range during the minimization process of Equation 3.2.

The first principle and Equation 3.3 are used to obtain the disparity in Figure 3.2e, where occluded regions are invalidated (marked in blue) such that the result comes closer to the ground truth.

We still notice multiple issues: corners are blurred and fine details like the lamp stand aren't matched properly. By reducing the block size, we can slightly improve on fine details and edges, albeit by introducing noise, as displayed in Figure 3.2f.

Normalized Cross Correlation

Another issue of using the SAD for matching is it's missing invariance on brightness and contrast changes. In other words, the stereo pair must not record images of different lighting conditions each. Formally, those changes can be described by:

$$\tilde{I}(x, y) = \alpha \cdot I(x, y) + \beta, \tag{3.4}$$

which cause SAD based matching to fail, as we see in Figure 3.3. This motivates correlation based metrics. We will use normalized cross correlation in this work for intensity-based matching [9]:

$$\mathrm{NXC}(x, y, d) = \frac{\sum_{x,y \in b} \left[I(x, y) - \bar{I}_b\right] \left[I'(x + d, y) - \bar{I}'_b\right]}{\sqrt{\sum_{x,y \in b} \left[I(x, y) - \bar{I}_b\right]^2 \sum_{x,y \in b} \left[I'(x + d, y) - \bar{I}'_b\right]^2}}, \tag{3.5}$$

with $\bar{I}_b$ as the mean over each block individually. The optimization problem of Equation 3.2 needs to be reformulated:

$$d(x, y) = \arg\max_d \mathrm{NXC}(x, y, d). \tag{3.6}$$

Note that the range of NXC is in $[-1, 1]$, with good matches corresponding to correlations close to 1. This enables us to filter the resulting disparity map and only accept values with a high coefficient. Still, the main downside compared to the SAD is the computational cost associated with computing the NXC for each block, as it requires more complex operations. Ideally, the input data is preprocessed to reduce the search space of NXC-based matching. We will introduce both principles further down.

3.1.2 Semi-Global Matching

The single-shot method used in this work is *Semi-Global Matching*, introduced by Hischmüller [17, 18]. In contrast to block-matching, it characterizes itself by pixel-wise matching with a global smoothness constraint, meaning that more than just the pixels along the epipolar line are used for matching a pixel. This was originally motivated by aerial imagery, where the image-rectification of pushbroom cameras is not always possible [18].

Mutual Information

We already noticed in Figure 3.3 that relying on pixel intensities for matching is unreliable. To make the matching process as independent on individual intensities as possible, Hirschmüller uses *mutual information* for pixel matching as introduced by Kim *et al.* [19]. Minimizing this matching cost is then represented as maximizing the mutual information between candidates. We now introduce the parts that make up the definition of the mutual information matching cost (CMI) [17, 18].

Intensity Probability Distribution Between two images, a probability distribution of corresponding intensities is defined as:

$$P_{I,I'}(i_1, i_2) = \frac{1}{n} \sum_{x,y \in I} \mathcal{T}\left\{[i_1, i_2] = [I(x,y), I'(x,y)]\right\} \tag{3.7}$$

$$\mathcal{T}\{\text{expr}\} = \begin{cases} 1 & \text{if } \textit{true} \\ 0 & \text{else} \end{cases} \tag{3.8}$$

In other words, the probability of two intensities i_1, i_2 given two images I, I' equals the number of times this combination occurs in both images at equal coordinates, over the total number of pixels n in one image.

Joint Entropy Data Term The distribution $P_{I,I'}$ is used to compute a data term:

$$h_{I,I'}(i_1, i_2) = -\frac{1}{n} \log\left(P_{I,I'}(i_1, i_2) \otimes \mathcal{G}(i_1, i_2)\right) \otimes \mathcal{G}(i_1, i_2), \tag{3.9}$$

where $\mathcal{G}(i_1, i_2)$ is a 2D gaussian [47] and $\otimes$ denotes a convolution. This term is related to the joint entropy of the two images: a low joint entropy H indicates that the second image can be predicted from the first. This is the case for a match image that was warped w.r.t. an accurate disparity map.

Mutual Information Matching Cost The mutual information data term of two images is defined with:

$$\text{mi}_{I,I'}(i_1, i_2) = h_I(i_1) + h_{I'}(i_2) - h_{I,I'}(i_1, i_2), \tag{3.10}$$

where $h_I, h_{I'}$ are each entropy terms of the single images:

$$h_I(i) = -\frac{1}{n} \log\left(P_I(i) \otimes \mathcal{G}(i)\right) \otimes \mathcal{G}(i), \tag{3.11}$$

with the single image probability distribution derived from Equation 3.7:

$$P_I(i_1) = \sum_{i_2} P_{I,I'}(i_1, i_2). \tag{3.12}$$

From Equation 3.10, we get the matching cost:

$$CMI(x, y, d) = -\text{mi}_{I,f_D(I')}(I(x,y), I'(x+d, y)). \tag{3.13}$$

Note the subscript: the data terms are computed based on the match image I' that has been warped according to the disparity map D. This requires an initial guess. Hirschmüller suggests a hierarchical solution where starting from a random, downscaled disparity map, the disparity is iteratively recalculated by upscaling the disparity map of the previous iteration.

Global Energy Function

To formalize the problem of finding the optimal disparity map, Hirschmüller defines the energy of the disparity image, similar to [19]:

$$E(D) = \sum_{x,y \in I} \Bigg[\overbrace{CMI(x,y,D(x,y))}^{E_{\text{data}}} \\ + \underbrace{\sum_{x',y' \in \mathcal{N}(x,y)} \Big(P_1 \mathcal{T}\left\{ |D(x,y) - D(x',y')| = 1 \right\} + P_2 \mathcal{T}\left\{ |D(x,y) - D(x',y')| > 1 \right\} \Big)}_{E_{\text{smoothness}}} \Bigg], \tag{3.14}$$

where the second term represents the smoothness constraint in the neighborhood $\mathcal{N}$ around a given pixel:

- disparity changes equal to 1 (oblique surfaces) get penalized according to P_1,
- changes larger than 1 (discontinuities) get penalized by P_2.

A requirement on P_1, P_2 is that $P_2 \geq P_1$. As defaults, libSGM sets $P_1 = 10,\ P_2 = 120$.

Aggregation over Multiple Paths

The optimization of Equation 3.14 over the whole 2D image is inefficient. Thus, Hirschmüller reduces this cost aggregation problem to multiple 1D aggregations along lines that end in the pixel of interest at a disparity d, shown in Figure 3.4. The cost of one line equals to:

$$\begin{aligned} L_{dx,dy}(x,y,d) = \text{CMI}(x,y,d) + \min \Big[& L(x-dx, y-dy, d), \\ & L(x-dx, y-dy, d-1) + P_1, \\ & L(x-dx, y-dy, d+1) + P_1, \\ & \min_{d_i} L(x-dx, y-dy, d_i) + P_2 \Big], \end{aligned} \tag{3.15}$$

with dx, dy as the direction of the line. This implements Equation 3.14 for one line. The recursion stops at the image borders, where $L(x,y,d) = \text{CMI}(x,y,d)$. Summing over the set of directions $\mathcal{D}$ finally gives us the cost used for determining the disparity:

$$\text{SGC}(x,y,d) = \sum_{(dx,dy) \in \mathcal{D}} L_{dx,dy}(x,y,d), \tag{3.16}$$

$$d(x,y) = \arg\min_{d} \text{SGC}(x,y,d). \tag{3.17}$$

This gives a disparity image from the left view. Similar to Equation 3.3, this computation can be performed with the right view as reference to then check its disparity result against the initial left result.

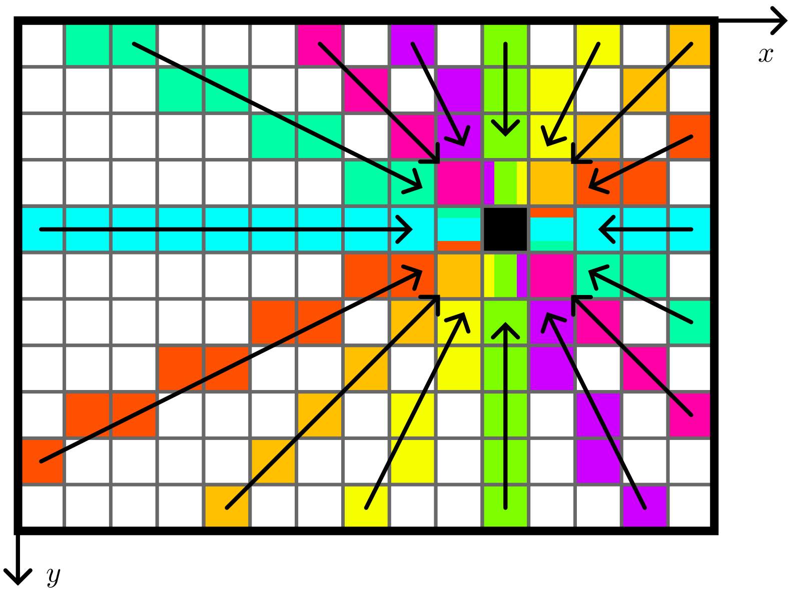


Figure 3.4: Multiple path aggregation along 16 paths. For a given pixel and it's candidate in the match image, the total cost equals the sum of aggregated costs along paths from left to right $(dx, dy) = (1, 0)$, top left left to bottom right right $(dx, dy) = (2, 1)$, top left to bottom right $(dx, dy) = (1, 1)$, and so on. Figure adapted from [17, 18].

Variations & Implementations

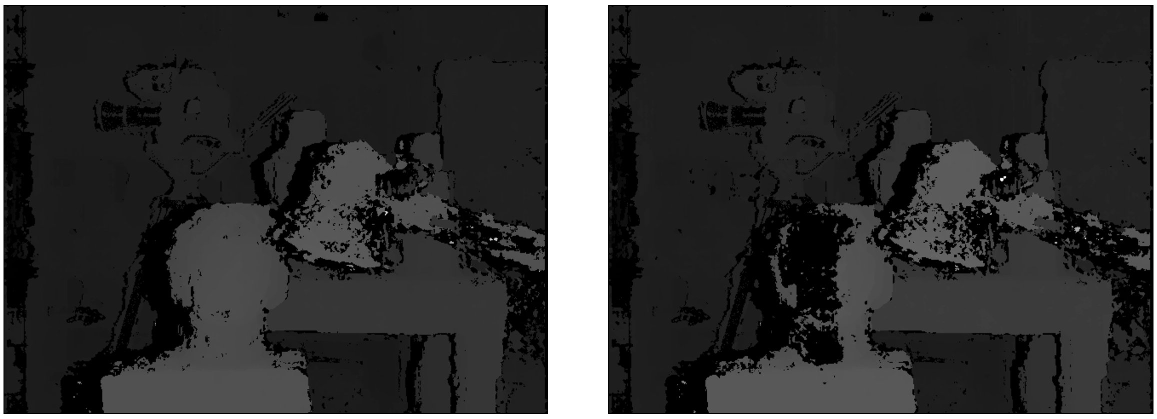
It is possible to replace the CMI cost function in Equation 3.15. Hirschmüller himself discusses the use of a sampling insensitive cost of [2], where the intensities of a 1×3 window around a match pixel along the epipolar line is interpolated to find an optimal match.

In fact, the present implementation in OpenCV [4] uses a *census transform* [43, 52] which creates a binary vector of a pixel's brightness in relation to its neighborhood, to then compute the matching cost as the hamming distance between those vectors. The same applies to libSGM [31] which will be the reference SGM implementation of this work with its result on the Tsukuba images shown in Figure 3.5.

3.2 Multi-Shot

Apart from the aforementioned single-shot approaches that are *passive*, as in: the stereo camera system only passively receives energy reflected off of objects, we can also make it send out energy and turn the scanner into an *active* one. Additionally, instead of relying on only one view per camera, we can match correspondences over a stack of images each, turning the matching process into a *multi-shot* one.

The principle of multi-shot is that now the "block" of block-matching needn't be a spatial one: it might as well be temporal vector, as depicted in Figure 3.6. This approach implies that there must be some change over time. We introduce this change by projecting structured light into it.



(a) Unaltered input images **(b)** Input with right view of Figure 3.3a

Figure 3.5: Semi-Global Matching result of libSGM. Both results are practically identical with the exception to the right side of the face, where the radiometric changes leave no visual structure for SGM to match on.

3.2.1 Binary Features

To match these vectors, one may also use the principle of computing the normalized cross correlation of Equation 3.5 to achieve accurate results. However, performance will suffer because of the computational requirement. As already suggested, preprocessing the images to create an intermediary disparity map reduces the computational load. We implement the method introduced by Dietrich *et al.* [8] – BICOS+ – for computing this intermediary disparity map. It is characterized by creating a binary feature vector encoding the change of each pixel's intensity over time.

