Ph.D. Thesis

DTU Compute
Department of Applied Mathematics and Computer Science

Differentiable formulations for inverse rendering

Morten Hannemose

Kongens Lyngby 2020

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

ISSN: 0909-3192

"Any sufficiently advanced technology is indistinguishable from magic"

- Arthur C. Clarke's third law

Summary

Computer vision has become ubiquitous in the modern world and is applied in both industrial and consumer applications, where it is used for things such as pose estimation and 3D scanning.

However, most of the algorithms used for this have a disadvantage. They work by computing positions of points, such as corners or 3D points, and then use only these points to compute the result. This is an effective way of reducing the amount of data taken into consideration, but it disregards possible uncertainties or ambiguities in the detected points.

In this thesis, we will address this and other issues by using inverse rendering. In inverse rendering, we fit a model to the objects present in the image. To fit the model we use optimization, which is faster and significantly more robust when our model is differentiable. This is why we focus on differentiable models in this thesis.

Our goal is to develop practical methods for common problems in computer vision that are more accurate or widely applicable than previous methods. Specifically, we contribute with two methods for common problems using differentiable inverse rendering. The first is a method for camera calibration that uses all pixels of the calibration artifact thereby overcoming problems of relying on a corner detector. The second is a method for 3D scanning, where our method can reconstruct a 3D mesh directly from images taken with a structured light 3D scanner, which enables us to bypass the usual mesh reconstruction step.

In addition to this, we also provide a method for video frame interpolation, where we use a neural network-based method to generate interpolated frames. We train the neural network to minimize the differences between real and interpolated images, as a form of inverse rendering. Additionally, we present a practical method for estimating the pose of an object and a light source, again using a differentiable model. The estimated poses can be used to compare a photograph to a rendering, to quantify their differences.

Finally, we also present applications within industrial automation and augmented reality, where camera calibration and 3D scanning play a key role.

Danish Summary

Datamatsyn er til stede overalt i den moderne verden og anvendes i både industrielle- og forbrugerapplikationer, hvor det bruges til ting som f.eks. estimering og 3D-scanning.

De fleste af de dertil anvendte algoritmer har dog en ulempe. De fungerer ved at beregne positioner af punkter, såsom hjørner eller 3D-punkter, og bruger derefter kun disse punkter til at beregne resultatet. Dette er en effektiv måde at reducere mængden af data, der tages i betragtning, men det ignorerer mulige usikkerheder eller uklarheder i de fundne punkter.

I denne afhandling vil vi adressere denne og andre ulemper ved at bruge invers rendering. I invers rendering tilpasser vi en model til de objekter, der findes i billedet. For at tilpasse til modellen bruger vi optimering, som er hurtigere og betydeligt mere robust, når vores model er differentiabel. Derfor fokuserer vi på differentiable modeller i denne afhandling.

Vores mål er at udvikle praktiske metoder til almindelige problemer i datamatsyn, der er mere nøjagtige eller bredere anvendelige end tidligere metoder. Specifikt bidrager vi med to metoder til almindelige problemer ved hjælp af differentiabel invers rendering. Det første er en metode til kamerakalibrering, der bruger alle pixels i kalibreringsartifakten og derved lørser problemet med at stole blindt på en hjørnedetektor. Det andet er en metode til 3D-scanning, hvor vores metode kan rekonstruere et 3D-trekantsnet direkte fra billeder taget med en struktureret lys 3D-scanner, som gør det muligt for os at forbigå det sædvanlige netrekonstruktionstrin.

Derudover kommer vi også med en metode til interpolering af videobilleder, hvor vi bruger en metode baseret på neurale netværk til at generere interpolerede billeder. Vi træner det neurale netværk til at minimere forskellene mellem virkelige og interpolerede billeder, som en form for invers rendering. Derudover præsenterer vi en praktisk metode til at estimere position og orientering af et objekt og en lyskilde, igen ved hjælp af en differentiabel model. De estimerede positioner og orienteringer kan bruges til at sammenligne et fotografi med en rendering for at kunne kvantificere deres forskelle.

Slutteligt præsenterer vi også applikationer inden for industriel automatisering og augmenteret virkelighed, hvor kamerakalibrering og 3D-scanning spiller en central rolle.

Preface

This Ph.D. thesis was prepared at the section of Image Analysis and Computer Graphics at the Technical University of Denmark (DTU) in fulfillment of the requirements for acquiring a Ph.D. degree in computer science. The work presented in this thesis was funded jointly by DTU Compute and Innovation Fund Denmark through the Flexdraper project.

For as long as I can remember I have had an interest in digital images, and the will to take these images into the computer and use them to accomplish tasks has been a key driving force of my interest.

The topic of this is the application of differentiable inverse rendering to problems in computer vision. Seven papers have been written in the process of completing this Ph.D. thesis. An overview of the papers can be found on page vii. Relevant papers are available to the reader as a part of this thesis as Contributions A to F. Four of these (A to D) are peer-reviewed and have been published. Contribution E is still undergoing peer-review at the time of writing, and we expect to submit Contribution F to a peer-reviewed journal soon. Contribution G is peer-reviewed as well but is left out of the thesis as it deals with a different topic.

This thesis has been supervised by Associate Professor Jeppe Revall Frisvad from DTU and co-supervised by Assistant Professor Jakob Wilm from the University of Southern Denmark (SDU). Professor Thiusius Rajeeth Savarimuthu from SDU was also a co-supervisor. The research activities have taken place at DTU Compute at the Section for Image Analysis and Computer Graphics, except for the external stay at Cornell Tech under Professor Serge Belongie.

Kongens Lyngby, 31 August 2020

Morten Hannemose

Acknowledgements

First and foremost, I would like to thank my supervisor Jeppe Revall Frisvad for his extensive support during my Ph.D. and especially for his clear guidance and feedback, writing assistance, and the many in-depth discussions we have had during my studies.

Additionally, I would also like to thank my co-supervisor Jakob Wilm for his sound advice, guidance, good company, and for introducing me to the wonders of structured light.

I would like to thank Serge Belongie, Harald Haraldsson and the other great people from the Mixed Reality Collaboratory at Cornell Tech for their warm welcome during my external stay.

I would like to thank Anders Bjorholm Dahl and Andreas Bærentzen for many helpful ad-hoc discussions and to thank Henrik Aanæs who helped me find funding for my Ph.D.

Additionally, I would like to thank all of my wonderful colleagues, both past and present, with whom I have had the privilege of sharing many conversations, cups of coffee, and beers throughout these studies. You have all had your part in making coming to work such an enjoyable experience. A special thanks goes to my partner in crime and office mate, Janus Nørtoft Jensen for our many great collaborations both professionally and socially, and for joining me on 11 563 km of road trips.

I want to thank my parents for helping me make the right decisions and always believing in me. I also owe a huge thanks to my girlfriend Laura for her love and support during my Ph.D. and for always being up for discussing a paper or two.

Finally, my special thanks go out to you, the reader, for taking the time to read this far. You are so good!

List of Contributions

Peer Reviewed

A **Morten Hannemose**, Jakob Wilm, and Jeppe Revall Frisvad. "Superaccurate camera calibration via inverse rendering". In: *Modeling Aspects in Optical Metrology VII*. volume 11057. International Society for Optics and Photonics. SPIE, 2019, pages 252–260. DOI: 10.1117/12.2531769. [HWF19]

B L.-P. Ellekilde, J. Wilm, O.W. Nielsen, C. Krogh, E. Kristiansen, G.G. Gunnarsson, T.S. Stenvang, J. Jakobsen, M. Kristiansen, J.A. Glud, **M. Hannemose**, H. Aanæs, J. de Kruijk, I. Sveidahl, A. Ikram, and H.G. Petersen. "Design of Automated Robotic System for Draping Prepreg Composite Fabrics". In: *Robotica* (2020), pages 1–16. DOI: 10.1017/S0263574720000193. [Ell+20]

C **Morten Hannemose**, Janus Nørtoft Jensen, Gudmundur Einarsson, Jakob Wilm, Anders Bjorholm Dahl, and Jeppe Revall Frisvad. "Video frame interpolation via cyclic fine-tuning and asymmetric reverse flow". In: *Scandinavian Conference on Image Analysis*. Springer. 2019, pages 311–323. DOI: 10.1007/978-3-030-20205-7_26. [Han+19]

D Janus Nørtoft Jensen*, **Morten Hannemose***, Jakob Wilm, Anders Bjorholm Dahl, Jeppe Revall Frisvad, and Serge Belongie. "Generating spatial attention cues via illusory motion". In: *Third Workshop on Computer Vision for AR/VR*. 2019. URL: http://people.compute.dtu.dk/jnje/illusory-motion/. [Jen+19]

Under Peer Review

E **Morten Hannemose**, Mads Emil Brix Doest, Andrea Luongo, Søren Kimmer Schou Gregersen, Jakob Wilm, and Jeppe Revall Frisvad. "Alignment of rendered images with photographs for testing appearance models". In: *Applied Optics* (2020). [Han+20]

*These authors contributed equally

Manuscripts

F Janus Nørtoft Jensen*, **Morten Hannemose***, J. Andreas Bærentzen, Jakob Wilm, Jeppe Revall Frisvad, and Anders Bjorholm Dahl. "Surface Reconstruction from Structured Light Images using Differentiable Rendering". In: *Peer Reviewed Journal (To be decided)* (2020). [Jen+20]

Non-Included Peer Reviewed Contributions

G **Morten Hannemose**, Jannik Boll Nielsen, László Zsíros, and Henrik Aanæs. "An image-based method for objectively assessing injection moulded plastic quality". In: *Scandinavian Conference on Image Analysis*. Springer. 2017, pages 426–437. DOI: 10.1007/978-3-319-59129-2_36. [Han+17]

*These authors contributed equally

Contents

List of Abbreviations

AAM Active appearance model

AD Automatic differentiation

BRDF Bidirectional reflectance distribution function

CAS Computer algebra system

CMM Coordinate-measuring machine

CNN Convolutional neural network

DTU Technical University of Denmark

FFT Fast Fourier transform

GPU Graphics processing unit

ICP Iterative closest point

LSM Least-squares matching

PCA Principal component analysis

ReLU Rectified linear unit

SDU University of Southern Denmark

CHAPTER 1

Introduction

1.1 Scope

This thesis focuses on fitting computer models to image data to extract useful information. Our goal is to develop practical methods for common problems in computer vision that are more accurate or more widely applicable than previous methods. We accomplish this goal by providing differentiable models of objects present in the images, such that we can extract the desired information by fitting our models to image data with optimization. Usually, these problems are solved by two or more algorithms, but our approach enables us to solve these problems with a single algorithm, which lets the algorithm internally handle uncertainties and ambiguities without explicit modeling of these.

The contributions in this thesis span multiple areas, from foundational topics as camera calibration, to pose estimation, 3D scanning (see Figure 1.1), and applications of these, with the overall goal of solving these problems in new ways with fewer limitations than previous methods.

Solving these computer vision problems more accurately or in more situations has a vast number of applications, whereof we will address specific applications within:

- industrial automation
- industrial quality control
- digital twins and how to compare to physical samples
- augmented reality

In this thesis, optimization has been used in most of the contributions, yet we have not focused on the optimization itself, but rather used it as a tool. For this reason, we have not described it in further detail, but we refer the reader to Nocedal and Wright [NW06] for more details. The same goes for convolutional neural networks (CNNs) for which we refer the reader to Goodfellow, Bengio, and Courville [GBC16].

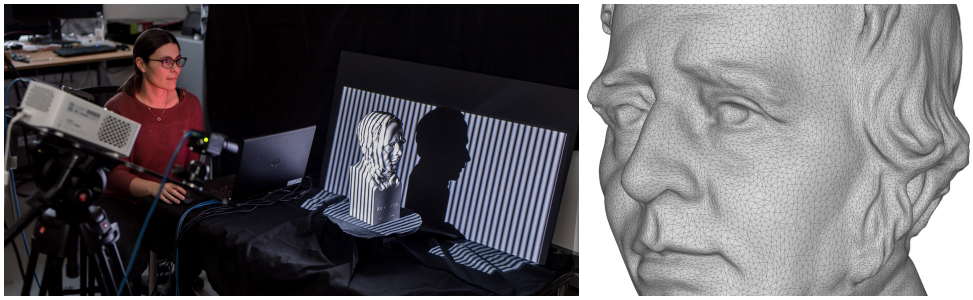


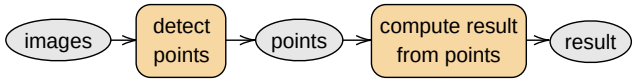
Figure 1.1: 3D scanning is one example of extracting information from images. The figure depicts the 3D scanning of a bust of Hans Christian Ørsted. Left: scanning with the bust with structured light and a stereo camera. Right: a simplified version of the triangle mesh from the scanning. Photos by the author.

1.2 Motivation

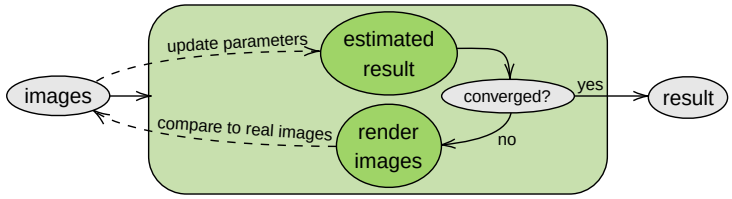
Being able to extract meaningful information from images is the goal of computer vision, and there is therefore, unsurprisingly, a huge amount of work within this field. This has brought us to a point today where we can take a photo of a dog with our phone and it can almost instantly tell us which breed it is. We can also use the stereo camera on our phone to capture a 3D photo of the dog, which gives us the depth of each pixel in the image. To solve these computer vision problems the phone applies deep neural networks and more classic computer vision algorithms. Then we take the output the system has given us and use it to answer the question we actually wanted answered. This could be, whether the dog is family friendly.

It is, however, very often that we fail to take into account that there are uncertainties associated with these answers or that there might not be a single correct answer. The dog could be a crossbreed thus having traits from multiple races. A lot of the time a single output does not tell the whole story of the image because information is discarded in the process of getting the simpler output. This can become a problem when we use these outputs to do further computation to reach the actual answer we were looking for, which is the case in many computer vision algorithms.

Many of the ways we extract meaningful information from images are based on points. What this means is that an algorithm analyses the image and computes points from the image. The algorithm detecting these points will often have a description associated with the detected points, whether it is the location of the corner on a checkerboard, an estimated point in 3D, or a computed feature from a SIFT descriptor [Low04]. These points are then used as inputs for another algorithm that solely uses these points as input to estimate the information that we desired in the first place. However, these point-based methods have some drawbacks such as handling uncertainties, which we



(a) Example of a method that first detects points and then computes the result from these points.



(b) In inverse rendering the result is found by seeing how well images generated from the result match the input images.

Figure 1.2: Conceptual comparison of a point-based method **(a)** and an inverse rendering-based method **(b)**. Note how the original images are used in every iteration of the inverse rendering **(b)**.

will discuss in Section 1.2.3.

These drawbacks led us to focus on another method of extracting information from images in this thesis. This method does not use points and therefore does not have these drawbacks. It is as follows: make a model that can generate the type of images we are working with from a set of parameters. Using this model, search for parameters, such that the image synthesized is as close as possible to the target image. From the estimated parameters we can then extract the information we wanted about the image if we formulated the model correctly. This is the basic description of the method that is referred to by many names such as analysis by synthesis, inverse graphics, or inverse rendering. A conceptual example of inverse rendering is in Figure 1.2.

The search for parameters is done with an optimization method. This is an algorithm that iteratively refines the estimated parameters to minimize or maximize a given objective function. Optimization methods that use access to gradients are faster and more robust but only work for differentiable objective functions, which is why our focus is on differentiable models.

To motivate our contributions further, we will in the two following sections introduce two problems and how they are typically solved using point-based methods. These problems are camera calibration and 3D scanning. We will also use these problems to explain some drawbacks of the point-based methods in Section 1.2.3. These drawbacks also led us to provide improved solutions to these problems by using inverse rendering, which are in Contributions A and F.

1.2.1 Camera Calibration

Any application of computer vision that involves 3D, such as those in industrial automation or quality control, requires having a good understanding of where each camera is in relation to other cameras, and which direction each pixel on the image sensor corresponds to in the real world. Measuring this for a camera setup is called camera calibration.

A camera calibration usually starts with using the camera to take pictures of a checkerboard from a lot of different angles and positions. In each of these pictures, the checkerboard is located, and a corner finding algorithm is used to find the pixel coordinates of the corners. This is often done at sub-pixel precision. As the positions of the checkerboard corners are known in 3D we can compute where the points will project to on the image sensor if we have a calibrated camera and know from where the pictures are taken. The distance between the detected and projected points is the reprojection error. To obtain a camera calibration this error is minimized with respect to the camera parameters and position and orientation of the camera in each image. This is explained in more detail in Section 2.4.

1.2.2 3D scanning

A calibrated camera is the starting point for making a 3D scanner. These have many applications in industrial or recreative settings, such as checking for defects in a produced part or to capture the aforementioned 3D photo of a dog. 3D scanning with a camera can be thought of as equivalent to figuring out how far the object in each pixel is from the camera. This is why cameras are well suited for 3D scanning as each pixel in the camera is its own observation. This makes it much faster to scan an object than methods only making a single observation at a time such as consumer laser distance measures or a coordinate-measuring machine (CMM). There are many ways of doing 3D scanning with cameras, such as time of flight, passive stereo, structured light, and many more. An example of 3D scanning with structured light is in Figure 1.1.

For ease of explanation here, we will focus on methods using a calibrated stereo camera setup, but most of the drawbacks mentioned later will be applicable to single camera methods as well. If we can identify the same point in both images, we can measure the distance of this point to the cameras by triangulation. In structured light, this means that we project a sequence of patterns onto the scene to be able to uniquely identify each point in the scene in both cameras. An example of such a unique coding of the scene is described in Section 2.7 on page 16. In the case of passive stereo, we rely on the texture of the objects in the scene to identify the same points.

For each point in one camera, we need to locate a corresponding point in the other camera. As the stereo setup is calibrated, this implies that we know the position of the cameras relative to each other, we can restrict our search for the point in the other camera to points on a line. This line is the epipolar line. To efficiently search

for matching points, the images can be rectified. This is an operation that warps the images such that each pixel in the rectified image will have its corresponding pixel on the same row, and vice versa, consequently making all epipolar lines horizontal. Once we have two matching points, we can triangulate which point in 3D the two points correspond to, and this process can be repeated for all points in the images.

After doing this for all the corresponding points, we have a point cloud, i.e. a lot of points in 3D but not a solid object. To convert it into a surface we need to apply a surface reconstruction algorithm, such as Poisson reconstruction [KBH06; KH13]. In this step, there is a smoothness/accuracy trade-off. This is because the estimated points will have some noise in them and a surface that goes exactly through all of them would not be desirable in most applications.

1.2.3 Drawbacks of point-based approaches

In each of the two previous examples, the algorithms are split into distinct parts. In the camera calibration, the calibration parameters are estimated only from the detected corner points in the images. This process assumes that all corners are detected correctly without any regard for how certain the estimate is. This means that points detected on an image that was out of focus will have the same weight as points detected on a sharp image. There is no uncertainty passed on between the two steps of the algorithm. These same issues occur in the 3D scanning when the triangulated points are used for reconstructing the surface, without any regard for uncertainty in the estimation of these points.

Another potential issue in the 3D scanning is the interpolation of pixel values. Computing the rectified images requires resampling the pixel values, which means that the pixels the algorithm is operating on are not pure samples from a single pixel anymore. In the best case this will just blur out information, and in the worst case, the interpolation takes place over a discontinuity in the image. If the discontinuity were in focus then there would only have been a single line of pixels affected by the edge before interpolating, but this will have increased to three pixels afterward. This is a problem in structured light where the unique coding on the edge pixels will be a linear combination of the two objects. The linear combination can yield a technically valid coding which will have appeared at a very wrong place in the image, creating an outlier.

1.2.4 Inverse rendering

To overcome the problems of the point-based approaches mentioned in the previous section, we will go back to the image intensities in the image and use these directly in our solutions, similar to what we described on page 4, which we will call inverse rendering. The word “rendering” here, is used to describe all kinds of generation of

an image on a computer. This is important to clarify as it usually refers to image generation methods in computer graphics such as rasterization and ray tracing.

Briefly explained, rasterization projects a triangular mesh to the image plane and computes which pixels each triangle overlaps. A technique called z-buffering is used to ensure that a pixel ends up with the triangle closest to the camera out of the triangles overlapping the pixel. When rendering images in this way, it is common to get jagged edges as each pixel points at only one triangle. To combat this, rasterization uses anti-aliasing, which can be thought of as rendering the image at a higher resolution and downsampling to obtain the final resolution. In this case, each pixel becomes an average of multiple triangles. Because of this, the intensity in a pixel, as a function of the position of an object, is not differentiable or even continuous.

Although rasterization is a hugely useful tool for computer graphics, it is not the best tool of choice for inverse rendering. This is because optimizing a function that is neither differentiable nor continuous is very hard. In fact, we have not used rasterization in any of the works in this thesis. Instead, we have relied on other ways of rendering the image such as using homographies to render images of planes (Section 2.6.1) and other methods using the pinhole camera model.

1.3 Thesis outcome

During the completion of this thesis, we have produced several academic publications. A list of these is on page vii, descriptions and discussions of them are in Chapter 4, and the relevant publications are available as appendices to this thesis.

The main contributions of these publications are in using differentiable inverse rendering to solve problems in computer vision in new ways, which has enabled us to create algorithms that use the information present in the images more directly. We have worked on a wide range of problems in computer vision from camera calibration, to pose estimation, and to applying 3D scans in augmented reality. In particular Contributions A and F introduce new methods with inverse rendering for the well-known computer vision tasks of camera calibration and 3D scanning. In addition to this in Contributions B and D we tackled problems in the interdisciplinary fields of robotics and augmented reality.

1.4 Thesis structure

This thesis is divided into three main chapters. In Chapter 2 we will go through background knowledge that is necessary to understand the contributions made in this thesis. Following this is Chapter 3, where we describe related work in the field of inverse rendering, to familiarize the reader with alternative approaches. Last but not least, in Chapter 4 we present and explain the contributions of this thesis. We also present some additional contributions that are not present in the publications.

In addition to these main chapters, there is an appendix at the end with the manuscripts related to the topic of this thesis. These are Contributions A to F. An overview of these is on page vii.

CHAPTER 2

Background

In this chapter, we will introduce the terminology necessary to understand the contributions of this thesis.

For easier readability we will stick to the following notation: bold and uppercase ($\mathbf{A}$) refers to matrices, bold ($\mathbf{a}$) refers to vectors, and subscript h ($\mathbf{a}_h$) refers to vectors in homogeneous coordinates, such that

$$\mathbf{a}_h = \begin{bmatrix} s\mathbf{a} \\ s \end{bmatrix} = \begin{bmatrix} sa_x \\ sa_y \\ sa_z \\ s \end{bmatrix}, \tag{2.1}$$

when $\mathbf{a}$ is a three-dimensional vector. Also note our use of subscripts x, y, and z to refer to the elements of the vector $\mathbf{a}$.