This encoding is simply represented via:

$$v_i = \begin{cases} 1 & \text{if } \tilde{I}_v < \tilde{I}_w, \\ 0 & \text{else} \end{cases} \tag{3.18}$$

As the i-th element of the feature vector, comparing two values $\tilde{I}$ computed from the temporal block. Given n intensity values for a pixel, we get:

- $n-1$ comparisons between temporally consecutive intensities,
- n comparisons between each I_i and their mean $\bar{I}$,
- $n-2$ comparisons of temporally consecutive intensities with one-inbetween intensity, and
- $n-3$ comparisons between non-overlapping *pairsums*,

yielding $4n-6$ bits per descriptor. Figure 3.7 shows how the descriptor is computed. Pairsums are the result of comparisons between sums of two intensities, such that the sums do not share an

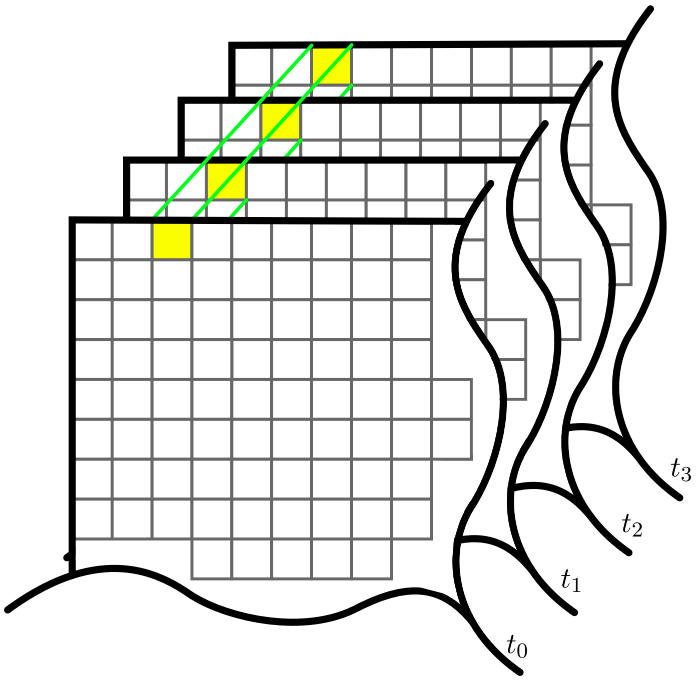


Figure 3.6: Temporal block of multi-shot matching. Given a time-series of images at multiple time-points $t_{0\dots n}$, a temporal vector – or block – of pixel values consists of the intensity of a pixel at multiple consecutive time points.

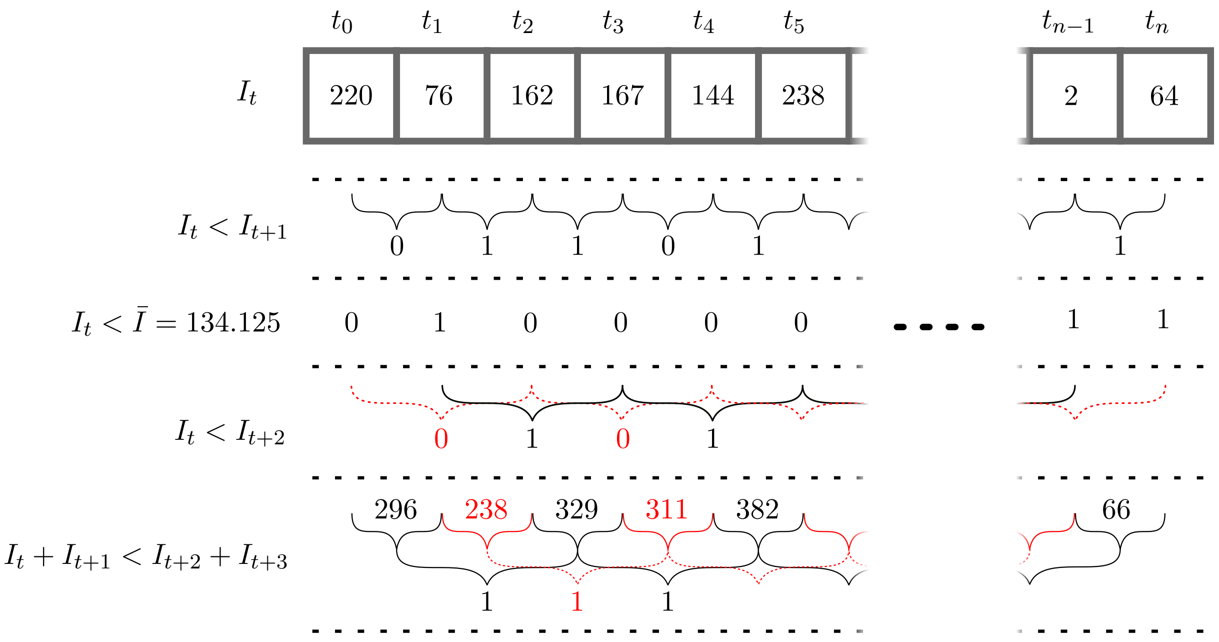


Figure 3.7: BICOS binary descriptor. On the top, a vector of intensities I_t of a given pixel is shown for multiple time points $t_{0\dots n}$. The four rows below depict the comparison scheme between intensities. Adjacent (top), Mean $\bar{I}$ (second), one-inbetween (third), and pairsums (bottom).

intensity.[1] The cost for matching these vectors computes as the hamming distance between them. Because the BICOS feature vectors are binary, the hamming distance between them simply computes with two operations: XOR and popcount. Depending on the processor architecture and the length of this binary vector, this can be reduced to two processor instructions, which is almost negligible compared to a NXC computation.

Disparity Pipeline

The disparity pipeline using BICOS now looks like this:

1. Rectify the input images.
2. Compute descriptors for the input image stack.
3. For each pixel in the left descriptor image, search for a minimum hamming distance on the corresponding epipolar line of the right descriptor image.
4. If there are multiple minima (disparities with equal hamming distance), invalidate the match.
5. For each valid disparity, compute the normalized cross-correlation over the original image stacks, and invalidate matches below a certain threshold.

Optional steps include median-filtering after step 4 or 5. Empirically, this does not significantly improve the results, however. Instead, step 5 improves by subpixel-interpolation.

3.2.2 Subpixel Accuracy

We achieve subpixel accuracy by interpolating between the 1D neighborhood in the image stack via fitting a 2nd order polynomial, such that the interpolated intensity vector $\tilde{\mathbf{I}}$ computes as:

$$\tilde{\mathbf{I}}(z) = \mathbf{a}z^2 + \mathbf{b}z + \mathbf{c}. \tag{3.19}$$

Given a disparity estimate $\tilde{d}$ of step 4 of the pipeline, we interpolate between $[\tilde{d} - 1, \tilde{d} + 1]$ in the image stack $\mathbf{I}$ containing intensity vectors. We can directly solve for $\mathbf{a}, \mathbf{b}, \mathbf{c}$ given the neighborhood around $\mathbf{I}(x + \tilde{d})$:

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -1 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{I}(x + \tilde{d} - 1) \\ \mathbf{I}(x + \tilde{d}) \\ \mathbf{I}(x + \tilde{d} + 1) \end{bmatrix} \tag{3.20}$$

such that interpolation around $\mathbf{I}(x + \tilde{d})$ is possible by choosing $z \in [-1, 1]$ and computing Equation 3.19. In practice, we set a step-size to compute points on that polynomial to then only consider the interpolated disparity with maximum correlation coefficient. A reasonable choice for this discrete step is between 0.1 and 0.25.

[1]This differs from [8] where the authors choose to compare a pairsum against every other pairsum of the vector

3.2.3 CUDA-based Binary Correspondence Search

A problem with high-resolution images in stereo vision is the runtime of stereo-vision algorithms, since initially, every pixel on the epipolar line must be accessed for matching. Methods to decrease this search-space exist, the most simple being to directly limit the disparity search space which results in only certain pixels in the epipolar line being considered for matching. However, this implicitly reduces the measurable 3D volume of the stereo vision system. This is why it is necessary to consider the runtime-efficiency of stereo algorithms for them to be considered for real-time applications.

Again, BICOS reduces the operations for comparing descriptors to a minimum; The issue is the sheer amount of pixels needing to be processed. Noting that every left-view pixel's comparisons are independent of each other, we parallelize this exhaustive search by utilizing the compute capabilities of GPUs. We now take a look at how this search can be implemented using CUDA v12, by first giving a brief overview of CUDA terminology and pitfalls, based on information from the CUDA C++ programming guide [29].

Grids, Blocks & Kernels

CUDA programs are organized in *kernels*, which are run on the device. To define a kernel, the `__global__` attribute is used in code otherwise similar to C++:

```
// helloworld.cu
#include <stdio.h>

__global__ void my_kernel() {
    printf("Hello world\n");
}
```

This creates a global symbol which is called – or, in the context of kernels: *launched* – from *host* code:

```
int main(void) {
    my_kernel<<<1,1>>>();
}
```

The kernel launch arguments, passed via triple brackets `<<<>>>` indicate the *grid*- and *blocksize* of the kernel. With both set to 1, a single thread inside a single block will run the kernel once.

These arguments are important to partition input images between the available GPU threads and multiprocessors. For example, say we have an image with dimensions `rows,cols` at contiguous memory `data`, which is supposed to undergo a photometric transform as of Equation 3.4:

```
 1  // image.cu
 2  __global__ void photometric(
 3      unsigned char *data, int rows, int cols,
 4      float alpha, float beta
 5  ) {
 6      const int col = blockIdx.x * blockDim.x + threadIdx.x,
 7                row = blockIdx.y * blockDim.y + threadIdx.y;
 8
 9      if (cols <= col || rows <= row) return;
10
11      unsigned char &pixel = data[row * cols + col];
12
13      pixel = std::saturate_cast<unsigned char>(alpha * pixel + beta);
14  }
```

The individual pixel coordinates `col,row` (line 6f) are computed from global variables `blockIdx`, `blockDim` and `threadIdx`. These are set on a per-kernel-invocation basis, with respect to the block and grid parameters that were set when launched. Then, the kernel guards against out-of-bounds access in line 9, to then access a single pixel (line 11) which is then modified (line 13) by clamping its floating-point computation. To run this kernel across the GPU's multiprocessors with maximum thread count, we write:

```
15  int main(void) {
16      const int blocksize = 1024;
17      unsigned char *data_host, *data_device;
18      int rows, cols;
19
20      read_image(&data_host, &rows, &cols);
21
22      cudaHostRegister(data_host, rows * cols, 0);
23      cudaHostGetDevicePointer(&data_device, data_host, 0);
24
25      dim3 grid((cols + blocksize - 1) / blocksize, rows);
26      photometric<<<grid, blocksize>>>(data_device, rows, cols, 1.5f, -10.0f);
27
28      cudaHostUnregister(data_host);
29
30      // ...
```

if there are no overlapping intensities. In this work, only adjacent pairsums are compared.

```
}
```

Analyzing this code, we first set a blocksize and forward declare variables (lines 16-18). Then, some function reads an image into host memory (line 20), pointed to by `data_host` with its dimensions in `rows` and `cols`. Following, lines 22f handle one pitfall of CUDA programming: CPU and GPU don't necessarily share memory. There are multiple ways of solving this: one involves uploading the `rows*cols` sized memory onto the device with `cudaMalloc` and `cudaMemcpy`. Our approach pins the host memory pointed to by `data_host` via `cudaHostRegister` to then be made accessible by the device in `data_device` by `cudaHostGetDevicePointer`.

Before actually launching the kernel, we partition the input image by computing the `grid` according to the input image dimensions and `blocksize`.[2] We then invoke the kernel with its parameters, launching batches of 1024 threads on the GPU. This hints at the benefits of GPU programming: while consumer-grade CPUs (at the time of this thesis) do not exceed 32 threads, GPUs can utilize multiples of that, albeit for simpler computations. In a figurative sense, this will make a 1×1024 sized kernel move over the data, similar to convolutions.

It is worth noting that both block and grid dimensions can be up to three-dimensional. With a blocksize of 32×32, accesses of one block would require multiple rows. Our photometric kernel will also work with two-dimensional block sizes. (Albeit that the second grid dimension must be computed differently.) We decide against this, since access to a pixel's 2D neighborhood is not a requirement and by convention images are laid out in memory as the concatenation of their rows, which is why cache-locality is improved by using a 1D blocksize. This also holds for our BICOS implementation.

Finally, calls to `cudaHostRegister` require a corresponding call to `cudaHostUnregister` to unpin the memory again.

Exhaustive Pixelwise Hamming Distance Minimization

Now that we have an understanding about basic CUDA terminology, we can look at the implementation of matching the binary descriptors of BICOS, presented in the previous section. There, we also discussed how the number of input images n as the length of the intensity vector $\mathbf{I}$ determines the length of the binary feature vector. It is clear that k bits may be represented in code as an unsigned integer, given that k is smaller or equal to the maximum size of unsigned integers on that platform. With `nvcc` – the CUDA compiler – there is support for up to 128-bit unsigned integers. However, it is not necessary to forcibly use 128-bit descriptors for all k; It makes sense to determine this dynamically based on n. Thus, we need to differentiate between bit-widths in code, which calls for function templating or -overloading. We start with the intrinsic comparison between feature vectors:

```
// bicos.cu
#include <stdint.h>
```

[2]This is practically achieved by calling `cv::cuda::device::divUp(cols, blocksize)` of OpenCV [4] which implements this behaviour, and from where the grid computation listed was borrowed from.

```

__device__ int ham(uint32_t a, uint32_t b) {
    return __builtin_popcount(a ^ b);
}
```

This implements what we have already described: hamming distance computation with two operations, a binary XOR (^) and a call to `__builtin_popcount` which is a compiler-builtin that is translated into a machine instruction, verified by the generated PTX assembly[3] as the instructions that are run on the device equal to:

```
.visible .func  (.param .b32 distance) ham(unsigned int, unsigned int)(
    .param .b32 a,
    .param .b32 b
) {

    ld.param.u32    %r1, [a];            // load a into register 1
    ld.param.u32    %r2, [b];            // load b into register 2
    xor.b32         %r3, %r2, %r1;       // compute difference into register 3
    popc.b32        %r4, %r3;            // compute popcount into register 4
    st.param.b32    [distance+0], %r4;   // store register 4 in distance
    ret;                                 // return
}
```

The types `uint32_t` of `stdint.h` guarantee a 32-bit wide integer. Note that `__device__` specifies code which can only be called from the GPU. The implementation for 64 bit is analogous;

```
__device__ int ham(uint64_t a, uint64_t b) {
    return __builtin_popcountll(a ^ b);
}
```

The only difference being the intrinsic as `__builtin_popcount`longlong. Its generated PTX is equal with the only difference being the 64-bit width. 128-bit integers require some intervention however, since there is no builtin popcount for that width:

```
using uint128_t = __uint128_t;

__device__ int ham(uint128_t a, uint128_t b) {
```

[3]Generated via `https://godbolt.org` using `nvcc 12.5.1`.

```
    const uint128_t diff = a ^ b;
    const int lo = __builtin_popcountll((uint64_t)diff),
              hi = __builtin_popcountll((uint64_t)(diff >> 64));
    return lo + hi;
}
```

As anticipated, the computational requirements increase with larger n. We also see that `uint128_t` as an alias for `__uint128_t` is in fact only an emulation by the compiler, indicated by the double underscore prefix. Given those intrinsics, we can have a look at the actual kernel for the initial disparity computation:

```
template<typename TDescriptor>
__global__ void bicos_disparity_kernel(
    const TDescriptor *ldesc,
    const TDescriptor *rdesc,
    cv::cuda::PtrStepSz<int16_t> disparity
) {
    const int x = blockIdx.x * blockDim.x + threadIdx.x,
              y = blockIdx.y * blockDim.y + threadIdx.y,
              step = disparity.cols;

    if (disparity.cols <= x || disparity.rows <= y) return;

    const TDescriptor l = ldesc[y * step + x];
    const TDescriptor *rrow = rdesc + y * step;

    int best_rcol = -1,
        min_cost = INT_MAX,
        num_duplicate_minima = 0;

    for (int rcol = 0; rcol < retval.cols; ++rcol) {
        const TDescriptor r = rrow[rcol];

        int cost = ham(r, l);

        if (cost < min_cost) {
            min_cost = cost;
            best_rcol = rcol;
            num_duplicate_minima = 0;
        } else if (cost == min_cost) {
            num_duplicate_minima++;
        }
    }

    retval(y, x) = 0 < num_duplicate_minima ? -1 : std::abs(x - best_rcol);
}
```

`TDescriptor` as a template parameter needs to be set by the calling code to indicate the feature-bitwidth. Starting with line 37, we search along a single right row (index `rcol`) of the right image to find a minimum that has no duplicates, else we invalidate the match by setting -1. One of the input parameters of this kernel is an OpenCV `PtrStepSz`. It wraps an image's data pointer, dimensions and row step and can be implicitly constructed from a `cv::cuda::GpuMat` which is the OpenCV representation for images located on the GPU. For this kernel, it needs to be preallocated.

This implementation can likely be improved; For example, the left and right descriptor rows can be fetched into shared memory. As the name implies, this memory is shared between threads inside a block. Thus, memory access speed could be improved by not relying on global memory.

Chapter 4

Scanner Hardware

This chapter gives a brief overview of the hardware displayed in Figure 4.1 which was used for the experiments of this work.

4.1 Cameras

We use two Teledyne FLIR Oryx 10GigE cameras for our scanner: the model variant ORX-10G-71S7C-C. It uses a 3208×2200 Sony IMX420 sensor with a pixel size of 4.5 µm and a bayer color filter. Up to 112 frames per second with a ADC-bit depth of 12 bit can be captured. We utilize 12 bits for our stereo processing. However, this puts additional requirements onto the network hardware, since 12 bits are transmitted as 16 bits per pixel in the image, effectively doubling the data rate compared to standard 8-bit imagery.

For lenses, we use the Computar V0828-MPY2. The manufacturer advertises this model as a "ultra-low distortion lens". We will achieve that objective in combination with the calibration process.

4.2 Trigger

To synchronize both cameras, we use a Raspberry Pi Pico. This microcontroller board has the capability of programmable IO pins, utilizing the domain-specific assembly language *pioasm*, consisting of a small set of instructions that each only take one processor cycle. Therefore, customized, cycle-accurate square-waves are generated, across multiple outputs. Our usecase for this functionality in the projects' scope is the triggering of the projector and the cameras, such that both cameras are triggered once, without projector illumination, and afterwards once with illumination.

4.3 Projector

For initial experiments, we use an aftermarket projector that was freely available at the robotics chair; An EPSON EMP-730. To generate patterns, the software `VRRTest` was used, available on GitHub. It's original purpose is the detection of tearing on PC monitors. For this, it generates

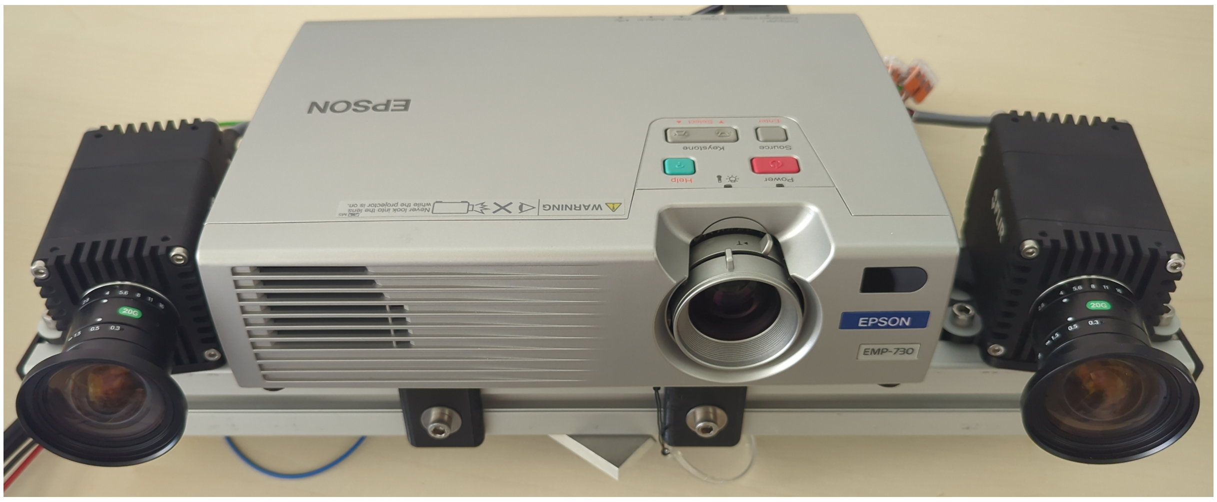


Figure 4.1: The stereo scanning setup.

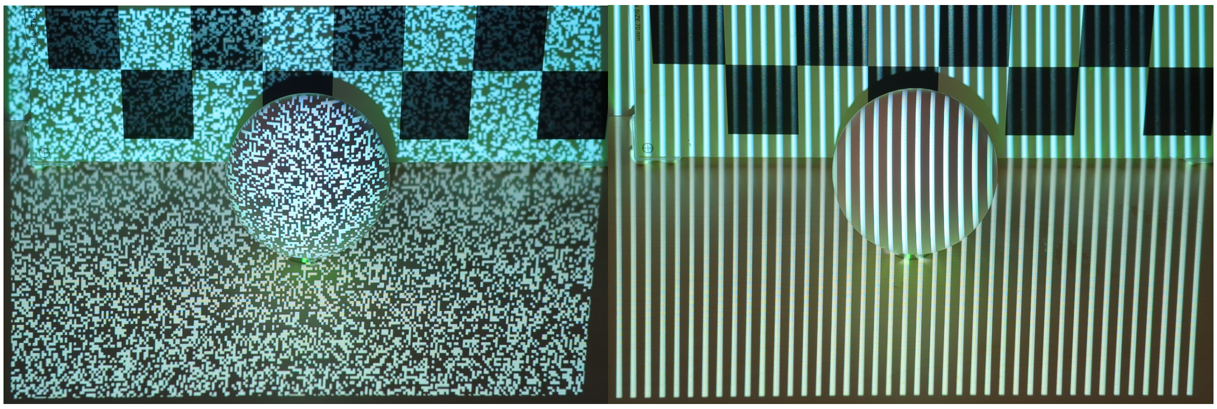


Figure 4.2: Speckle (left) and fringe (right) patterns for structured light, varying over time.

bar patterns or moving squares. It was modified for the purpose of generating structured light patterns. Figure 4.2 shows the two kinds of structured light projected. The projector was mounted together with the cameras on an aluminum profile, using a 3D-printed adapter. Its projections were not synchronized with the cameras during our experiments.

Chapter 5

Evaluation

This chapter evaluates the performance of our stereo scanner. First, we discuss the calibration result. Then we look at the industry standard's definition of evaluating a 3D scanners definition, namely VDI/VDE Guidelines 2634, to be able to quantize the performance. We then finally evaluate the performance above water with an aftermarket projector, to then give an outlook of the effects of a water surface between the scanner and the scanned object.

5.1 Calibration

We calibrate the stereo-pair using a 29×29 ChArUco board with a checker size of 25mm. In total, 146 images of the calibration pattern in different poses are used for the calibration. Figure 5.1 shows five calibration pairs as an example. Images are taken by both cameras simultaneously using a hardware trigger signal. Pattern points are extracted via OpenCV. We first calibrate each camera separately by `cv::calibrateCamera` and then use the intrinsic parameters as an initial guess for `cv::stereoCalibrate`. Table 5.1 shows the residual root-mean-square reprojection error of the calibration with maximum per-view-errors, both in pixels. Over all calibration images, we achieve an RMS of one third of a pixel.

The calibration process also returns the per-view extrinsics of the camera w.r.t. the calibration pattern, such that the reprojection of the ChArUco pattern can be computed. This already happens internally in `cv::calibrateCamera` for the RMS computation. In Figure 5.2, we show two binned quiver plots with arrows indicating the residual errors in relation to the image region. To compute these residuals, we use `cv::projectPoints` to reproject points that were detected in each image, for each image, using the initial intrinsic result of `cv::calibrateCamera`, since `cv::stereoCalibrate` optimizes them. Then we simply take the difference between detected and reprojected points. The images show that the only significant residuals exist in the image corners, which is acceptable, since after rectification we won't rely on those regions.

5.2 VDI/VDE 2634

To define a measure for 3D scanning performance of the scanner, we touch on the acceptance tests of the VDI/VDE guidelines 2634 [46]. These include measurements of a reference sphere,

Camera	RMS Error	per-view error max
left	0.34909	0.96066
right	0.36122	0.91533
pair	0.32080	-

Table 5.1: Calibration errors in pixels.

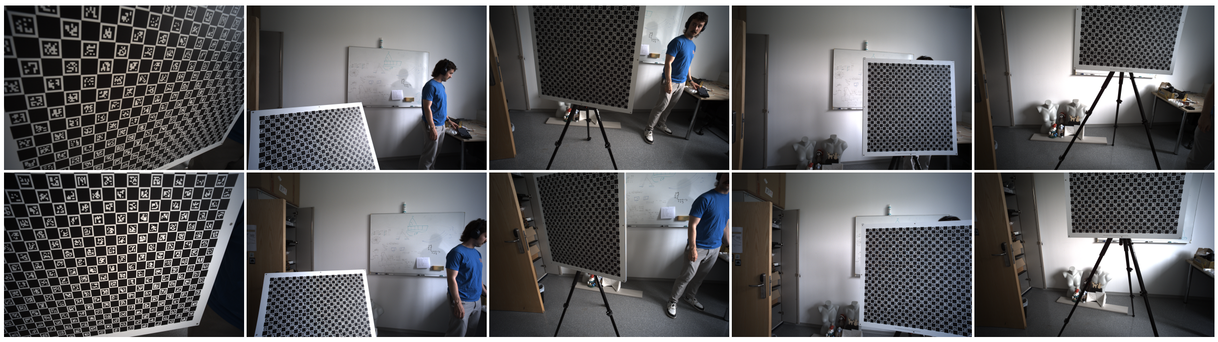


Figure 5.1: Example calibration image pairs. Top: left camera; Bottom: right camera.

plane and dumbbell.