2.1 Pinhole camera

Most of the works included in this thesis use a camera to gather information about the world. However, because the world has three dimensions, while the image from a camera only has two, we need a model to describe how we project points in $\mathbb{R}^3$ to the image plane of the camera. The most popular model for doing this is the pinhole camera model [HZ03]. This can be written as

$$q_x = \frac{p_x}{f_x} + c_x \tag{2.2}$$

$$q_y = \frac{p_y}{f_x} + c_y, \tag{2.3}$$

where f_x and f_y are the focal lengths, and c_x, c_y is the principal point. We can also write this in matrix form using homogeneous coordinates

$$\begin{bmatrix} sq_x \\ sq_y \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \tag{2.4}$$

$$\mathbf{q}_h = \mathbf{K}\mathbf{p}. \tag{2.5}$$

Here $\mathbf{q}_h$ is (q_x, q_y) represented in homogeneous coordinates. $\mathbf{K}$ is the camera matrix.

These equations assume that the camera is at the origin, and is looking in the z-direction and is not rotated around z. This is the identity rotation. If this is not the case, we need to apply a rotation and translation to the points before projecting them such that the points get transformed to the coordinate system where the camera is at the origin with identity rotation. Let $\mathbf{R}_c$ and $\mathbf{t}_c$ be the rotation and translation of the camera. Then we can project points by

$$\mathbf{q}_h = \mathbf{K}(\mathbf{R}_c\mathbf{p} + \mathbf{t}_c) \tag{2.6}$$
$$= \mathbf{K}[\mathbf{R}_c \quad \mathbf{t}_c]\mathbf{p}_h \tag{2.7}$$
$$= \mathbf{P}\mathbf{p}_h, \tag{2.8}$$

where $\mathbf{p}_h$ is $\mathbf{p}$ in homogeneous coordinates and $\mathbf{P}$ is the 3×4 projection matrix.

2.2 Representing rotations

In (2.7) above, we have represented the rotation as a 3×3 rotation matrix, which is not a very good representation for optimizing as rotation matrices only span a small subspace of all 3×3 matrices. Euler angles parameterize rotations as three rotations around the x-, y-, and z-axis respectively, however, this parameterization has singularities. These are often referred to as gimbal lock [HO18]. This makes Euler angles well-suited only in situations where the optimized rotation can stay close to the identity. In our contributions, we have mostly used quaternions to represent rotations [Sho85]. Quaternions use four numbers to represent rotations and do not have any singularities. The four numbers have the constraint that the squared sum of all the numbers is 1, which can be enforced by constraining the optimization to only take steps in the tangent space where this constraint is satisfied, or less elegantly by re-normalizing the quaternion every time it is needed.

2.3 Lens distortion

When working with real cameras, the pinhole camera model is rarely sufficient to describe the image formation process. We often need to be able to model distortion caused by the camera lens. To do this it is useful to normalize the image coordinates, such that they are centered around the principal point, which lets us readily compute the distance to the principal point (r).

$$\bar{\mathbf{q}} = \mathbf{q} - \mathbf{c} = \tag{2.9}$$
$$\begin{bmatrix} \bar{q}_x \\ \bar{q}_y \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{2.10}$$
$$r^2 = (\bar{q}_x)^2 + (\bar{q}_y)^{\,2} = \|\bar{\mathbf{q}}\|_2^2 \tag{2.11}$$

An often-used lens distortion model is Brown-Conrady [Bro66; Con19]. Here is a simple example with only radial distortion.

$$r^d = r(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.12}$$
$$= r d_r(r) \tag{2.13}$$
$$\bar{\mathbf{q}}^d = \bar{\mathbf{q}} d_r(r) \tag{2.14}$$

There can also be terms that are not only a scaling factor on the radius, such as tangential distortion. The above-given model and most of the more advanced distortion models can be written as scaling for the radial distortion and a translation for the tangential distortion as follows

$$\bar{\mathbf{q}}^d = \bar{\mathbf{q}} d_r(r) + d_t(\bar{\mathbf{q}}). \tag{2.15}$$

2.3.1 Inverting lens distortion

The models presented in the previous section are forward models, i.e. models that take an ideal coordinate and computes where it ends up after going through the lens distortion of the camera model. When doing inverse rendering, we often need to compute the value at a certain pixel coordinate, which involves being able to invert the lens distortion model. This is not to be confused with undistorting an image, which is commonly done for distorted images, where the distortion model never needs to be inverted. This is because one just needs to compute the distorted version of each integer pixel coordinate to get the position where the distorted image should be sampled.

Therefore, knowing how to invert lens distortion is useful , but the details of how to do it are rarely discussed in the literature. Often, we work with only radial distortion. Even if we only have up to r^4 in $d_r(r)$ inverting the distortion requires solving a fifth-degree polynomial, for which no closed-form solutions exist, so we must resort to iterative numerical methods.

An iterative method for inverting lens distortion is implemented in the OpenCV function `undistortPoints()` [Bra00]. This solves (2.15) approximately, by assuming that $d_r(\|\bar{\mathbf{q}}\|_2) \approx d_r(||\bar{\mathbf{q}}^d||_2)$ and $d_t(\bar{\mathbf{q}}) \approx d_t(\bar{\mathbf{q}}^d)$. Now we can iteratively estimate the undistorted $\bar{\mathbf{q}}$ using

$$\bar{\mathbf{q}}_{k+1} = \frac{\bar{\mathbf{q}}^d - d_t(\bar{\mathbf{q}}_k)}{d_r\left(\|\bar{\mathbf{q}}_k\|_2\right)} \tag{2.16}$$

starting from $\bar{\mathbf{q}}_0 = \bar{\mathbf{q}}^d$. The appeal of this method is that it only requires forward distortion to compute, but it has the disadvantage of being relatively slow to converge and does not converge if the assumptions are too far off. This, however, only happens for extreme distortions.

Other approaches have been suggested in the literature such as the one by Drap and Lefèvre [DL16], but these all have other drawbacks in accuracy or speed and are not described here.

2.4 Camera Calibration

Doing a camera calibration means finding the parameters of a camera model such that they describe a camera we have in real-life. For the pinhole camera with lens distortion presented above, it means finding the focal lengths f_x and f_y, the principal point c_x, c_y, and the distortion parameters k_1, k_2, k_3. These all describe the relationship between the direction of incoming light and where hits the image sensor and are referred to collectively as intrinsics. The counterpart to these is extrinsics and describes the relative or absolute poses of the camera(s) being calibrated. Here pose refers to both the position and orientation of an object.

Again, note that the model presented in Sections 2.1 and 2.3 is only one example of a camera model, but there exist many variations on this and even completely different models. A good overview of different models can be found in Schops et al. [Sch+20]. The usual method for computing a camera calibration consists of three steps:

- detecting the calibration object (usually a checkerboard)
- finding features with sub-pixel accuracy
- fitting the camera model

If the setup has multiple cameras, we can capture images of the calibration object in the same pose with each camera and let the relative pose of the camera be part of the estimated parameters. When we capture multiple images of the calibration object, we need to change the pose of the camera between each shot. It is worth noting that changing the pose of the camera(s) or the pose of the calibration object are equivalent, as long as the cameras do not change their poses in relation to each other. We will restrict our explanation to 2D checkerboards, but the same three steps apply for other calibration objects, such as three-dimensional ones.

There are multiple ways of detecting the checkerboard. One of them uses image filtering to find the lines of the checkerboard and combines this with where there are saddle points [Pla+14]. This, however, only gives us the corner points of the checkerboard at integer pixel coordinates, which needs to be refined further to get a high-quality camera calibration. A common method for this was created by Förstner and Gülch [FG87], which is implemented in the OpenCV function `cornerSubPix()` [Bra00]. This uses the observation that the image gradients around a saddle point should be orthogonal to the vector pointing to the saddle point, and iteratively refines the detected point to maximize this. Later approaches have achieved higher sub-pixel

precision in locating the corner by fitting a polynomial surface to the image intensities around the corner [Ha+17] or using that the intensity at an offset from the corner should be the same at the negative offset [Sch+20].

The final step of the camera calibration is to estimate the intrinsics. The most used method for this is the one introduced by Zhang [Zha00]. Recall that we compute the camera calibration by minimizing the reprojection error. The first step in Zhang's method is to compute a linear solution that minimizes an approximation of the reprojection error, followed by non-linear optimization to minimize the actual reprojection error.

To increase the accuracy of the camera calibration further, this step is sometimes followed by bundle adjustment [Tri+99]. Recall that in the previous estimation, the positions of the points of the checkerboard in $\mathbb{R}^3$ were known. However, the calibration artifact is manufactured by some physical process and will have deviations from the ideal shape. This can be something as simple as a piece of paper not being perfectly flat. In the bundle adjustment step then all parameters are optimized again, in addition to the positions of the features on the calibration artifact. This does of course introduce more degrees of freedom which can lead to overfitting, but with enough images, this is not a problem.

2.5 Triangle meshes

When we 3D scan an object, being able to represent it as a surface instead of a point cloud is a more efficient representation, as it is more efficient to do computations on and takes up less space when stored. The most common way of representing surfaces on the computer is with a triangle mesh. A single triangle of the mesh is given by three vertices which are points in $\mathbb{R}^3$. These are connected by lines, such that they form a triangle in $\mathbb{R}^3$. The inside of the triangle is then describing the surface of the object. We can create more triangles, and if the triangles share some of their edges such that the same two vertices are part of two different triangles, we can describe a larger surface. If all edges in a triangle mesh are shared with another triangle, the mesh describes a closed surface. A triangle edge can also be shared by more than two faces, in the case of a triangle mesh that contains multiple materials.

Triangle meshes are a very efficient representation of a surface when working with a graphics processing unit (GPU). Additionally, with a triangle mesh, we are enforcing that the represented surface is locally planar which is a smoothness constraint. We will exploit both properties in Contribution F. An example of a triangle mesh is on the right in Figure 1.1 on page 3.

2.6 Homographies

As mentioned in the introduction, we are interested in using inverse rendering for many problems, one of them being camera calibration. For this we need a differentiable way of rendering a plane with a known texture. To do this we need to know where to sample the texture-function of the plane in each pixel. Using the undistorted pixel coordinate as we obtained in the previous section, we can use a homography to map the pixel coordinates on the image plane to the local coordinates on the plane of the calibration object. A homography is a full-rank 3×3 matrix that can describe the mapping between the image planes of any two cameras that are viewing the same plane in 3D. The mapping uses homogeneous coordinates, like so

$$\mathbf{b}_h = \mathbf{H}\mathbf{a}_h = \tag{2.17}$$

$$\begin{bmatrix} sb_x \\ sb_y \\ s \end{bmatrix} = \mathbf{H} \begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix}, \tag{2.18}$$

where $\mathbf{H}$ is a homography that maps from $\mathbf{a}$ to $\mathbf{b}$. Inverting the homography matrix gives the mapping that goes the other way, which can be done analytically as shown here for the sake of completeness

$$\mathbf{H}^{-1} = \frac{1}{\det(\mathbf{H})} \begin{bmatrix} H_{22}H_{33} - H_{23}H_{32} & H_{13}H_{32} - H_{12}H_{33} & H_{12}H_{23} - H_{13}H_{22} \\ H_{23}H_{31} - H_{21}H_{33} & H_{11}H_{33} - H_{13}H_{31} & H_{13}H_{21} - H_{11}H_{23} \\ H_{21}H_{32} - H_{22}H_{31} & H_{12}H_{31} - H_{11}H_{32} & H_{11}H_{22} - H_{12}H_{21} \end{bmatrix}. \tag{2.19}$$

As this is a closed-form expression, the derivative of it can be computed. Note that the inverse involves the determinant of $\mathbf{H}$ as well, but as homographies work purely with homogeneous coordinates they are also scale-invariant, and the division by the determinant can be skipped when implementing it. For more details on homographies, the reader is referred to Hartley and Zisserman [HZ03].

2.6.1 Homography based rendering of planes

With the 3×4 projection matrix $\mathbf{P}$ from (2.8) we can project points from the global coordinate system to the image plane of the camera. When we combine this with the rotation and translation $(\mathbf{R}_p, \mathbf{t}_p)$ of the plane in $\mathbb{R}^3$ that we want to render, we can project points from the plane as follows

$$\mathbf{q}_h = \mathbf{P} \begin{bmatrix} \mathbf{R}_p & \mathbf{t}_p \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}, \tag{2.20}$$

where (u, v) is a point on the plane. The third coordinate is zero, as we have defined our plane in the x-y directions, such that $z = 0$ everywhere on the plane. The fourth

coordinate is one because it is in homogeneous coordinates. We can remove the column of the last matrix that is multiplied with zero

$$\mathbf{q}_h = \mathbf{P} \begin{bmatrix} R_{c,1} & R_{c,2} & \mathbf{t}_p \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.21}$$

$$= \mathbf{H} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{2.22}$$

where $R_{c,i}$ is the i^{th} column of $\mathbf{R}_c$, and $\mathbf{H}$ is a 3×3 matrix that maps points on the plane to the image plane of the camera with homogeneous coordinates. It is therefore a homography. When we invert this homography, it maps from the image plane to the plane we want to render. We can therefore render the plane by mapping each pixel with the homography, which gives us the point where we should sample the texture function for each pixel.

2.7 3D structured light scanning

As we mentioned in the introduction, 3D scanning a scene with structured light requires the projection of a series of patterns onto the scene that can uniquely code each point in the scene. In this section we will introduce phase shifting with two patterns and unwrapping with the heterodyne principle [RRT97], which we have used in multiple of our contributions. There are many other ways to do this, but we have chosen this method for its high accuracy, yet simple implementation.

We project the patterns such that they vary in a direction that is approximately parallel to the epipolar lines. For a camera setup with horizontal separation between the cameras this means horizontally varying patterns. We will code the x-coordinate of the projector with a linear global phase signal, where the left side of the projector has $\theta = 0$ and the right side has $\theta = 2\pi$. We will now describe how to generate patterns to project, and how to estimate θ in each pixel of the captured images.

We can now project a sinusoidal pattern onto the scene that has a total of n_1 periods. We will shift this pattern s_1 times, giving us s_1 different images we need to project

$$P_1 = \left\{ \cos\left(n_1\theta + \frac{2\pi s}{s_1} \right), \quad s \in \{0, 1, \ldots, s_1 - 1\} \right\}. \tag{2.23}$$

We do the same with n_2 periods, and generate s_2 of these patterns,

$$P_2 = \left\{ \cos\left(n_2\theta + \frac{2\pi s}{s_2} \right), \quad s \in \{0, 1, \ldots, s_2 - 1\} \right\}. \tag{2.24}$$

As these patterns lie in the range $[-1; 1]$ we scale them to the intensity range of the projector before projection. We can now fit a sinusoid to the intensities observed in each pixel for the images obtained when projecting P_1 and P_2. We will refer to the phase offset fitted for these as θ_1 and θ_2 respectively. The sinusoid can be fitted in several ways, such as with a fast Fourier transform (FFT) or a linear system of equations. The number of shifts s_1 and s_2 needs to be four or larger, with larger values giving a more accurate estimate, due to less influence of noise. We have often used values of $s_1 = 16$, $s_2 = 8$.

The phases θ_1 and θ_2 are measurements of θ with wrapping. As our estimates will include some noise, θ_1 and θ_2 are given by

$$\theta_1 \equiv n_1\theta + \epsilon_1 \pmod{2\pi} \tag{2.25}$$
$$\theta_2 \equiv n_2\theta + \epsilon_2 \pmod{2\pi}. \tag{2.26}$$

We can do an approximate unwrapping by applying the heterodyne principle. We refer to this as the phase cue

$$\theta_c \equiv \theta_2 - \theta_1 \pmod{2\pi} \tag{2.27}$$
$$\equiv (n_2 - n_1)\theta - \epsilon_2 + \epsilon_1 \pmod{2\pi}. \tag{2.28}$$

If we have chosen n_1 and n_2 such that $n_2 = n_1 + 1$, then the phase cue is a noisy measure of θ

$$\theta_c \equiv \theta - \epsilon_2 + \epsilon_1 \pmod{2\pi}. \tag{2.29}$$

We can use the phase cue to compute how many times θ_1 has wrapped. This is the order

$$o_1 = \left\lfloor \frac{n_1\theta_c - \theta_1}{2\pi} \right\rceil, \tag{2.30}$$

where $\lfloor\cdot\rceil$ means rounding to the nearest integer, which makes it robust to noise. Then we can unwrap θ_1 with our order o_1

$$\theta_{1,uw} = \frac{2\pi o_1 + \theta_1}{n_1} \pmod{2\pi}. \tag{2.31}$$

This should be equal to θ, except for the noise. However, note that θ_1 here is divided by n_1, which drastically decreases the influence of the noise-term ϵ_1. $\theta_{2,uw}$ can be computed similarly to $\theta_{1,uw}$, and we can take the average to get the most accurate estimate of θ

$$\theta_{uw} = \tfrac{1}{2}(\theta_{1,uw} + \theta_{2,uw}) \pmod{2\pi}. \tag{2.32}$$

This final average should be computed in a way that is angle-aware, such that $(\epsilon, 2\pi - \epsilon)$ would average to something close to 0.

Now that we have estimated θ in each pixel of the camera, and if we have a stereo camera setup, we can look for the point with the corresponding θ along the epipolar line in the other camera. Once we have found this point, we can triangulate the position of the point in $\mathbb{R}^3$. If we have a calibrated projector and a single camera, we can still triangulate the point, as we already know the x-coordinate of this point in the projector. Once we have triangulated all the points, we have a point cloud.

It is worth noting that all the patterns projected with the projector are continuous. Because of this, a projector with a lower resolution will still generate an approximately continuous signal, especially if its slightly out of focus. Because of this, the resolution of the projector is not very important and can be an order of magnitude smaller than the resolution of the camera with the system still being able to perform well.

We will finally discuss the error propagation in phase shifting. The phases θ_1 and θ_2 are least-squares fits to their respective sets of images (P_1 and P_2). Our unwrapped phase θ_{uw}, is however not the least-squares fit to P_1 and P_2. Once we rectify the images and match and triangulate the points, the points in the point cloud are further away from being least-squares fits to the original data. This happens because each step of this process at best is a least-squares fit to the output of the previous step.

2.8 Pose estimation

Recall that the pose is the position and orientation of an object. Pose estimation can refer to multiple different tasks, but central to all of them is to estimate the pose of one or more objects. For example, doing a camera calibration involves estimating the pose of the camera for each picture. It could also be human pose estimation where the task is to estimate the pose of each body part of a person in one or more images. In this thesis we will use it to refer to the problem of estimating the pose of an object in one or multiple images.

The most common way of estimating the pose of an object uses points on the object in 2D and the corresponding position of these points in 3D. From this the pose of the object can be found as the non-trivial least-squares solution to a linear system of equations. However, establishing these correspondences between 2D and 3D points is by no means an easy problem. One way of solving it is to use a feature descriptor such as SIFT [Low04] on each point for matching or to attach a unique marker to each point of the object.

More recent approaches use a large database of images from the object from almost every viewing angle. A statistical model is then fitted to these, such that the pose of the object can be inferred. Examples of this is in the related work of Contribution E.

2.9 Obtaining derivatives

Recall that we are interested in being able to optimize a digital model to fit our observations as well as possible. To do this we need to know how a change in the parameters of our model affects the result. Gradients are able to describe the shape of the objective function around the current parameter estimate, which helps the optimization algorithm decide in which direction and and how far it should take its next step.

When we are writing the code for a program to solve a problem, we must decide how we obtain these gradients. In this thesis, we have used four different methods for obtaining gradients, all of which are widely used. In the following sections we will go through these.

2.9.1 Analytical gradients

Using analytical gradients is often considered the gold standard of computing derivatives. To obtain these one simply takes out pen and paper or a computer algebra system (CAS) and uses this to derive the explicit mathematical formulas for the gradients of the objective function. Once this has been done and implemented in code, they are very efficient to use as the gradients only involve the exact computations needed to obtain them. However, there is a large overhead involved for the person implementing the method, as implementing the derivative is often as much work or more than implementing the objective function and introduces the risk of having an error in the derivation of the derivatives.

2.9.2 Automatic differentiation (AD)

Automatic differentiation is a term that covers several methods that can compute derivatives of a function, automatically and without using approximations. They do this by storing additional information when evaluating the function, that can be used to apply the chain rule recursively to evaluate the derivatives. The following two sections introduce the two most used methods for AD: reverse mode AD and forward mode AD. For a more thorough introduction to the topic of AD the reader is referred to the book by Griewank and Walther [GW08].

2.9.2.1 Reverse mode AD

In situations where it is necessary to compute the derivatives of a small number of variables with respect to a large number of input parameters, reverse mode AD is the most efficient. It has a long history, but its usage has exploded in recent years as it is used to train neural networks, but when using it for this purpose it is referred to as backpropagation [RHW86].

Computing reverse mode AD consists of systematic recursive application of the chain rule and has two steps: a forward and a backward pass. During the forward pass, the function is evaluated as usual, but it is necessary to store the function evaluation as a graph with each node only doing a simple computation. In the backward pass, one starts from the output of the function, where the derivative of the output is 1 per definition. Then the computation goes back through the graph, and for every node the derivative of the output with respect to the inbound value is computed using the chain rule. An illustration of this concept is in Figure 2.1. The green numbers in the

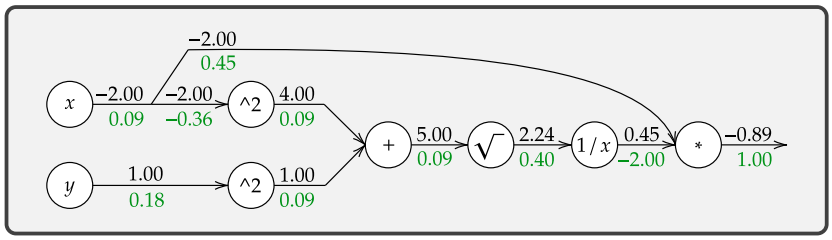


Figure 2.1: Example of using reverse mode AD to compute $\frac{\partial}{\partial x} f_x(x, y)$ and $\frac{\partial}{\partial y} f_x(x, y)$, where $f_x(x, y) = \frac{x}{\sqrt{x^2+y^2}}$, $x = -2$, and $y = 1$. Green numbers contain the derivative of the output with respect to the variable above. First, the function (black numbers) is computed from left to right, and then the derivative of the output (green numbers) is computed by applying the chain rule from right to left, meaning that the resulting derivatives are all the way to the left.

figure are computed by evaluating the local gradient of the unit and multiplying it with the gradient to the right of it. For example the derivative of the output with respect to the output from the square root in Figure 2.1 is 0.4 as

$$\underbrace{\left(-\frac{1}{2.24^2}\right)}_{\substack{\text{derivative of} \\ 1/x \text{ at } 2.24}} \cdot(-2.00) = 0.40. \tag{2.33}$$

2.9.2.2 Forward mode AD

In a situation where we have a large number of variables that we need the derivatives with respect to a smaller number of input parameters, forward mode AD is a good choice. With it, we can compute the derivative of any number of outputs, which makes this a very efficient method when we want the Jacobian matrix, as is required for Levenberg-Marquardt [Lev44; Mar63]. Levenberg-Marquardt is a pseudo-second-order method for nonlinear optimization that switches back to gradient descent when it is far from the minimum.

The principle of forward mode AD is similar to reverse mode AD in that the computation is split into its elementary operations, the difference being that each number is augmented with a vector containing the derivatives of this number with respect to each of the input parameters. Therefore, for the input parameters these vectors will be one-hot encoded, which means that they will have zero everywhere, except for a one at the vector index corresponding to the input. The function is then evaluated only once without storing the computation graph, and each arithmetic operation uses the chain rule to compute the corresponding vector of its output. This can be implemented in software using custom classes and overloading common operators. An example of using forward mode AD is in Figure 2.2.