5.2.1 Sphere Probing Error

The sphere probing error consist of two parts, the *form*- and *size* error. Both of them are depicted in Figure 5.3a. The guidelines require the best sphere to be determined via a least-squares fit. We can compute this fit as follows; The sphere equation:

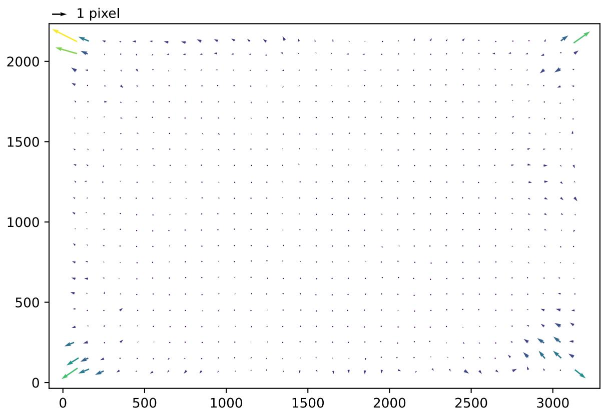
$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \tag{5.1}$$

can be rewritten as:

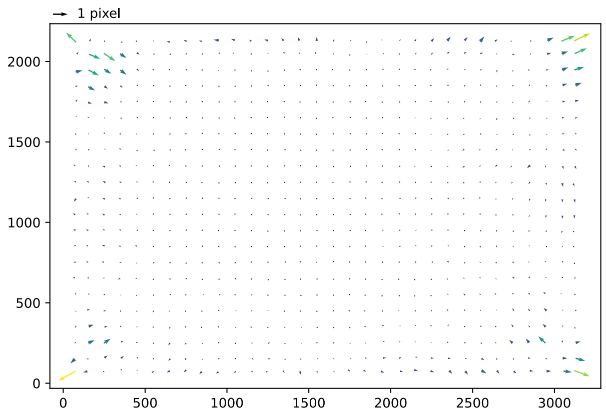
$$x^2 + y^2 + z^2 = 2xx_0 + 2yy_0 + 2zz_0 + r^2 - x_0^2 - y_0^2 - z_0^2. \tag{5.2}$$

This can in turn be represented in matrix form for n points $(x, y, z) \equiv \mathbf{v}$ of the measured pointcloud:

$$\underbrace{\begin{bmatrix} x_1^2 + y_1^2 + z_1^2 \\ x_2^2 + y_2^2 + z_2^2 \\ \vdots \\ x_n^2 + y_n^2 + z_n^2 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 2x_1 & 2y_1 & 2z_1 & 1 \\ 2x_2 & 2y_2 & 2z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 2x_n & 2y_n & 2z_n & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ r^2 - x_0^2 - y_0^2 - z_0^2 \end{bmatrix}}_{\mathbf{x}=[\mathbf{v}_0|w]^T}, \tag{5.3}$$



(a) Left camera



(b) Right camera

Figure 5.2: Residual errors of left and right views after calibration, over the whole, unrectified image area. The arrow direction indicates the residual direction of reprojected points to detected points. Each bin has the dimensions of 100 × 100 pixels.

for which the best-fit $\mathbf{x}$ can be optimized for with functions like `numpy`'s `linalg.lstsq(A,b)`. The probing errors then compute as:

$$e_f = \max_{\mathbf{v}} \underbrace{|\mathbf{v} - \mathbf{v}_0|}_{d} - \min_{\mathbf{v}} |\mathbf{v} - \mathbf{v}_0|, \tag{5.4}$$

$$e_s = 2\,(r - r_0) \tag{5.5}$$

with r_0 as the reference sphere's radius and d as the distance between a point and the sphere center.

5.2.2 Plane Flatness Error

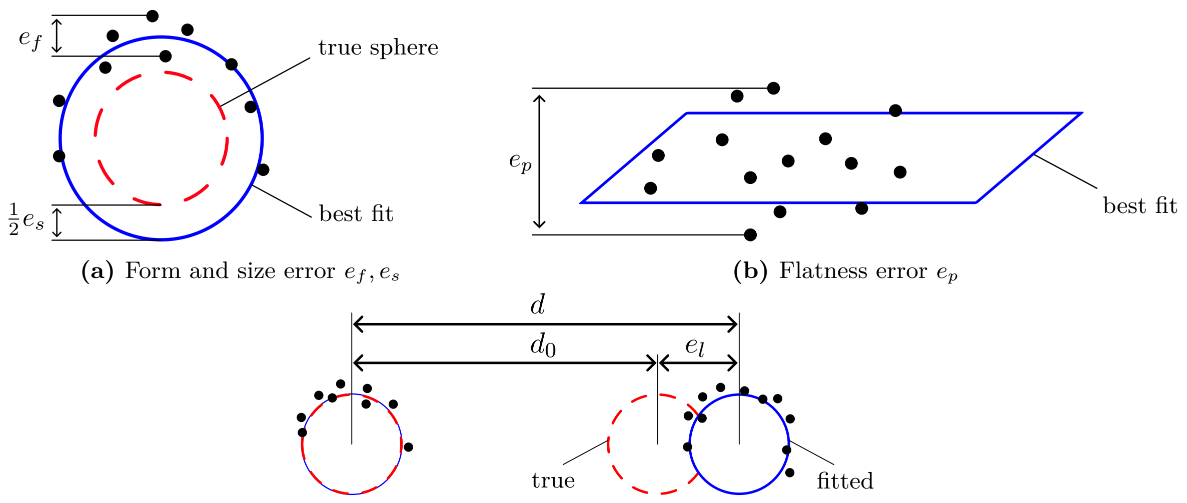
The flatness error is, as shown in Figure 5.3b, computed from the thickness of a best-fit plane. In the same fashion as for spheres, the guidelines require the best-fit plane for a set of points to be computed using least-squares.
We compute this fit via an eigenvalue-decomposition of the covariance matrix $\Sigma = \sigma_{\{\mathbf{v}\}}\sigma_{\{\mathbf{v}\}}^T$, where the base point of the plane is the mean $\bar{\mathbf{v}}$ of the set of points, and its normal vector $\mathbf{n}$ is the decomposition's eigenvector corresponding to the smallest eigenvalue.
Given these parameters, the flatness error computes as:

$$e_p = \max_{\mathbf{v}} \underbrace{[(\mathbf{v} - \bar{\mathbf{v}}) \cdot \mathbf{n}]}_{d} - \min_{\mathbf{v}} [(\mathbf{v} - \bar{\mathbf{v}}) \cdot \mathbf{n}] \,. \tag{5.6}$$

This follows from the fact that the perpendicular distance d of a point $\mathbf{v}$ to a plane is the vector from $\bar{\mathbf{v}}$ to $\mathbf{v}$ projected onto the normal $\mathbf{n}$.

5.2.3 Sphere Spacing Error

Lastly, the sphere spacing error indicates how distances measured by the scanner deviate from reality. To compute this error, a reference dumbbell of two spheres at a fixed, known distance



(a) Form and size error e_f, e_s **(b)** Flatness error e_p

(c) Spacing error e_l

Figure 5.3: VDI/VDE 2634 metrics used in evaluation. The form error is defined as the "thickness" of the pointcloud sphere, similarly, the flatness error is the thickness around a best fit plane. The sphere size error computes as the diameter difference. The sphere spacing error is the difference between the distances of two fitted spheres against their true (calibrated) distance. Figures adapted from [3].

from each other are scanned. Into each sphere's pointcloud, a sphere with fixed radius is fitted using least squares, to then compute the error as the difference between the distance of the fitted spheres d and the true distance d_0. Figure 5.3c visualizes this definition.

The fit given a fixed sphere radius can be computed similar to Equation 5.3, simply by subtracting r^2 from $\mathbf{x}_4$ and $\mathbf{b}_i$. The error for spheres a, b then computes as:

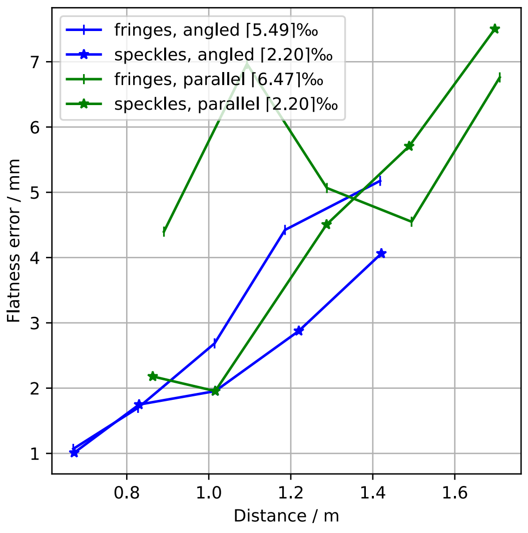
$$e_l = \underbrace{|\mathbf{v}_0^a - \mathbf{v}_0^b|}_{d} - d_0. \tag{5.7}$$

5.3 Measurements

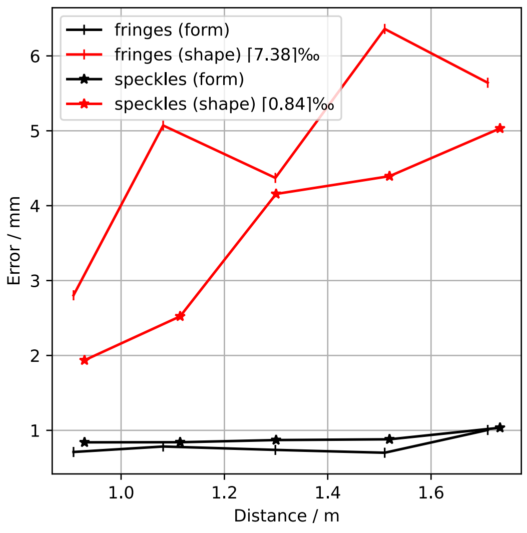
For the in-air evaluation, we scan both a reference sphere (d = 14.5 cm), a plane and a resin-printed dumbbell (shown in Figure 5.5a), a garden gnome and lastly a larger, synthetic scene. The reference shapes are evaluated in accordance to the metrics of the previous section, while the other scans are compared to reference scans. Additionally, it is worth to note that the guidelines indicate that it is allowed to remove coarse outliers from the scans, limited to 3‰ points of the original pointcloud. We use a 6σ threshold criterion for rejecting outliers, which fulfills this requirement for most scans, as we now show.

5.3.1 VDI/VDE Metrics

Figures 5.4 and 5.5b show the errors of each metric plotted over the distance from the 3D scanner. This distance is computed from the respective fits; In the plane-case it is the plane's base point, for spheres it is their center. For dumbbells, it is the middle point on the line connecting both

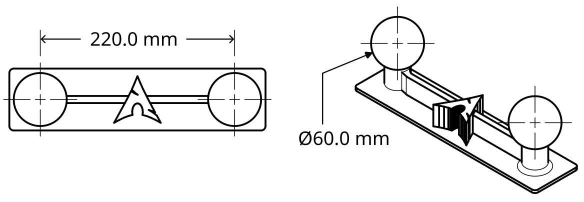


(a) Plane flatness

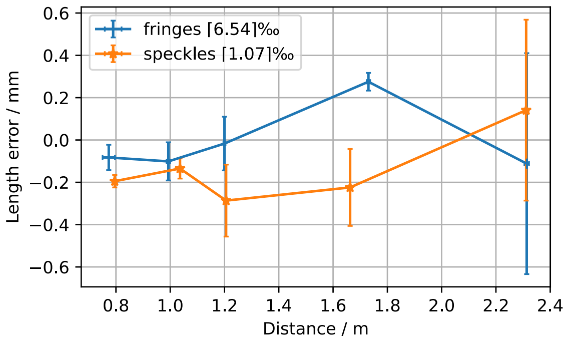


(b) Sphere probing

Figure 5.4: Plane flatness and sphere probing errors, at different distances inside the measurement volume. On the left, plane flatness is compared between fringe and speckle patterns, for two plane configurations; One where the scanner (its baseline) is parallel to the plane (green); Another where the plane is angled at approximately 45 degrees (blue). On the right, sphere shape error (red) and sphere form error (black) is shown. Points are filtered such that outliers with a deviation larger than $6 \cdot \sigma$ are discarded, resulting in given permilles of points being removed.



(a) Dumbbell model and dimensions



(b) Sphere spacing error

Figure 5.5: Reference dumbbell and sphere spacing errors. Each point on the right graph includes errorbars. These errorbars are equal to the standard error of the mean, and necessary because we take measurements of the dumbbell in three different orientations for each distance. The legend indicates the maximum ratio of points that were omitted in the second fitting step.

spheres. The norm of these vectors gives us the measurement distance. Since the dumbbell is oriented in three different ways at each distance, we compute the mean over the distances to each line's middle-point. Coarse outliers are filtered by discarding points that exceed $6 \cdot \sigma$, and in case of the dumbbell, the least-square fit is repeated. Inspecting the plotted plane-flatness and sphere-shape error, we notice that fringes are error-prone in the fronto-parallel configuration. This is because of the ambiguities caused by a periodic fringe pattern. In practice, we try to mitigate this by repeatedly phase-shifting that pattern, while propagating it to the right over the projected area. Still, this method lacks robustness, especially compared to speckle patterns that require much less configuration. In this case, this configuration only consists of:

- framerate,
- white/black ratio, and
- speckle size.