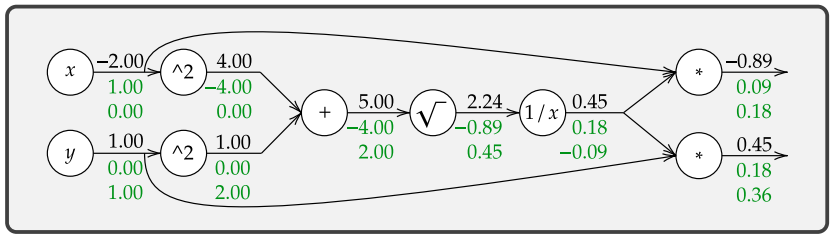


Figure 2.2: Example of using forward mode AD to compute $\frac{\partial}{\partial \mathbf{p}} f_x(x, y)$ and $\frac{\partial}{\partial \mathbf{p}} f_y(x, y)$, where $\mathbf{p} = [x \quad y]^\mathrm{T}$, $\mathbf{f}(x, y) = \frac{\mathbf{p}}{\sqrt{x^2+y^2}}$, $x = -2$, and $y = 1$. The two green numbers below every variable are the derivatives of the black number with respect to x and y respectively. All computations are carried out from left to right, the green numbers are computed using the chain rule, and the resulting derivatives are thus all the way to the right.

Additionally, forward mode AD with a single input parameter is equivalent to using dual numbers, which is a mathematical formalization of forward mode AD that we will not cover, but instead refer the reader to Hoffmann [Hof16]. Jets are a similar mathematical concept that can describe forward mode AD with multiple input parameters [Pus96].

2.9.3 Finite differences approximation

Finite differences are a method to approximate the gradient of a function using only evaluations of the function itself. This makes it a very simple method to implement when the function is already implemented. They are thus very well suited to check whether an implementation of analytical gradients or AD is correct.

Finite differences can be derived from Taylor's theorem. The first order Taylor series

of f at x can be written as

$$f(x+h) = f(x) + \frac{d}{dx} f(x)h + O(h), \tag{2.34}$$

where $O(h)$ is the error term. We can rearrange this to get an approximation of the derivative

$$\frac{d}{dx} f(x) = \frac{f(x+h) - f(x)}{h} + O(h). \tag{2.35}$$

This is forward differences, and the error is linear in the step size h. Backward differences is similar except that the Taylor series is evaluated at $f(x-h)$. We can reduce the error to $O(h^2)$ by starting from a second order Taylor series

$$f(x+h) = f(x) + \frac{d}{dx} f(x)h + \frac{1}{2}\frac{d^2}{dx^2} f(x)h^2 + O(h^2). \tag{2.36}$$

This can also be evaluated $x - h$

$$f(x-h) = f(x) - \frac{d}{dx} f(x)h + \frac{1}{2}\frac{d^2}{dx^2} f(x)h^2 + O(h^2) \tag{2.37}$$

We can subtract (2.36) from (2.37), so the zeroth and second order terms cancel such that we obtain

$$\frac{d}{dx} f(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2), \tag{2.38}$$

which gives us the central differences approximation of the gradient. When using finite differences to compute the derivatives with respect to many parameters we need twice the number of function evaluations with central differences as we do with forward differences, as $f(x)$ is the same for all parameters. There are many other variants of finite differences for computing gradients, even of higher order, but forward, backward, and central differences are the most widely used.

The major disadvantage of finite differences is that they are very slow to compute, as they require at least one additional function evaluation per parameter we need the gradient with respect to. They are also the only of the presented approaches that only yield approximations of the gradients, which can be a problem for the convergence of the optimization. Another problem is that it is not a parameter-free method, but requires the user to choose the parameter h. This can be fixed to a small percentage of x which works well in most cases, but still runs into problems once x is very close to zero.

CHAPTER 3

Related Work

Attempting to solve problems in computer vision by creating a model to generate images and finding the model parameters that best match the observed images is the core of analysis by synthesis which is also referred to as vision by inverse rendering. There is a long tradition of using this approach to solve problems in vision, which goes back to early work in the field [Hor77; Gre81].

Much of the work that has been done in differentiable rendering has focused on making models that are differentiable with respect to the appearance of the scene, but not with respect to the position or shape of the objects in the scene [KPC93; Mar98; Max+19]. Such work is based on methods that are differentiable with respect to the illumination and appearance of the scene, however because they only work with the appearance these methods are not differentiable with respect to the geometry. Therefore, we do not consider them to be relevant in the context of this thesis.

To recover the parameters that make a rendering model best describe a scene, optimization is an often-used approach. In some situations, gradients are infeasible to obtain, which makes solving the optimization problem slower and less robust. In these situations, most people rely on the Nelder-Mead method, also known as the downhill simplex method, which only needs function evaluations to optimize the function [NM65]. Rockwood and Winget [RW97] worked with fitting meshes to images. They were not able to differentiate their specific rendering process, and they used a combination of simulated annealing and the Nelder-Mead method to optimize their objective function. However, having access to gradients can greatly improve the convergence, speed, and robustness of the optimizer. These factors have created great interest in making rendering differentiable.

The following four sections will present different approaches that have been taken to make rendering differentiable. First, in Section 3.1 we will present task specific methods, where the renderer only works for a very specific task, which gives more freedom to make assumptions. Following this in Section 3.2 we present work on creating differentiable renderers able to render arbitrary scenes. Then in Section 3.3 we present methods that render images by using models that get pixel intensities from one or more other images, and finally in Section 3.4 we present methods that modify the image formation process to make it easier to differentiate.

For an in-depth survey of general inverse rendering methods, the recent survey paper

by Kato et al. [Kat+20] is recommended.

3.1 Purpose built methods

In many cases, it is sufficient to create a renderer for a specific task, which enables more simplifications and assumptions to be made such that computing the derivatives is an easier task.

Early work following this approach was done by Fua and Leclerc [FL95], in which they describe a method for surface reconstruction from multiple views based on optimization. However, to have a differentiable objective function they do not render images and compare the pixel values directly. Instead they use a set of predefined points on each triangle face that they project to the images and interpolate the intensity of this point in each image. Their objective function is then to minimize the variance of the interpolated intensities for each predefined location, which is simpler to compute derivatives of. In addition to this they also have a prior on the shape and that the albedo of the surface is smooth. They used Maple (a CAS software) to compute and implement the gradients.

The work by Smelyansky et al. [Sme+02] lies very close to the spirit of this thesis. In the paper they create a differentiable renderer for triangular meshes and use it to optimize both the vertex positions, and the camera positions by minimizing the pixel-wise difference to observed images. They use a Lambertian reflectance model to render the surface and have an albedo for each vertex that is also part of the parameters they optimize. In each pixel they compute the intensity as an average of all triangles within the pixel, weighted by how much of the pixel they take up. This is what makes their renderer differentiable. As this is a very simple way of rendering, they introduce the assumption that there are no shadows or occlusions in the observed images.

The works by Rehder et al. [Reh+17; RS17] use differentiable rendering to render checkerboards for camera calibration of a moving camera. Their rendering uses a polynomial model for illumination of the checkerboard and incorporates both rolling shutter effects and sampled motion blur, which they use to estimate the shutter time and the rolling shutter of a camera. They only render and compare the images on the edges of the checkers.

Gkioulekas et al. [Gki+13] computed derivatives of renderings which they used to do inverse rendering to measure the physical light scattering properties of various materials. They did this by shining a collimated laser beam through a sample of the liquid and observing the resulting scattering for varying angles. They compute gradients with finite differences but can compute them efficiently by reusing light fields. Additionally, they exploit the noisy images generated when rendering with ray-tracing to obtain noisy gradient estimates, which enables them to use stochastic gradient descent to optimize their parameters.

A similar approach was used by Velinov et al. [Vel+18] to estimate the thickness of the tooth enamel. They did because doing realistic renderings of teeth depends not only on the shape of the teeth but also geometry below the surface. They restricted the enamel shape to vary along certain directions to obtain a practical parameterization of the enamel thickness.

In Contribution F, we also introduce a differentiable renderer for a specific purpose. Our renderer works for rendering images of triangular meshes illuminated by a projector.

3.2 General methods

Having a differentiable renderer able to render any kind of scene seems like a wonderful concept, which should eliminate the need to create a differentiable renderer for a specific purpose, such as we have done multiple times in this thesis. However, all the methods presented in this section have drawbacks making them not applicable in our situations, but it is a rapidly advancing field. In this section we will focus on methods for triangular meshes.

Loper and Black [LB14] were the first to create a general framework for differentiable rendering. They hand-engineer the gradients, which they compute completely separately from the rendering itself. To compute them they segment the image into pixels that lie entirely on a face, and pixels on boundaries, and combine image space derivatives to approximate the true derivatives.

Liu et al. [Liu+17] extended this approach to support bidirectional reflectance distribution functions (BRDFs) and environment maps. Later Kato, Ushiku, and Harada [KUH18] created a simple textured renderer for use with neural networks. Common for these three approaches is that the rendering is done with rasterization, followed by approximating the gradients using operations in screen space.

Li et al. [Li+18] used ray tracing to render the images, and were able to compute exact gradients using reverse mode AD, but relies on sampling values along edges to account for visibility, which increases the number of samples necessary to get low variance in the gradients. This is improved in work by Loubet, Holzschuch, and Jakob [LHJ19], where they use a change of variable in the rendering equation to make it differentiable with respect to the scene parameters. However, all the methods using ray tracing with backpropagation are limited with a quickly growing memory footprint that limits their use to only simple scenes of lower resolution. This was partly addressed by Nimier-David et al. [Nim+20], who re-cast the backpropagation for rendering, such that the entire computation graph no longer needed to be stored. In this reformulated approach they do however not provide a means to compute partial derivatives of object positions, as these affect scene visibility. It can be noted that the general differentiable renderers that only compute approximations of the gradients are based on rasterization, while the ones with accurate gradients are based on ray tracing and that all of them use reverse mode AD.

3.3 Image-based rendering

In image-based rendering, images are used as the input to create a model of how to render similar images. This is a simpler way of creating photo-realistic images than traditional rendering, as the many details required for photo-realism can be drawn from other images.

Active appearance models (AAMs) [CET98] are an early example of this type of rendering, but still creates realistic enough images to directly compare them least-squares with real images. This is done by fitting a statistical model of the shape of texture of the object to a large database of images. This statistical model is fitted by doing a principal component analysis (PCA) on both the shape and intensity of the images, and using only the most important principal components, such that the model can be parameterized with a smaller number of parameters. In the paper they use this to create an AAM of human faces able to describe 98% of the variance of the original data using only 80 parameters. Interestingly, they use an approximation to find the gradients, by assuming that these are linear in the difference image. From a randomly sampled displacement in parameter space, the true difference image can be computed. By repeating this process many times, they use multivariate linear regression to be able to compute the gradient of the parameters from a difference image. Using this they can fit the AAM to new images.

Eisert [Eis02] uses an inverse rendering based approach for camera calibration with a calibration object with a known geometry. Knowing the poses of two frames, one can use the geometry of the calibration object to warp pixels from one frame to the other, which can be thought of as an optical flow where the constraint is that there should be constant brightness. He then does a first order Taylor expansion of the objective function and minimizes this. In addition to this he uses a multi-step approach where the images are smoothed a lot initially, with the amount of smoothing being reduced each time the problem is solved.

Least-squares matching (LSM) is a method for estimating the transform between a reference and a real image [Luh+13]. Multiple different types of transformations can be estimated such as the affine or projective transform, both of which are can be computed with bilinear interpolation. The objective is to minimize the squared difference between the reference image and the transformed target. To make the images match as well as possible, part of the transformation is also to estimate a linear transformation of the image intensities.