In contrast, the configurable parameters of our fringe pattern are:

- fringe width + change of fringe width,
- maximum & minimum number of fringes (cycling),
- propagation speed, and
- duration between phase shift.

Additionally, there is the projector's adjustable focus. Experiments show that a sharp speckle-pattern helps accuracy, while fringes work best with out-of-focus projection. However, a fine-tuned fringe configuration may result in higher accuracy. Concerning form and spacing errors, we observe that sub-millimeter accuracy is achieved. This speaks for the quality of the calibration; Distances in reality are well-reproduced in the 3D scans. Respective errors only significantly increase for higher distances, as the set of calibration images does not include calibration patterns at that range. Additionally, the cameras' optics are not focused on that distance.

5.3.2 Free-Form Scans

Free-form scans help to get an idea about how our system will perform when used to scan actual 3D scenes. This is why this subsection will analyze scans of two different scenes at different distances.

Garden Gnome

Figures 5.7 and 5.8 show free-form scans of a garden gnome in Figure 5.6 at a distance of 90 cm. The ground truth scan is provided by the chair of robotics and was recorded using a laserscanner mounted on a robotic arm to allow for a complete capture. The recorded pointclouds were aligned with ICP using the software CloudCompare.

A more quantitative analysis is provided in Figure 5.9. There, the raw point-to-point errors as computed by CloudCompare are displayed as histograms with 512 bins. We see that at close

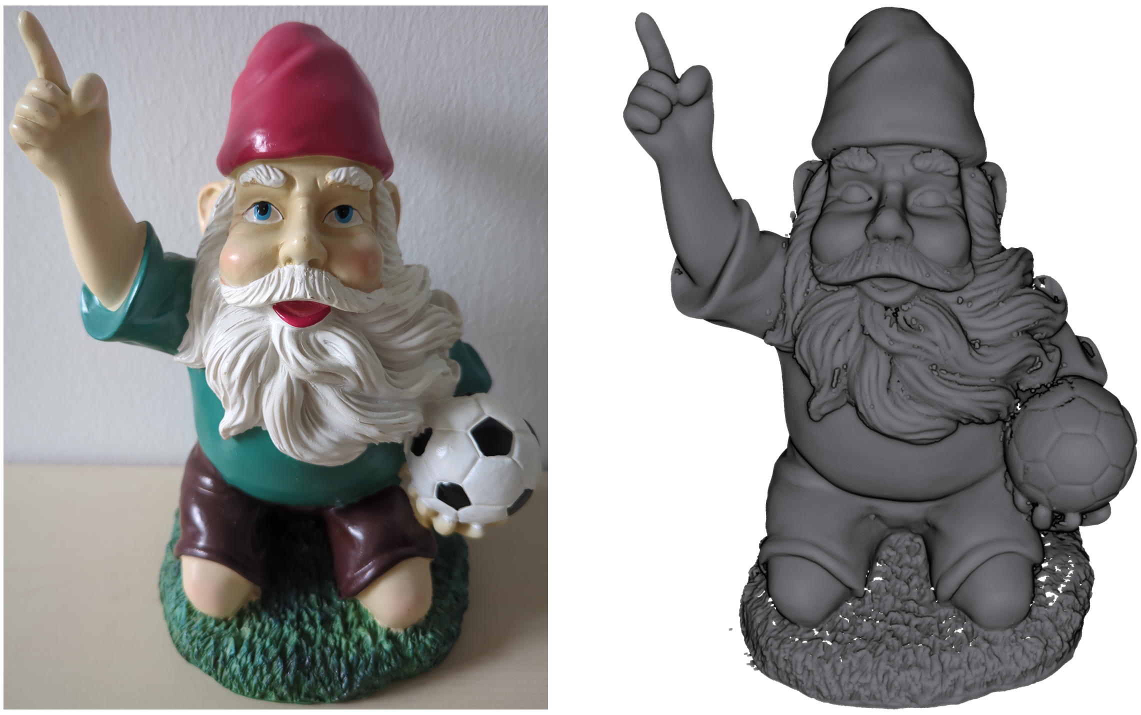


Figure 5.6: The garden gnome in natural light and his ground truth reference scan.

range, the errors are in a similar range of up to 25 mm. Noting the logarithmic y-axis, we see that the majority of errors lie below 2.5 mm. However, using fringes, a denser pointcloud is achieved.

The scans show that especially the top of the gnome's bonnet and the bottom parts of his belly cause errors. One of the reasons for this is the reduced reflectivity caused by the darker color in combination with the angle of the surface. As a contrast example, the beard is well-captured, in part because of the high reflectivity of white.

Adam & Eve

Figure 5.10 shows the artificial scene "Adam & Eve". Scans with the in-air setup are depicted in Figures 5.11, 5.12 and 5.13. They were recorded at a distance of 150 cm. The ground-truth scan has been recorded using an IDS-Imaging Ensenso X36-5FA-8 structured-light scanner. Similar to the evaluation of the gnome's scans, Figure 5.14 shows the errors of our different scanning configurations. We now include scans with libSGM in the comparison.

Starting with SGM with no additional projected pattern – "raw single-shot" – we see that the scan is erroneous, with many errors higher than 10 cm. This is mainly caused by its implementation; Because of the memory-requirements of SGM, the disparity search space is limited to 256, severely reducing the scan-volume of the scanner. This causes any pixels that would have their match outside of this range to be matched with a wrong pixel. In practice, this implementation of SGM is only applicable in two cases: one where the measurement volume can be limited to a given range, or another where binning of the input images is acceptable. This is

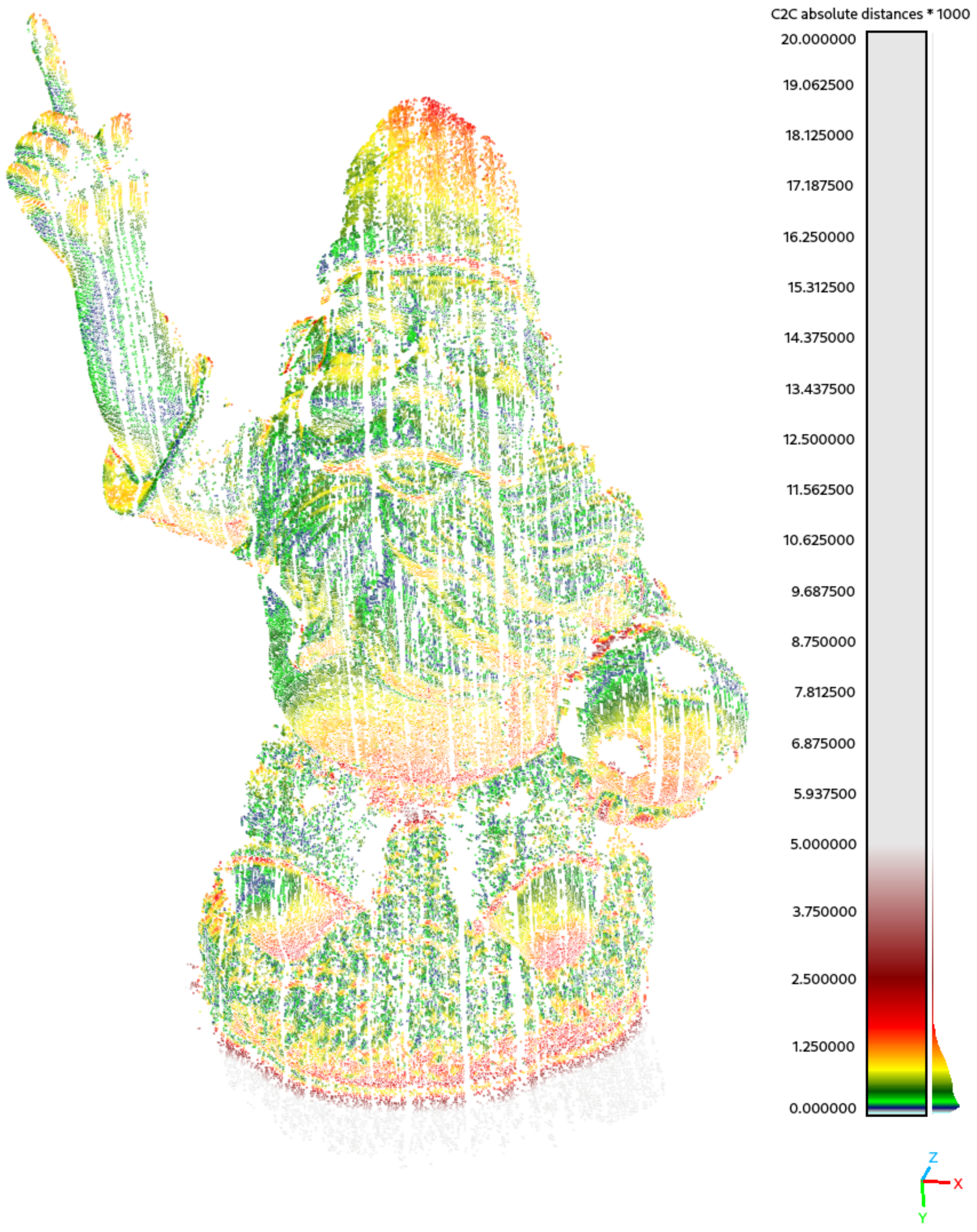


Figure 5.7: Scan of the garden gnome using a fringe pattern and a sequence of 17 images. The scalebar on the right relates each point's color to a numerical error in millimeters.

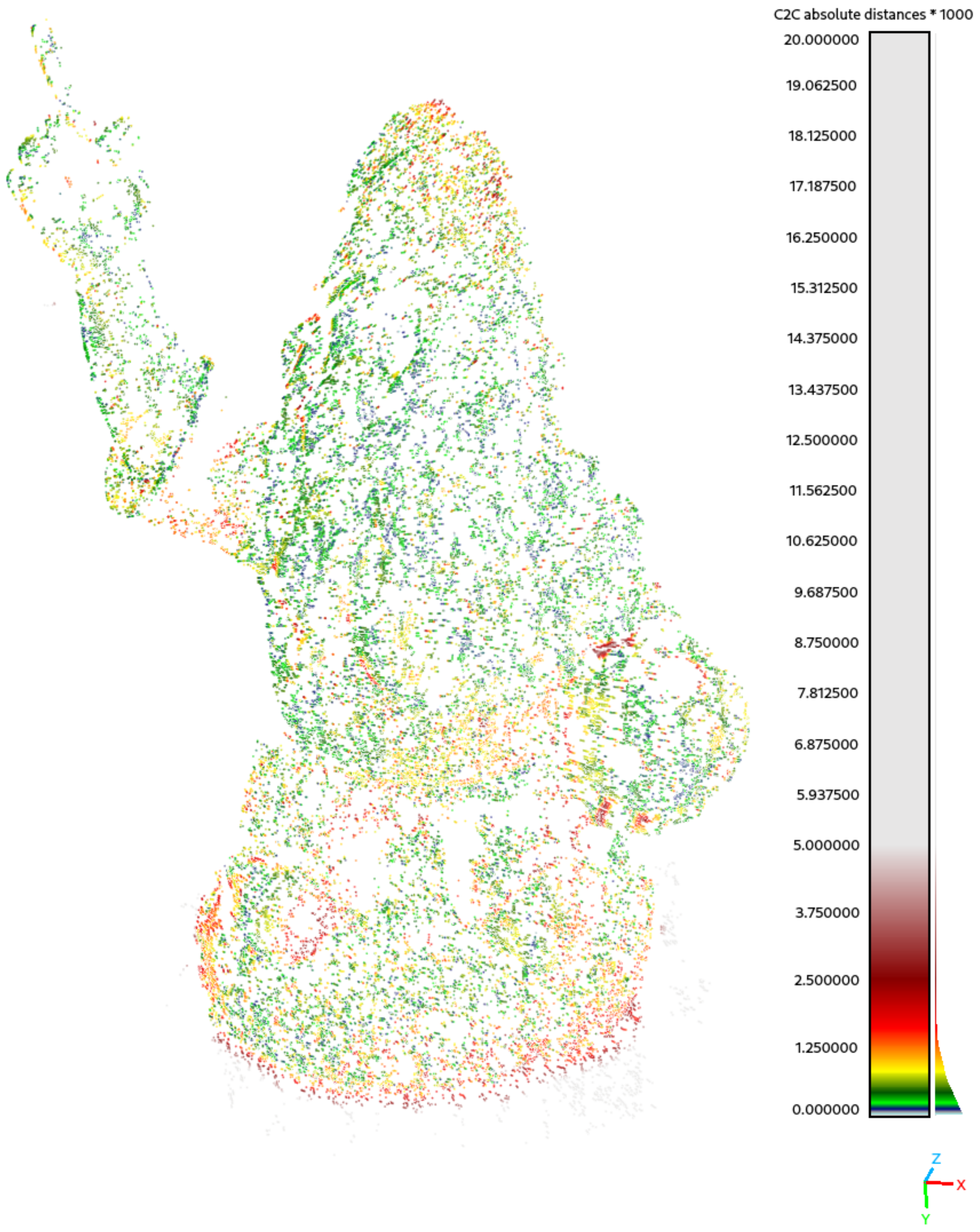


Figure 5.8: Scan of the garden gnome using a speckle pattern and a sequence of 17 images. The scalebar on the right relates each point's color to a numerical error in millimeters. Equal coloring parameters as in Figure 5.7 are used.

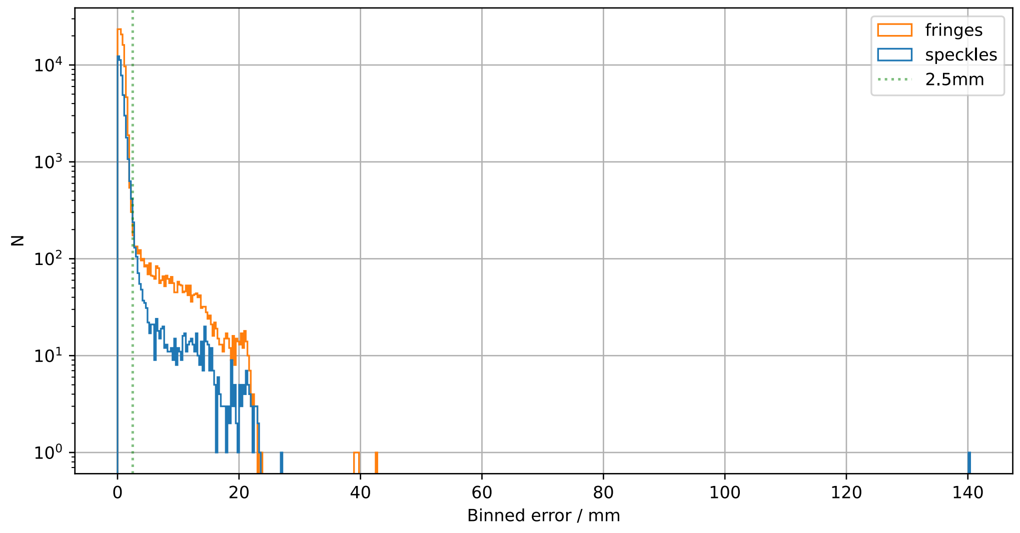


Figure 5.9: Histogram plot of errors between ground truth and each scan of the garden gnome. The errors are distributed into 512 bins with their frequency on a logarithmic y-axis. Each section of the histogram is centered around the center of its bin.

because reducing the resolution of the input images helps with memory consumption, at the cost of scan resolution. Also, binning requires adjusting the camera parameters, so this work does not investigate binned input images. Increasing the disparity range to 512 would exceed the memory available on the Nvidia RTX 4090 used for this work. An improvement with SGM is achieved when projecting additional structure into the scene; For this work, a random point pattern was projected by fixing a diffractive optical element (DOE) in front of a laser-pointer. This especially improves the matching of flat, otherwise structure-less surfaces.

Our BICOS implementation however does not require a limited disparity search range, since it is relatively memory-efficient. This yields a large measurement range, only limited by the cameras' resolution. One limitation of our evaluation of BICOS is the chosen projector: the EPSON EMP-730's field of view does not match those of our cameras, effectively (but *not* computationally) reducing the scanning ROI. The only task when using BICOS is its setup configuration, including the projected patterns. For our evaluation, we compare two fringe- and speckle-configurations. Starting with a sequence size of 17 images and a fringe pattern, we already effectively halve the error range of SGM to up to 100 mm. With the majority of errors below 7.5 mm. This is improved with a window size of 22, at the cost of a higher scan latency. Compared to fringe patterns, a sharply focused, randomly generated speckle pattern, where each pixel is set to white with a probability of 50% and the speckle size is a low multiple of the projector's pixel size, already provides accurate pointclouds at a lower sequence size of 13 images. Similarly, increasing the sequence length reduces errors of speckle matching. Generally speaking however, speckle patterns create more coarse outliers, while fringe patterns cause errors at discontinuities and ambiguities when poorly configured, manifesting themselves as "duplicated"

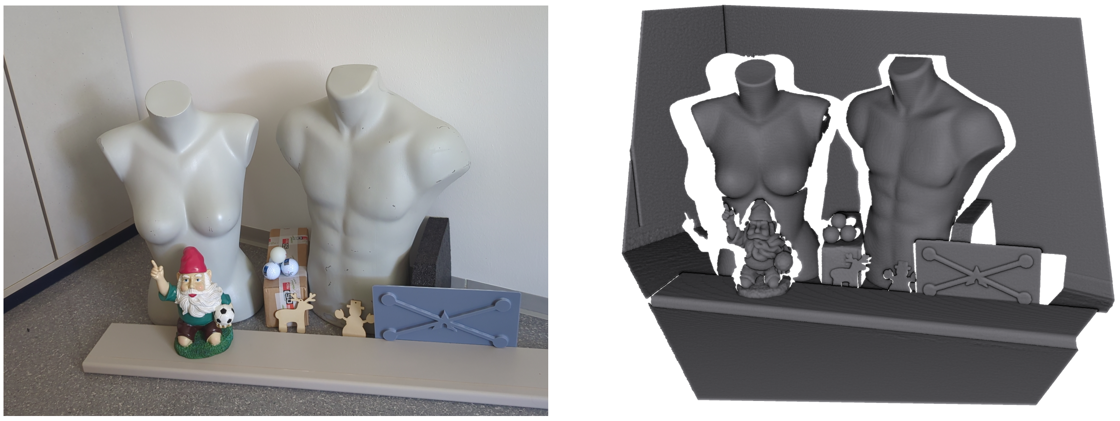


Figure 5.10: Evaluation scene: Adam & Eve. The left shows an image of the scene, while the ground truth pointcloud is shown on the right.

scans along the z-axis. Also, a higher sequence size limits pointcloud density while too many outliers are caused by sequences that are too short.

5.3.3 Scans through a Water Surface

To touch on the motivation of the enclosing research project *Mixed media 3D scanning* [30], we now have a look at the effects of a water surface inbetween our scanner and the scanned object. For a qualitative evaluation of these effects, we compare two scans of the reference dumbbell first placed on the ground, and second fully submerged in a box filled with approximately 10cm water. Figure 5.15 shows these scans with fit spheres in CloudCompare. It is easy to see in the bottom right figure that the submerged scan appears “shrunk” along the z-axis when compared to the non-submerged scan. This shrinkage effectively causes the fit spheres to have a different diameter, because only the top halves of the spheres are visible in the scans. Also, the plane on which the object is positioned appears bent, especially noticeable on the left and right edges in the image.

The cause for these deviations is Snell’s law, where light rays are refracted at the boundary between two media. There are ways to correct for this distortion, given the location and orientation of the water surface [45]. However, no solutions exist yet for the case when waves propagate through the surface.

Still, it is worth to note that even though every ray originating the object of interest is refracted, correspondences are found by our scanner nonetheless. This proves the versatility of our implemented approach. Given these scans, an accurate 3D model can be computed simply by correcting for refraction.

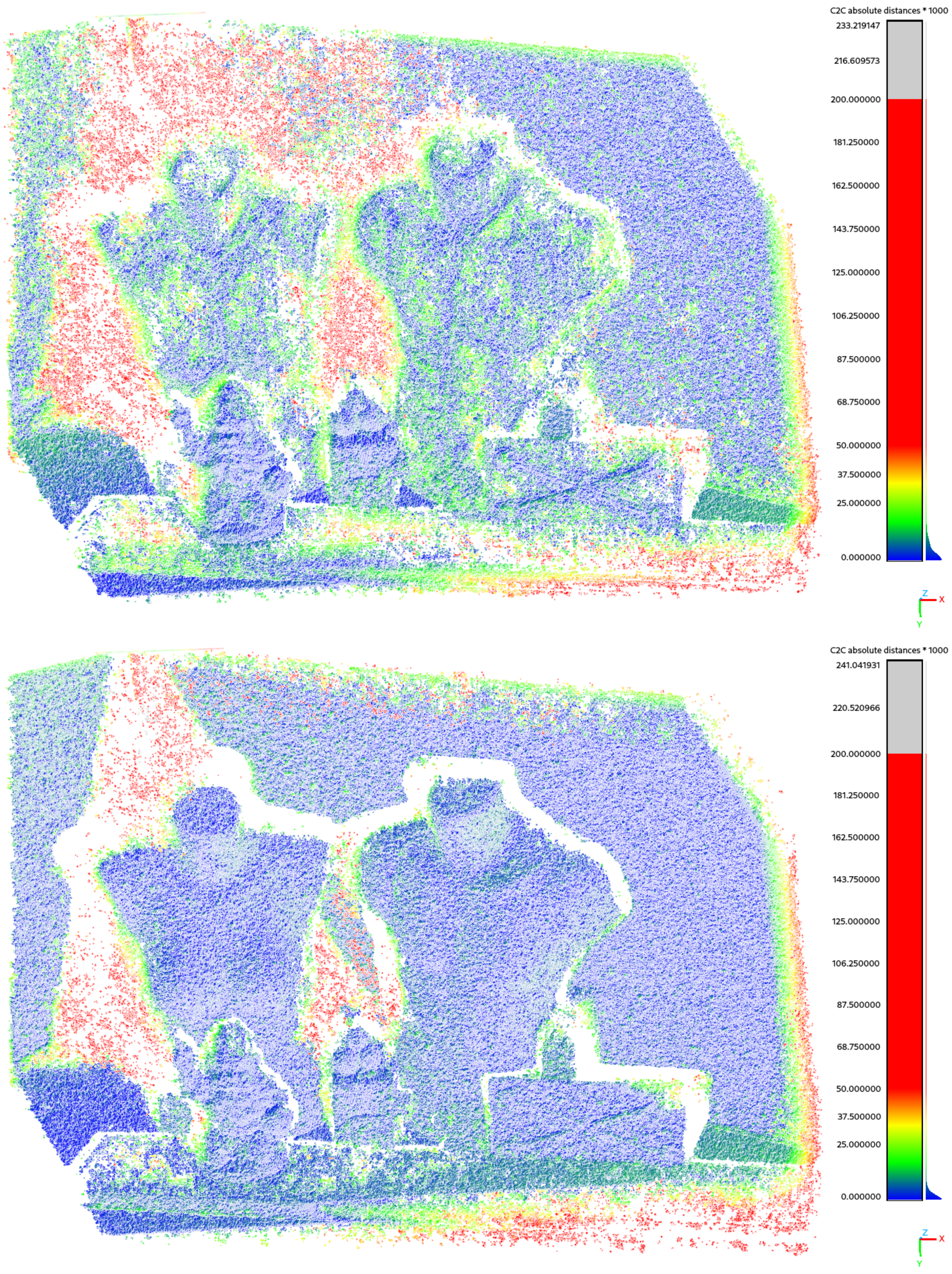
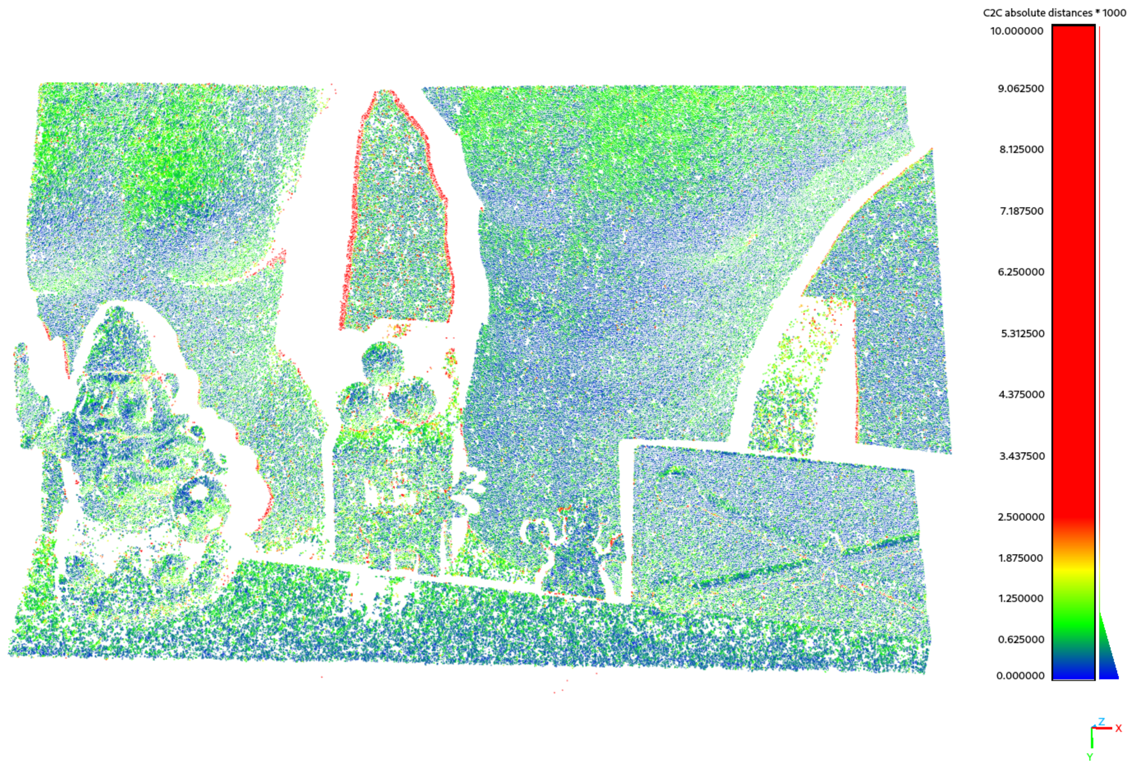


Figure 5.11: Scans of Adam & Eve using Semi-Global Matching. On the top, the raw scene is captured, while on the bottom, additional structure is introduced by projecting a laser point pattern. Scalebars relate color information to numerical errors in millimeters.