LSM is somewhat similar to our work in Contribution A, where we use an analytical function for the reference which gives us higher accuracy. Additionally, we use multiple views simultaneously and a protective camera model with lens distortion which is more advanced. Our work in Contribution C is also a type of image-based rendering, as we generate new frames by warping neighboring video frames.

3.4 Approximate renderings

As is visible from the previous sections, creating a differentiable rendering method is hard. Instead of attacking the problem by computing derivatives for regular rendering methods, other people have looked at reformulating the rendering operation itself, to make it differentiable.

Nguyen-Phuoc et al. [Ngu+18] did this in their work where they replaced the entire rendering pipeline by a deep neural network. Because all the operations in a regular neural network can be differentiated. As input for the network they use a voxel grid on which they do some 3D convolutions, followed by reshaping to 2D to do 2D convolutions until finally outputting an image with the correct number of channels. As they only use the neural network to render the images, the rendering automatically becomes differentiable. However, they were only able to train their network for a limited amount of shaders and having a voxel grid as input and a fixed output size makes it of limited use as a renderer.

Rhodin et al. [Rho+15] created a renderer where they render simple objects as spheres in a fully differentiable manner by replacing them with Gaussians where the visibility of a point is determined by the probability. They apply their method to pose estimation of humans with a multi-camera setup.

Work in the same vein was done by Palazzi et al. [Pal+18], in which they present a method for differentiable rendering of a silhouette. They do this by rendering the triangles of the mesh in a semi-transparent fashion, where pixels inside the triangle linearly become more opaque the further they are from the edge of a triangle.

A somewhat similar approach was used in the soft rasterizer introduced by Liu et al. [Liu+19]. Here they also use triangle meshes but replace the z-buffering and rasterization steps in a traditional rendering pipeline with differentiable versions thereof. Namely, they make the rasterization step soft by giving each triangle a probability of belonging to a pixel as a function of the signed distance to the edges of the triangle. They put this signed distance through a sigmoid function and use a function similar to softmax to do differentiable z-buffering. They then put these together, which yields a rendering where every pixel in the image has a contribution from every triangle, even those that would not be visible in a traditional rendering.

Overall, these approximate rendering approaches are different from what we have done in this thesis. However, the smoothing we introduce in Contribution A is similar to these approaches, as it in both cases is the key to the formulation being differentiable.

CHAPTER 4
Contributions

In this chapter, we describe the contributions that have been made during this Ph.D. This chapter is intended to serve as an overview of the contributions in the publications and how they fit in perspective to one another. We will not convey all details of each publication, but instead, mention the most important contributions in each work. The papers are available in their entirety as appendices to this thesis in Contributions A to F. In addition to this, we will discuss some work that did not make it into the articles and discuss possibilities for future work.

4.1 Homography-based rendering

We mentioned in Sections 1.2.1 and 2.4 that almost all previous methods for camera calibration rely on estimating points in 2D and minimizing the reprojection errors. As having an excellent camera calibration is essential for so many computer vision tasks and taking into account the difficulty of accurately locating corner points in images, and the point uncertainties not taken into account in current pipelines we decided to revisit the problem of camera calibration in Contribution A. As we mention in the introduction the point-based methods for camera calibration rely on detecting corner points in images and only uses the positions of the detected points to compute the calibration. The existing methods for localizing these points introduces some errors in the position, and these errors will propagate to the calibration. That the lines of the checkerboard also contain information away from the points is not exploited by these methods.

Our formulation of camera calibration minimizes the pixel-wise squared difference in intensity between the rendered and real images. To do this we introduce a novel differentiable rendering method for planes. This uses homographies as described in Section 2.6.1 on page 15 such that we can render the image by sampling a texture describing the checkerboard. To make the texture differentiable and to simulate blur, we convolve our texture function with a Gaussian kernel. Our rendering method is computationally efficient, even while computing gradients, which means that it is feasible to use it for optimization. We use forward mode AD for the gradients, as the number of parameters that we need the derivative with respect to in each pixel is relatively small. This enables us to compute the derivative of many outputs, which makes it cheap to compute the Jacobian and use Levenberg-Marquardt. Our

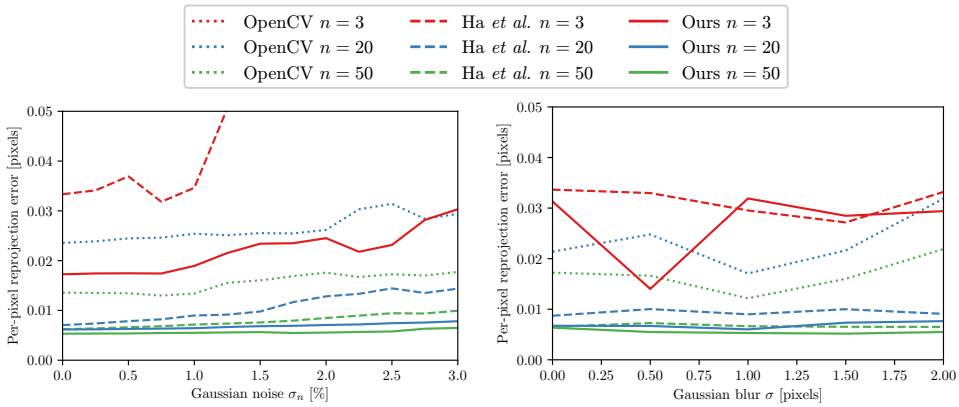


Figure 4.1: Comparing our method for camera calibration on synthetic data against Ha et al. [Ha+17] and OpenCV [FG87; Bra00] with varying number of images used to calibrate (n). Left: Varying σ_n with fixed $\sigma = 0.5$. Right: Varying σ with fixed $\sigma_n = 1\%$. OpenCV $n = 3$ lies above the plotted area. Figure from Contribution A.

contribution is thus a rendering-based pipeline for camera calibration where we only use the detected corner positions to obtain a starting point for the optimization.

To evaluate how close our estimated camera calibration was to the ground truth, we considered comparing each of the parameters to the ground truth value of the parameter, however, this is hard to do meaningfully especially for distortion parameters, as they can slightly counteract each other. Additionally, with a camera calibration, this will give a number for each parameter in the calibration, which will make it hard to compare two calibrations as different parameters can be closest to the true value in the two calibrations. This motivated us to introduce the *per-pixel reprojection error*, which enables us to compute a single number that measures how far an estimated camera calibration is from the ground truth camera. We do this by measuring the distance between where each pixel projects with the ground truth intrinsics compared to the estimated intrinsics.

First, we evaluated our method on synthetic data, as we know the ground truth camera parameters in this case. Here we performed better than the current point-based methods, even under the influence of Gaussian noise and blur, as one can expect to see in real images. The comparison can be seen in Figure 4.1. We also evaluate with real data, where we do not have access to the ground truth camera calibration. Therefore, we use the points detected by a good point-based method [Ha+17] to use instead of ground truth and detect the points using this method on a separate test set. For camera calibrations with few images ($n < 5$), our method performs better than the others. With more images, Ha et al. [Ha+17] get very similar reprojection errors

on the test set, and both of our methods are close to the lowest possible reprojection error, which we find by computing the reprojection error of the calibration for the training set as well.

In this work, we needed to invert the Brown-Conrady lens distortion model [Bro66; Con19] to know where to sample the rendering function. As mentioned in Section 2.3, this can be done iteratively in a naive manner. However, for this work, we used a more efficient method. With the notation from Section 2.3, computing the undistortion is equivalent to finding a root of

$$f(s) = sr^d d_r(sr^d) - r^d, \tag{4.1}$$

since when $sr_d = r$ then $f(s) = 0$. We can divide f by r^d and use Newton-Raphson to find s in an iterative fashion

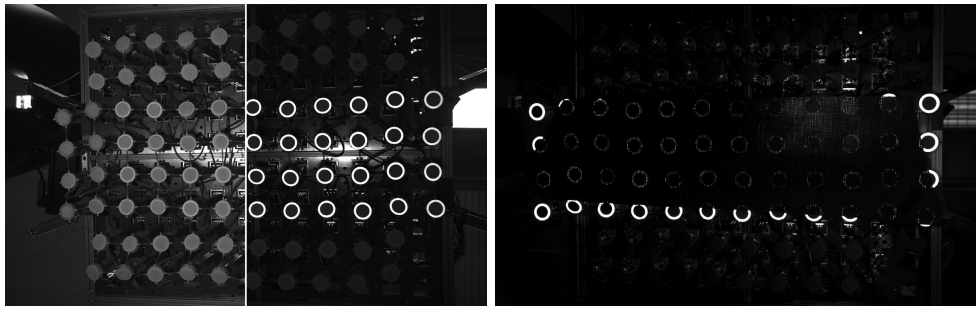
$$s_{k+1} = s_k - \frac{d_r(s_k r^d)s_k - 1}{\frac{d}{ds_k}\left(s_k d_r(s_k r^d)\right)}, \tag{4.2}$$

starting from $s_0 = 1$. This can be extended to tangential distortion as well by applying multivariate Newton-Raphson, but we did not use tangential distortion in this work. To the best of our knowledge, we are the first to apply this kind of optimization to the inversion of lens distortion. This converges in more cases than the method used by OpenCV [Bra00] and uses fewer operations to converge. We also apply the inverse function theorem to compute the exact gradients of the inverted lens distortion.

For the work we did in Contribution B, having an accurately calibrated camera was essential. In this contribution, we describe a robot setup for the difficult task of automating the production of carbon fiber parts. The project is focused on low volume production of low weight parts, which necessitates a flexible robot setup. This production is currently done by manual labour in Denmark, which makes it an obvious target for automation. The setup uses a grid of suction cups to grip a woven carbon fiber ply that has been pre-impregnated with epoxy which can drape the ply onto a mold.

In this process, we use three separate camera setups: a camera to find the position of the ply before picking it up, and two structured light setups with stereo cameras capable of producing 3D point clouds, one under the robot and another above the mold onto which the ply is draped.

The ply is stretchable in some directions, which it needs to be when draping it onto the curved mold, which is why the suction cups have been linked in a flexible grid that can adapt to the forces of the ply and the shape of the model. The flexibility causes the system to be underactuated, which means that there are more degrees of freedom in the system than there are motors controlling it. Because of these extra degrees of freedom, we cannot determine exactly where the suction cups are, but we need to know where the suction cups will make contact with the ply. To find the suction cups we use the 3D scanner underneath the robot, which gives us a 3D point



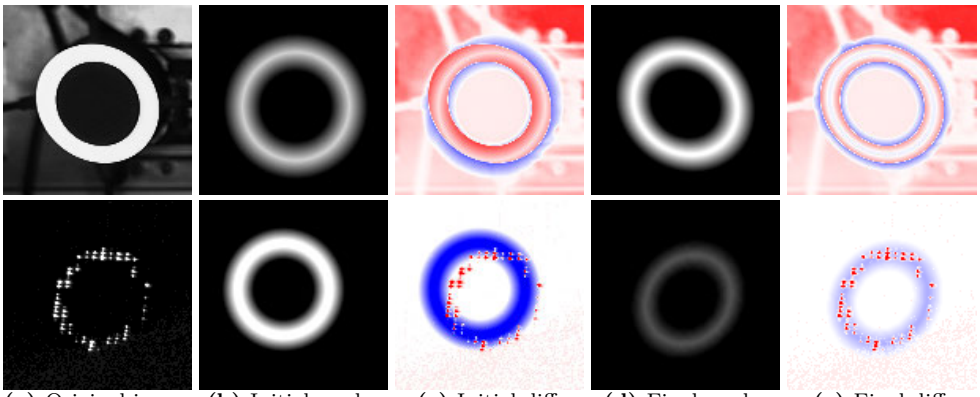
(a) Digital composite. Right: Lit by ring light. **(b)** Retro-reflectors visible through the ply.