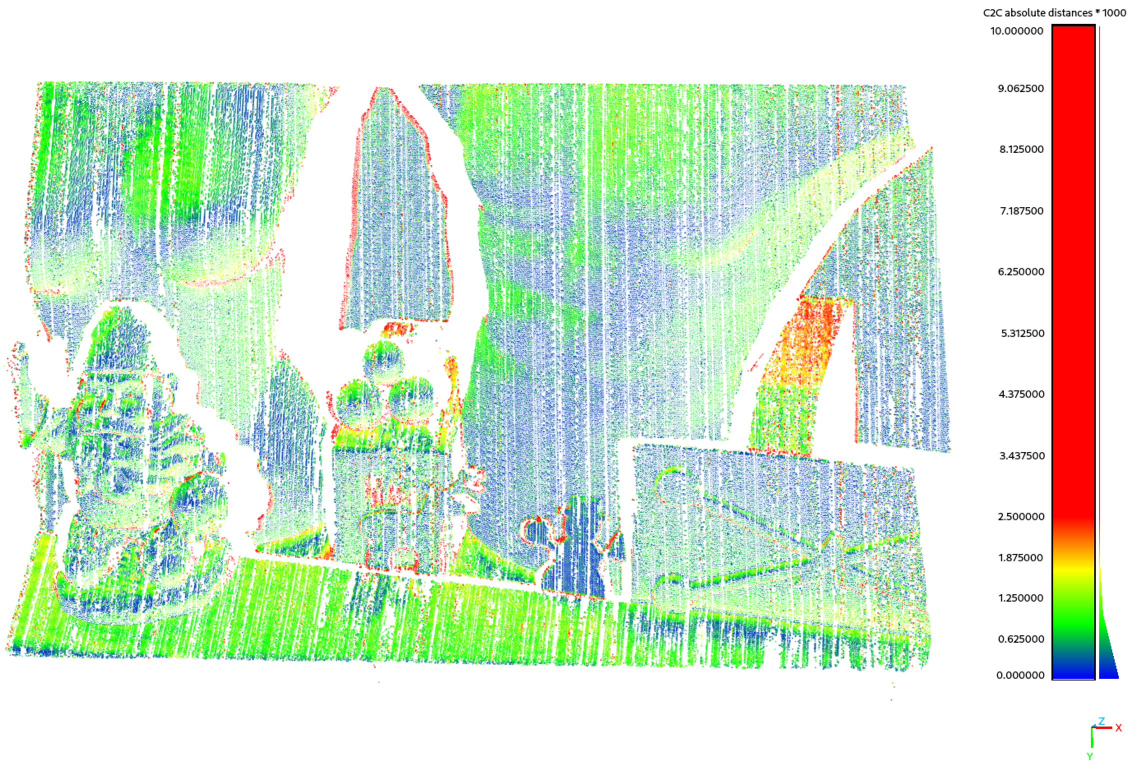
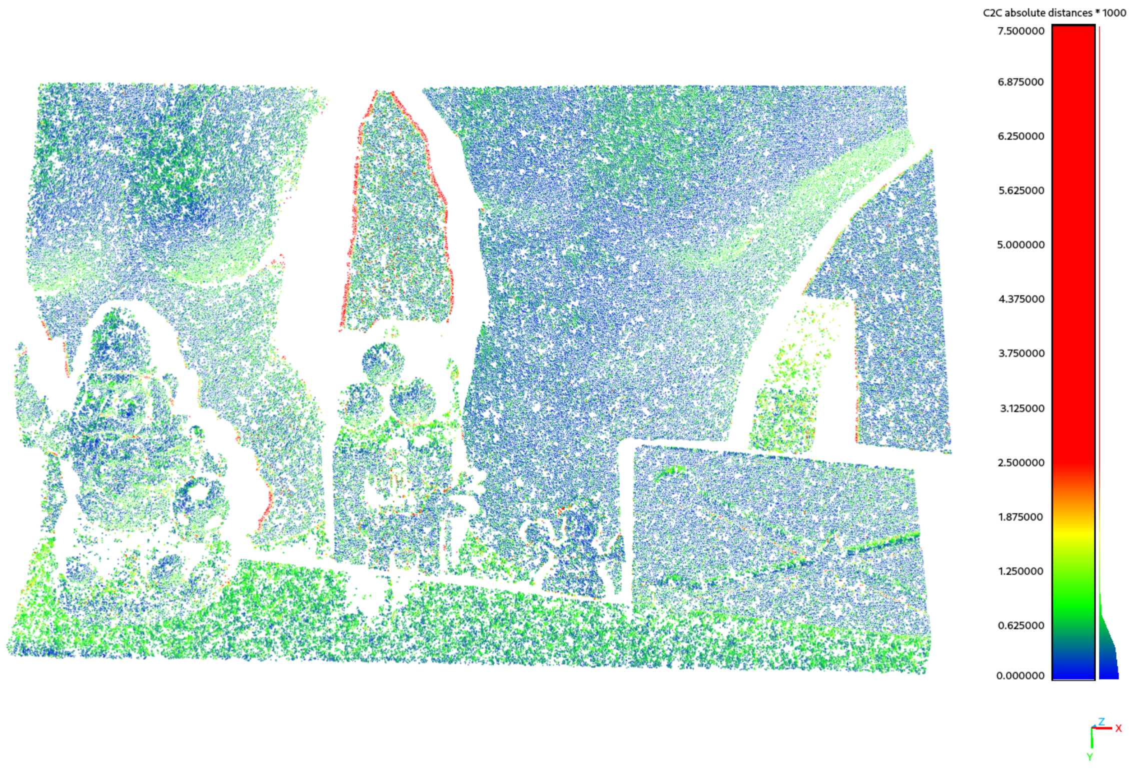


Figure 5.12: Scans of Adam & Eve using a small sequence size. On the top, the scene is captured using speckle projection and 13 images, while 17 are used on the bottom with fringes.



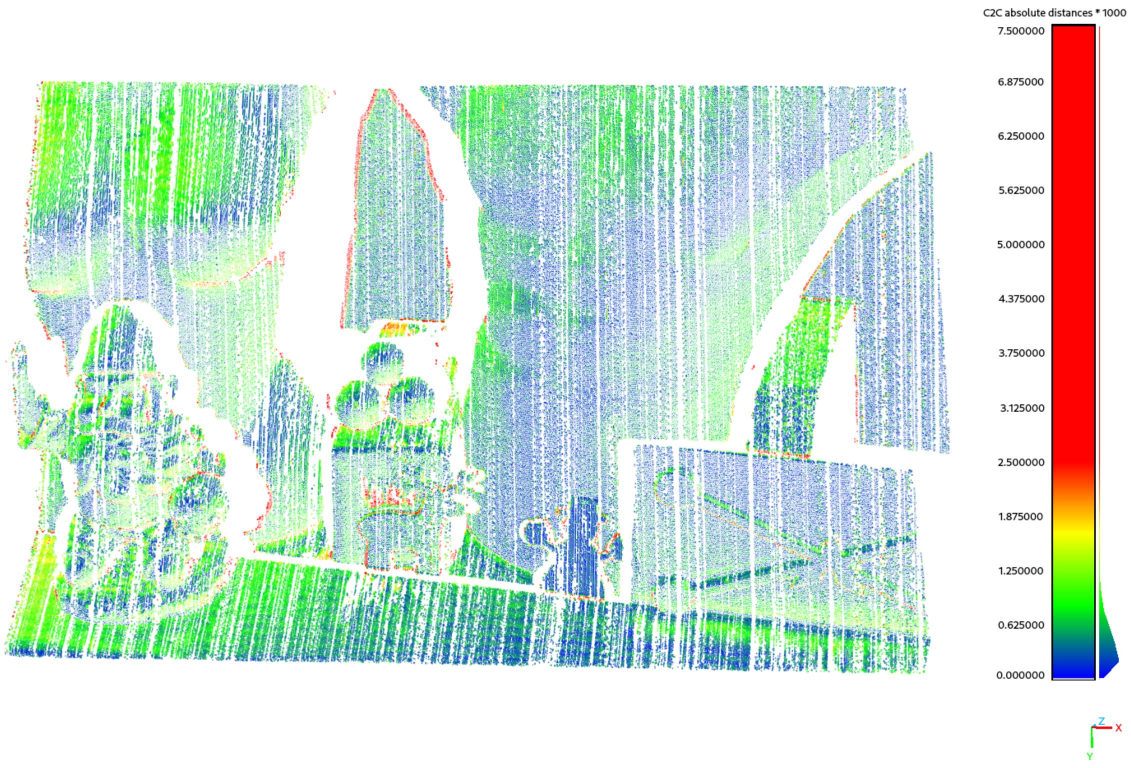


Figure 5.13: Scans of Adam & Eve using a large sequence size. On the top, the scene is captured using speckle projection and 22 images, while the same number is used on the bottom with fringes.

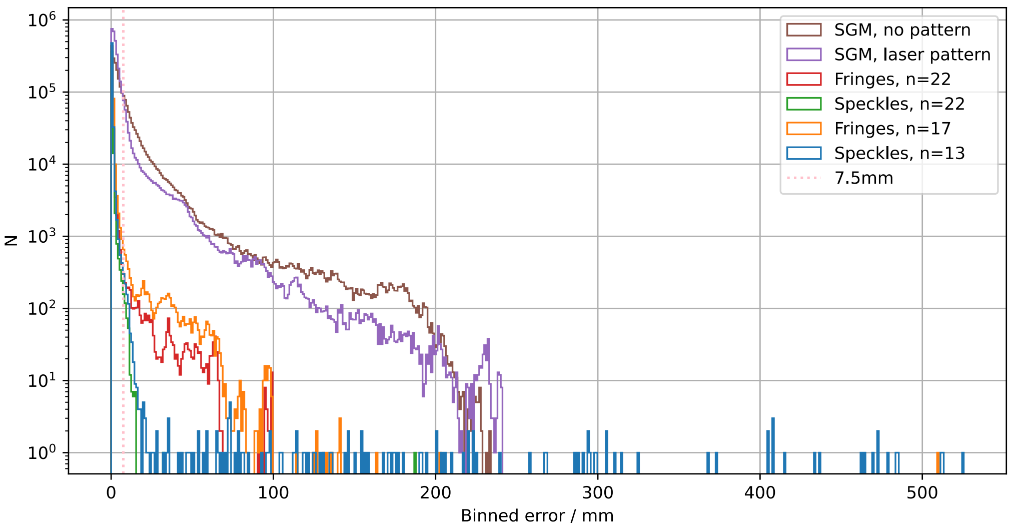


Figure 5.14: Histogram plot of errors between ground truth and each scan of the scene "Adam & Eve". The same format as of Figure 5.9 has been applied. Corresponding scans are shown in 5.11, 5.12 and 5.13.

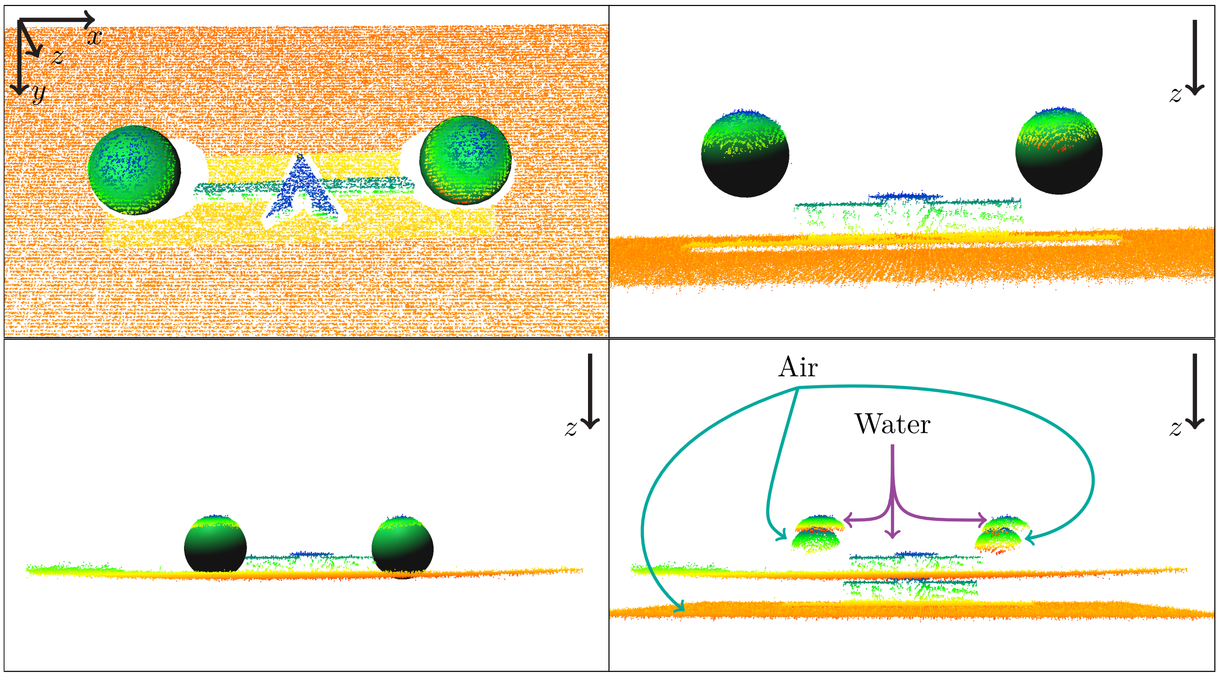


Figure 5.15: Comparison between scans with and without a water surface inbetween. Top left shows the scanned dumbbell from the top to give an overview of its shape, with the same scan on the top right from the side. Both of them include well-fit spheres with approx. 60 mm diameter. Bottom left shows the scan of the submerged dumbbell, with fit spheres of 72 mm diameter. The scan appears "shrunk" along the z-axis. Bottom right shows this effect, where both scans are visible.

Chapter 6

Conclusion

In this work, the necessary steps to develop a structured light stereo vision system for 3D scanning are presented. Starting with camera calibration, the well-known method of Zhang is a suitable way to obtain a camera's parameters. Because it has stood the test of time, calibration of cameras and stereo pairs is a relatively straightforward process, aided by implementations in freely available libraries. Compared with the correspondence problem however, it is clear that the research focus needs to lie on stereo matching. Thus, this work focuses on the implementation of such algorithms.

Similar to Zhang's method of camera calibration, Semi-Global matching can be considered to be a baseline method to compare other stereo matching implementations against. There, implementations already exist and are also freely available. Two reasons speak against Semi-Global matching however: its memory-intensiveness and weakness when it comes to unstructured, plain surfaces. The latter point combined with the fact that adverse, badly-lit environments require a stereo system to be turned into an active one motivates other approaches to do stereo vision. As soon as structured light can be projected into the scene, we may use entirely different methods for matching correspondences between two images. Utilizing a change of this lighting over time, more accurate results can be achieved. This work proves this by comparing the results of SGM against BICOS as a multi-shot method.

To prove the accuracy of the developed stereo system, we measure reference shapes according to industry criteria and show that errors lie in the sub- to low millimeter range: sizes can be measured with errors as low as 0.8 mm (sphere form), lengths with 0.2 mm (sphere spacing), while shape and plane errors with an optimal scanner configuration reach errors of 1 mm. These errors are achieved at a measurement distance of up to 2 m. As for scans of free-form shapes and scenes, the majority of errors lie below 2.5 mm at close range (90 cm) and 7.5 mm at medium range (150 cm).

Because there are no available implementations of BICOS, we also analyze how this algorithm is implemented efficiently using GPU-accelerated parallelism. This is especially motivated by the fact that a fast implementation of stereo matching increases the feasibility of realtime applications of structured light stereo.

Future work includes the analysis of our approach against different light patterns, including – but not limited to – changing laser patterns. Such patterns include rotating laser point patterns and/or lines. Also, investigating the refractive correction of scans of submerged objects has significant merit, especially because uncorrected scans promise dense results. Putting the scanning apparatus into a waterproof enclosure to conduct underwater surveys is another possibility for future research. Focusing on the implemented variant of BICOS supplied by this work, future work should optimize this implementation further, ideally converging into efforts to publish this implementation as a freely available library with support for multiple GPU types or computational frameworks. This way, others could benefit from a fast implementation of this algorithm to make structured light stereo more accessible. □

Bibliography

[1] Joan Batlle, E. Mouaddib, and Joaquim Salvi. Recent progress in coded structured light as a technique to solve the correspondence problem: a survey. *Pattern recognition*, 31(7):963–982, 1998.

[2] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35:269–293, 1999.

[3] Michael Bleier. *Underwater Laser Scanning - Refractive Calibration, Self-calibration and Mapping for 3D Reconstruction.* PhD thesis, Julius-Maximilians-Universität Würzburg, 2023.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Duane Brown. Decentering distortion of lenses. *Photogrammetric Engineering*, 32(3):444–462, 1966.

[6] Duane Brown. Close-range camera calibration. *Photogrammetric Engineering*, 37(8):855–866, 1971.

[7] Tianyu Chang, Xun Yang, Tianzhu Zhang, and Meng Wang. Domain Generalized Stereo Matching via Hierarchical Visual Transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9559–9568, June 2023.

[8] Patrick Dietrich, Stefan Heist, Martin Landmann, Peter Kühmstedt, and Gunther Notni. BICOS — An Algorithm for Fast Real-Time Correspondence Search for Statistical Pattern Projection-Based Active Stereo Sensors. *High-speed Optical 3D Shape and Deformation Measurement, Applied Sciences*, 9(16):3330, August 2019.

[9] Pascal Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6(1):35–49, 1993.

[10] Wolfgang Förstner and Bernhard P. Wrobel. *Photogrammetric Computer Vision.* Springer International Publishing, 2016.

[11] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, June 2014.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[13] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient Large-Scale Stereo Matching. In Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto, editors, *Computer Vision – Asian Conference on Computer Vision (ACCV) 2010*, pages 25–38, Berlin, Heidelberg, 2011. Springer.

[14] Feifei Gu, Zhan Song, and Zilong Zhao. Single-Shot Structured Light Sensor for 3D Dense and Dynamic Reconstruction. *Sensors*, 20(4):1094, January 2020.

[15] Stefan Heist, Peter Lutzke, Patrick Dietrich, Peter Kühmstedt, and Gunther Notni. Experimental comparison of laser speckle projection and array projection for high-speed 3D measurements. In Peter Lehmann, Wolfgang Osten, and Armando Albertazzi Gonçalves Jr., editors, *Optical Measurement Systems for Industrial Inspection IX*, volume 9525, page 952515. International Society for Optics and Photonics, SPIE, 2015.

[16] Stefan Heist, Peter Lutzke, Ingo Schmidt, Patrick Dietrich, Peter Kühmstedt, Andreas Tünnermann, and Gunther Notni. High-speed three-dimensional shape measurement using GOBO projection. *Optics and Lasers in Engineering*, 87:90–96, December 2016.

[17] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 2, pages 807–814. IEEE, 2005.

[18] Heiko Hirschmüller. Stereo Processing by Semi-Global Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, February 2008.

[19] Junhwan Kim et al. Visual correspondence using energy minimization and mutual information. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1033–1040. IEEE, 2003.

[20] Florian Klopfer, Martin Hämmerle, and Bernhard Höfle. Assessing the potential of a low-cost 3-D sensor in shallow-water bathymetry. *IEEE Geoscience and Remote Sensing Letters*, 14(8):1388–1392, 2017.

[21] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical Stereo Matching via Cascaded Recurrent Network With Adaptive Correlation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16263–16272, June 2022.

[22] Yuwei Liu, Pan Ou, Xinqi Xu, and Junhua Sun. Multi-line structured light binocular vision stereo matching method via coarse-to-fine spatial geometric constraints. *Optics & Laser Technology*, 176:110950, 2024.

[23] Luca Lucchese and Sanjit K. Mitra. Using saddle points for subpixel feature detection in camera calibration targets. In *Asia-Pacific Conference on Circuits and Systems*, volume 2, pages 191–195. IEEE, 2002.

[24] John Mallon and Paul F. Whelan. Which pattern? Biasing aspects of planar calibration patterns and detection methods. *Pattern Recognition Letters*, 28(8):921–930, June 2007.

[25] Sarah Martull, Martin Peris, and Kazuhiro Fukui. Realistic CG stereo image dataset with ground truth disparity maps. In *International Conference on Pattern Recognition (ICPR) workshop TrakMark2012*, volume 111, pages 117–118, 2012.

[26] San Mohan, Kasper Broegaard Simonsen, Ivar Balslev, Volker Krüger, and René Dencker Eriksen. 3D Scanning of Object Surfaces using Structured Light and a Single Camera Image. In *2011 IEEE International Conference on Automation Science and Engineering*, pages 151–156. IEEE, 2011.

[27] Srinivasa G. Narasimhan and Shree K. Nayar. Structured light methods for underwater imaging: light stripe scanning and photometric stereo. In *Proceedings of OCEANS 2005 MTS/IEEE*, pages 2610–2617. IEEE, 2005.

[28] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[29] NVIDIA. CUDA C++ Programming Guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`, 2024.

[30] Andreas Nüchter. Mixed Media 3D Scanning. Research proposal, June 2020.

[31] Shingo Otsuka, Akihiro Takagi, Yuta Noto, and Contributors. `https://github.com/fixstars/libSGM`, July 2024. original-date: 2016-02-12, 07:43:44.

[32] Martin Peris, Sara Martull, Atsuto Maki, Yasuhiro Ohkawa, and Kazuhiro Fukui. Towards a simulation driven stereo vision system. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1038–1042. IEEE, 2012.

[33] Yitong Rong, Xuyang Duan, and Jun Han. A high-throughput and low-storage stereo vision accelerator with dependency-resolving strided aggregation for 8-path semi-global matching. *Microelectronics Journal*, 146:106156, April 2024.

[34] Victoria Rudakova and Pascal Monasse. Camera Matrix Calibration Using Circular Control Points and Separate Correction of the Geometric Distortion Field. In *2014 Canadian Conference on Computer and Robot Vision*, pages 195–202, May 2014.

[35] Joaquim Salvi, Jordi Pages, and Joan Batlle. Pattern codification strategies in structured light systems. *Pattern recognition*, 37(4):827–849, 2004.

[36] Martin Schaffer, Marcus Grosse, Bastian Harendt, and Richard Kowarschik. High-speed three-dimensional shape measurements of objects with laser speckles and acousto-optical deflection. *Optics Letters*, 36(16):3097, August 2011.

[37] Martin Schaffer, Marcus Grosse, and Richard Kowarschik. High-speed pattern projection for three-dimensional shape measurement using laser speckles. *Appl. Opt.*, 49(18):3622–3629, 2010.

[38] Daniel Scharstein and Richard Szeliski. Stereo Matching with Nonlinear Diffusion. *International Journal of Computer Vision*, 28(2):155–174, June 1998.

[39] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, April 2002.

[40] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[41] Thomas Schops, Viktor Larsson, Marc Pollefeys, and Torsten Sattler. Why Having 10,000 Parameters in Your Camera Model Is Better Than Twelve. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[42] Thomas Schops, Johannes L. Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A Multi-View Stereo Benchmark With High-Resolution Images and Multi-Camera Videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[43] Robert Spangenberg, Tobias Langner, and Raúl Rojas. Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In *Computer Analysis of Images and Patterns: 15th International Conference, CAIP 2013, York, UK, August 27-29, 2013, Proceedings, Part II 15*, pages 34–41. Springer, 2013.

[44] Engin Tola, Vincent Lepetit, and Pascal Fua. DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2010.

[45] Joschka van der Lucht, Michael Bleier, Florian Leutert, Klaus Schilling, and Andreas Nüchter. Structured-light based 3D laser scanning of semi-submerged structures. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:287–294, 2018.

[46] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik. Optische 3-D-Messysteme - Bildgebende Systeme mit flächenhafter Antastung. VDI/VDE 2634, August 2012.

[47] Paul Viola and William M. Wells III. Alignment by Maximization of Mutual Information. *International Journal of Computer Vision*, 24(2):137–154, 1997.

[48] Kaijie Wei, Yuki Kuno, Masatoshi Arai, and Hideharu Amano. RT-libSGM: An Implementation of a Real-time Stereo Matching System on FPGA. In *Proceedings of the 12th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART '22, page 1–9, New York, NY, USA, 2022. Association for Computing Machinery.

[49] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 756–771. Springer, 2014.

[50] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. Hierarchical Deep Stereo Matching on High-Resolution Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5524, 2019.

[51] Mark Young, Erik Beeson, James Davis, Szymon Rusinkiewicz, and Ravi Ramamoorthi. Coded structured light. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[52] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Computer Vision—ECCV'94: Third European Conference on Computer Vision Stockholm, Sweden, May 2–6 1994 Proceedings, Volume II 3*, pages 151–158. Springer, 1994.

[53] Chen Zhang, Anika Brahm, Andreas Breitbarth, Maik Rosenberger, and Gunther Notni. Single-frame three-dimensional imaging using spectral-coded patterns and multispectral snapshot cameras. *Optical Engineering*, 57(12):1, December 2019.

[54] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000.

[55] Zhengyou Zhang. Camera calibration: a personal retrospective. *Machine Vision and Applications*, 27(7):963–965, October 2016.

[56] Fuqiang Zhong, Ravi Kumar, and Chenggen Quan. A Cost-Effective Single-Shot Structured Light System for 3D Shape Measurement. *IEEE Sensors Journal*, 19(17):7335–7346, September 2019.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, September 2024