Figure 4.2: Images from a camera in the stereo setup below the robot. Note the high visibility of the retro-reflectors when illuminated by the ring light of the camera.

cloud of the suction cups, which is not directly convertible to a pose. One of the parts I contributed to was in finding the pose of the suction cups from the point cloud. We did this by first segmenting the point cloud into quadrilaterals, each containing one suction cup, and fitting a planar surface within each segment with RANSAC [FB81]. Following this, we used a CAD model of the suction cup and aligned this with the point cloud with the iterative closest point (ICP) method [BM92] to obtain the pose of each suction cup.

We mentioned on the facing page that there were three camera setups involved. The 3D scanner above the mold is to measure the quality of the drape and the camera over the pickup table is to find the ply before picking it up. To estimate the position of rotation of the ply before picking it up, we create a homography using the four corners of the flat table during calibration. We find an initial estimate of the pose by thresholding the black ply on the white table. This threshold is hard to determine exactly, due to a smooth transition in image intensities between the table and therefore we refine the estimate of the pose of the ply further. We do this by rendering a smoothed version of the ply outline in a very similar fashion to our work in Contribution A and comparing it to the pixel intensities.

Following this work, we also devised another method to estimate the pose of the suction cups in the robot setup from Contribution B. For this we place a retro-reflective sticker on the perimeter of each suction cup and illuminate it with a ring light on each camera, to get clear images of the suction cups, such as those in Figure 4.2. Then we do inverse rendering to pose estimate the suction cups by applying a similar approach with homographies to render planes, but with a linear ramp function and having an optimizable parameter for the brightness of the rendering. We can estimate the pose in 3D because we have a calibrated stereo setup. Examples of the fitted images can be seen in Figure 4.3. This has the same advantages as our camera calibration of not



(a) Original image **(b)** Initial render **(c)** Initial diff. **(d)** Final render **(e)** Final diff.

Figure 4.3: Examples of fitting rings to images of retro-reflectors on suction cups. Top row: suction cup without ply, bottom row: suction cup obscured by ply. In the difference images, red indicates positive differences while blue indicates negative. All images in the bottom row have been multiplied by a factor of 20 for visualization. These examples have been chosen as they have initial parameters that are visibly off from the expected parameters.

needing to detect a discrete circle or compute points. The only point in time we detect a circle is to initialize our method, which we do with a Hough transform [DH72]. We compare our new method to the previous method in Table 4.1. We do see a bias of less than one mm in the z-direction. This is not caused by subsurface scattering or the thickness of the sticker as one could be inclined to believe, as both would have the effect of decreasing the number.

Because we no longer need to capture a full 3D scan of the suction cups, this approach has a much shorter acquisition time and does not require having a CAD model of the suction cups. Most importantly it is also able to work when the suction cups are currently gripping a ply, as there are small holes in the woven ply that let the retro-reflective light through, which can be used to improve the estimate of the robot state while the ply is gripped. This is important as more accurate knowledge of the robot state gives better control over the draping process. For suction cups covered by the ply, our method still works very robustly as long as we can initialize the circle such that it has some overlap with the correct circle.

Table 4.1: Evaluation of our method for pose estimation of suction cups that uses retro-reflectors. As we do not have access to ground truth poses, we report the difference between this method and the poses estimated by our ICP-based method from Contribution B. It can be seen from the differences that the poses estimated by both methods are very similar.

	Position [mm]			Rotation[°]
	x	y	z	
Mean difference	-0.05	0.06	0.76	2.61
Standard deviation	0.69	1.15	0.40	0.91

4.2 Optimization with triangle meshes

After the robot has draped the ply onto the mold, we need to inspect the quality of the drape. For this we have the structured light stereo camera mounted above the mold, which we use to scan the draped ply. Although we in Contribution B worked solely with point clouds, converting it to a surface such as a triangular mesh is more efficient for processing, storage, and visualization. Motivated by the drawbacks of most methods for mesh reconstruction mentioned in Section 1.2.3 and by the promising results we got from using inverse rendering in the other applications, we turned to applying inverse rendering to the problem of 3D reconstruction with a structured light 3D scanner setup, which we present in Contribution F.

The de facto standard method for 3D reconstruction with structured light is to unwrap the phases, match the points, and triangulate these to get a 3D point cloud as described in Section 2.7 on page 16. Following this, a mesh is fitted to the point cloud, which is usually done with a type of Poisson reconstruction [KBH06; KH13]. Each of the many steps in this process means that the final surface is not a least-squares fit to the original image data, as we also mention in the end of Section 2.7.

Our inverse rendering based approach represents the reconstructed surface as a triangle mesh and uses optimization to directly move the vertices in the mesh to their optimal positions, such that the difference between the real images and the renderings is minimized. To handle cases where the camera has lens distortion, we can use (4.2) to compute the undistorted pixel coordinates. This effectively fuses all steps of the 3D scanning into one end-to-end algorithm, which comes with many advantages, the biggest of these being that the surface is a least-squares fit to the original image intensities.

To initialize our optimization, we did our experiments starting from a simple sphere, and we demonstrate the same sphere working as initialization for three different objects. To make our optimization problem easier when our initial guess is as far off, as it is in this case, we optimize on the unwrapped phase ϕ_{uw}[1] from (2.32) on page 17. During

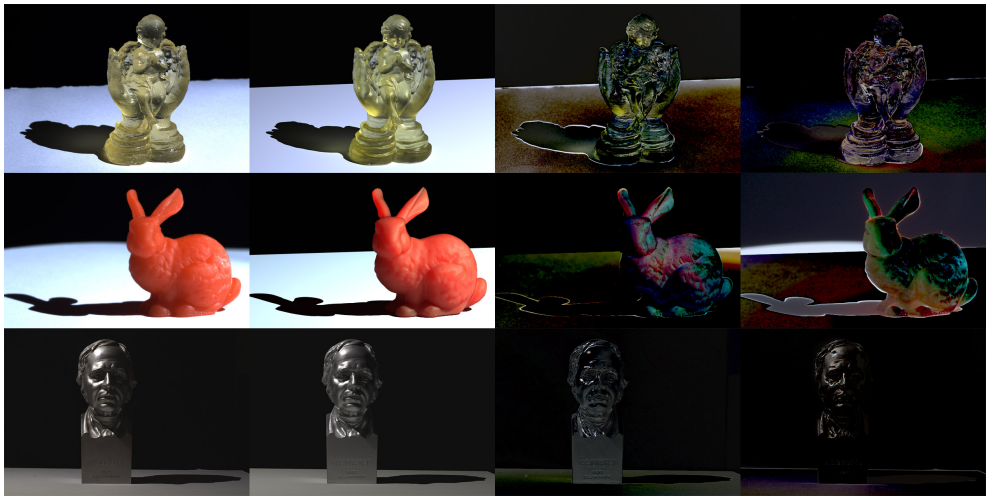
[1] In Contribution F we optimize on $X_c^{i,j}$, which is equal to $\phi_{uw}/(2\pi)$

Figure 4.4: Our method reconstructing the Stanford Bunny starting from a sphere. From left to right: Initial mesh (sphere), after 50 iterations, after 750 iterations, and the converged result. The final reconstruction has 13 780 vertices and has a symmetric volume difference of 0.30%, when compared to the ground truth. Figure from Contribution F.

the optimization where we optimize directly on the phase, we perform remeshing operations at regular intervals such that we are able to progressively increase the fidelity of the mesh. After a fixed number of iterations, we stop remeshing and change the objective function to optimize on the original phase shifted images $\{P_1, P_2\}$ in Section 2.7. An example of this process can be seen in Figure 4.4. We are of course also able to use another reconstruction method to obtain our initial guess.

The amount of reflected projector light in a pixel is a measure of how strong the measurement is. By reconstructing the mesh from all views at once, pixels are automatically weighed by the amount of reflected light from the projector, which is a proxy for signal strength. We compute all required gradients analytically on the GPU. We assume that each pixel in each camera has a fixed amount of background light and a fixed percentage of the light from the projector hitting the camera, which we estimate from the ground truth images.

With an accurate 3D shape and a well-calibrated camera, we have almost everything we need to be able to recreate a scene on the computer. We provide a method to do this in Contribution E. First, we provide a practical method for estimating the pose of a camera, a known object, and a point-like light source. Then, we give a step-by-step description of how to select an appearance model appropriate for the reflectance properties of the object. Our pose estimation method is based on finding the silhouette of the object in a way that is differentiable, i.e. without rasterization. We find it by projecting the edges of the mesh to the image plane of the camera and traversing them to extract the silhouette. We require that the input images have been segmented into object, shadow, and background, such that we can extract a ground truth silhouette of the object. The pose estimation is then done by minimizing the difference between the ground truth and rendered silhouettes. Our method also works for multiple camera positions and multiple light source positions. We demonstrate the usefulness of our method by creating renderings of a selection of scenes with different objects, see Figure 4.5. By comparing these renderings to the photographs, we can



photograph (x) rendering (y) $(\max(y-x,0))^{1/\gamma}$ $(\max(x-y,0))^{1/\gamma}$

Figure 4.5: Pixel-by-pixel comparison of renderings with a photograph enables a detailed investigation of the virtues and deficiencies of an appearance model. We have used our practical alignment technique from Contribution E to pose estimate the objects and light sources. Difference images were brightened with $\gamma = 2.2$ to visualize the deficiencies more clearly. Figure from Contribution E.

quantify the differences, and we use this tool in the process of selecting the rendering model. This quantitative comparison can also be used to identify the shortcomings of the current rendering methods and to see where the geometry differs.

The poses estimated by our method can also be used as an initial guess for an inverse rendering method, e.g. one of the methods in Section 3.2, to minimize the differences in pixel intensities between rendering and photograph even further. This is relevant because most inverse rendering methods need to be initialized relatively close to the correct solution.

When one has used this method to determine the most appropriate appearance model for a given object and found the appearance parameters that best describe it, we have captured the appearance of the object. The method can then be used to quantitatively to compare a manufactured part with this digital twin. This will take into account both the appearance and shape, because if just one of them has errors, this will be apparent in the difference image.

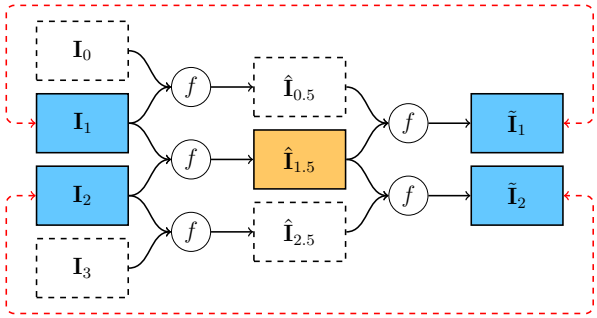


Figure 4.6: Diagram illustrating the cyclic fine-tuning process when predicting frame $\hat{\mathbf{I}}_{1.5}$. The model (f) is used to predict new frames, from frames the model has already predicted. In a perfect model, $\mathbf{I}_1$ and $\mathbf{I}_2$ would be identical to $\tilde{\mathbf{I}}_1$ and $\tilde{\mathbf{I}}_2$, their differences can be computed on the training data. Our cyclic fine-tuning therefore minimizes the differences illustrated by the dotted red lines, when we fine-tune the model to predict frame $\hat{\mathbf{I}}_{1.5}$. Figure from Contribution C.

4.3 Applications

In Contribution C we tackle the problem of video frame interpolation, i.e. generating additional frames in a video to increase the frame rate. This is a slightly different way of doing inverse rendering, in that the model that generates the synthetic images is replaced by a convolutional neural network (CNN), but we are still minimizing the differences of pixel values. Our model works by getting two images as input and outputting another one. As all the operations in our network design are differentiable, we can apply backpropagation and optimize the weights in our network to make it output in-between frames in a video. Our CNN uses the rectified linear unit (ReLU) activation function [GBC16] and a bilinear sampling function, both of which are differentiable almost everywhere, which lets us use backpropagation to train the model. One of our important contributions is the introduction of cyclic fine-tuning. This uses our cyclic loss, which uses frames predicted by the model to predict frames that we have ground truth for, see Figure 4.6. The quality of the reconstructed frame will be higher if the frames it is predicted from are also high. Because of this, we can use the cyclic loss as a proxy for the performance of the network on the current data, and therefore fine-tune the network on the testing data.

Our method was able to outperform all methods with publicly available implementations, and we have published our code as well.[2] We note that video frame interpolation is still a hot topic and since our paper, others have created models that perform better, notably the recent work by Niklaus and Liu [NL20].

[2] https://github.com/MortenHannemose/pytorch-vfi-cft

We chose to use a CNN for this task because this is a problem where formulating a model to generate the correct output is bordering on the impossible for humans, but the neural network can learn the correct function from large amounts of data. We have abundant amounts of data available, as videos can be readily obtained from the internet, and every second frame withheld to generate the ground truth. This means that we do not have the problem of lacking training data, which is so often the case when using deep learning methods. In our setting, the optimizer is used to find the weights of the model, instead of finding the scene parameters directly, and we even get a model that works on a wide variety of scenes if we have the variety in our training data. In this way we are also able to tackle harder problems in inverse rendering, for example, if 3D scanning and modeling the appearance of an object is too challenging, we can use CNNs to provide us with a plausible prediction of the output, even in these hard cases.

Finally, in Contribution D, we present a proof of concept for generating spatial attention cues that can be used to guide a user's visual attention in an augmented reality application. Doing this requires both accurate camera calibration and 3D scanning. Augmented reality is a topic that has been seeing increased interest in recent years, and the ability to guide a user's attention in a non-obtrusive way is essential if we are to make augmented reality applications easier to interact with, without alienating the users of the application.

Our concept uses a stereo structured light setup to do 3D scanning and then uses the projector to project a series of images that create the illusion of motion in the static scene. To accomplish this we start with doing a camera calibration of both cameras and the projector, followed by using this to do two separate 3D scans using each camera together with the projector, to also reconstruct points only visible in one camera. We use the two computed point clouds to render an image of the scene from the point of view of the projector. This enables us to apply filters to this image and project the resulting images back onto the scene, which creates a convincing appearance of continuous motion on selected parts of the scene. Videos of the effect are available online.[3]

For this project, we also went through the process of writing the software for and building our own structured light scanner, which let us have direct access to the projector and cameras. We needed to do this in order to have direct access to the same projector that did the scans because we needed to project the motion generating patterns from exactly the same point of view.

4.4 Discussion

From our work on camera calibration in Contribution A we also got additional insights into how the accuracy of a calibration artifact influences the calibration. In our

[3] http://people.compute.dtu.dk/jnje/illusory-motion

synthetic data, we used a texture with a finite resolution mapped to a plane. Here we observed that increasing the resolution of the checkerboard texture we used yielded a lower *per-pixel reprojection error*, even for very high-resolution textures (we ended up using a 20 000 × 15 000 texture). This indicates that even small imperfections in the calibration artifact can affect the accuracy of the calibration. This could also be why we are not seeing better performance for all the real data, as a real checkerboard is only as accurate as the manufacturing process used to produce it. These observations also make sense when we consider that our method has the assumption that the checkerboard is perfect.

Therefore, an interesting avenue for future work would be to expand the method, to allow for slight imperfections in the checkerboard. This could be done by doing something similar to bundle adjustment that some camera calibrations do. In this step, each point on the checkerboard is assumed to be unknown and is also found using optimization, which lets the algorithm account for checkerboards that are not completely flat or with not exactly square checkers. One approach to implement this bundle adjustment step into our rendering process would be by having each point be part of the optimization and letting the plane for each corner be defined by the best fitting plane to the neighboring points of the corner. This would enable us to reuse the homography-based rendering while letting us describe non-square checkers and non-planar checkerboards.

Additionally, it should be mentioned that we did not use this method in the subsequent contributions, as our method was only designed to render normal checkerboards. However, in the calibrations we needed to do, we used a ChArUco board, which is a checkerboard with ArUco markers on the checkers [Gar+14]. This makes it possible to detect a partially visible checkerboard, which in turn makes it easier to capture calibration images with features close to the image edges. We did this because the distortion parameters found by a calibration are only a good description for the part of the image plane that contains features. In future work, it would thus also be relevant to expand our method to handle other types of checkerboards.

In Contribution D, we spent a significant amount of time on getting the setup for generating images taken from the location of the projector to work. We did consider using a beam splitter instead of generating it by projecting the 3D point clouds, which would have enabled us to project and capture images using the same lens. However, this would have made the hardware setup more complicated. Additionally, having 3D information about the surface gave us the possibility of pursuing depth-aware versions of the illusion. One could imagine compensating for the light falloff from the projector using the inverse square law and to make the effect more viewpoint invariant by using the normals of the surface, and possibly incorporating multiple projectors.

CHAPTER 5

Conclusion

In this thesis, we have introduced and discussed the research that has been carried out during the past three years of Ph.D. studies. Through our research efforts, we have produced several scientific publications, wherein we provide solutions to several problems we identified, that have applications within a wide range of fields, such as industrial automation, quality control, and digital twins.

The research undertaken during this project has followed a largely exploratory approach, guided by the principle of tackling relevant problems we identified along the way. Our research has led us to explore both the mathematical aspects of finding and implementing derivatives by hand and through the more practical work in building and programming our own 3D structured light scanner. Thus, the path we have taken was not determined in advance but has materialized as a consequence of the findings we did along the way. We are therefore pleased to see that there is a common thread running through the contributions made in this thesis.

Our journey started with being part of a bigger project in automation, where we were faced with the challenges of integrating computer vision in a robot system, which we describe our solution to in Contribution B. One of these challenges was estimating the pose of the suction cups in the robotic setup, which we ended up trying two different methods for, with our second method being based on inverse rendering. Based on the efficiency and simplicity of this homography-based rendering method, we saw an opportunity to use the method to create an improved camera calibration method, which we did in Contribution A. This required us to modify the method to work for whole frames and finding derivatives of intrinsics and extrinsics.

During our work with differentiable rendering for camera calibration, we were inspired to also apply it to the problem of 3D reconstruction for structured light, which resulted in Contribution F. In this work we contribute with a method that can take images of structured light phase shifting and from these directly reconstruct a mesh representation of the object in the images.

Following this, we were inspired by the difficulty of digitally recreating even a simple photograph. This led us to do the work presented in Contribution E, where our contribution is a practical method for pose estimation of an object with known geometry and a point-like light source and a guideline on how to iteratively refine the appearance model of the object. This can be used to render the same scene on the

computer to compare to the photograph, which can among other things be used to identify deficiencies in the appearance model.

Between these, we were inspired by the many recent advances in computer vision that have been driven by deep learning and set our sights on a different application of inverse rendering, namely frame interpolation of video sequences, which we did in Contribution C.

Finally, in Contribution D we contribute with a method for being able to guide the attention of a user in augmented reality, for which we used a calibrated stereo camera-projector setup and 3D scanning of the scene.

The highlights of our contributions can be summarized as follows. We have:

- Created and evaluated a method for accurate camera calibration and evaluated the performance on real and synthetic data in Contribution A.
- Devised a formula for comparing a camera calibration to ground truth in Contribution A.
- Provided a simple method for inversion of the Brown-Conrady lens distortion model.
- Presented a practical method for estimating the pose of an object and a point-like light source in Contribution E.
- Presented a framework for quantitative comparison of a rendering and a photograph and described how to refine the appearance model to match the photograph better in Contribution E.
- Developed a method for reconstructing a mesh directly from structured light images in Contribution F.
- Presented an improved method for frame interpolation in video sequences using convolutional neural networks (CNNs) in Contribution C.
- Developed demonstrators in robotics and augmented reality that could benefit from our methods (Contributions B and D).
- Derived an elegant method for inverse rendering of planes in Contributions A and B.

With these contributions we have, as we originally intended, developed practical methods for common problems in computer vision that are more accurate or more widely applicable than previous methods.

Bibliography

[BM92] Paul J. Besl and Neil D. McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Volume 1611. International Society for Optics and Photonics. 1992, pages 586–606.

[Bra00] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[Bro66] Duane C. Brown. "Decentering distortion of lenses". In: *Photogrammetric Engineering and Remote Sensing* (1966).

[CET98] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. "Active appearance models". In: *European conference on computer vision*. Springer. 1998, pages 484–498.

[Con19] Alexander Eugen Conrady. "Decentred lens-systems". In: *Monthly notices of the royal astronomical society* 79.5 (1919), pages 384–390.

[DH72] Richard O. Duda and Peter E. Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (1972), pages 11–15.

[DL16] Pierre Drap and Julien Lefèvre. "An exact formula for calculating inverse radial lens distortions". In: *Sensors* 16.6 (2016), page 807.

[Eis02] Peter Eisert. "Model-Based Camera Calibration Using Analysis by Synthesis Techniques". In: *VMV*. 2002, pages 307–314.

[Ell+20] L.-P. Ellekilde, J. Wilm, O.W. Nielsen, C. Krogh, E. Kristiansen, G.G. Gunnarsson, T.S. Stenvang, J. Jakobsen, M. Kristiansen, J.A. Glud, M. Hannemose, H. Aanæs, J. de Kruijk, I. Sveidahl, A. Ikram, and H.G. Petersen. "Design of Automated Robotic System for Draping Prepreg Composite Fabrics". In: *Robotica* (2020), pages 1–16. DOI: 10.1017/S0263574720000193.

[FB81] Martin A. Fischler and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pages 381–395.

[FG87] Wolfgang Förstner and Eberhard Gülch. "A fast operator for detection and precise location of distinct points, corners and centres of circular features". In: *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*. Interlaken. 1987, pages 281–305.

[FL95] Pascal Fua and Yvan G. Leclerc. "Object-centered surface reconstruction: Combining multi-image stereo and shading". In: *International Journal of Computer Vision* 16.1 (1995), pages 35–56.

[Gar+14] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pages 2280–2292.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Gki+13] Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. "Inverse volume rendering with material dictionaries". In: *ACM Transactions on Graphics (TOG)* 32.6 (2013), pages 1–13.

[Gre81] Ulf Grenander. *Lectures in pattern theory I, II and III: Pattern Synthesis, Pattern Analysis and Regular Structures*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, 1976-1981.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[Ha+17] Hyowon Ha, Michal Perdoch, Hatem Alismail, In So Kweon, and Yaser Sheikh. "Deltille grids for geometric camera calibration". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pages 5344–5352.

[Han+17] Morten Hannemose, Jannik Boll Nielsen, László Zsíros, and Henrik Aanæs. "An image-based method for objectively assessing injection moulded plastic quality". In: *Scandinavian Conference on Image Analysis*. Springer. 2017, pages 426–437. DOI: 10.1007/978-3-319-59129-2_36.

[Han+19] Morten Hannemose, Janus Nørtoft Jensen, Gudmundur Einarsson, Jakob Wilm, Anders Bjorholm Dahl, and Jeppe Revall Frisvad. "Video frame interpolation via cyclic fine-tuning and asymmetric reverse flow". In: *Scandinavian Conference on Image Analysis*. Springer. 2019, pages 311–323. DOI: 10.1007/978-3-030-20205-7_26.

[Han+20] Morten Hannemose, Mads Emil Brix Doest, Andrea Luongo, Søren Kimmer Schou Gregersen, Jakob Wilm, and Jeppe Revall Frisvad. "Alignment of rendered images with photographs for testing appearance models". In: *Applied Optics* (2020).

[HO18] Evan G. Hemingway and Oliver M. O'Reilly. "Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments". In: *Multibody System Dynamics* 44.1 (2018), pages 31–56.

[Hof16] Philipp H.W. Hoffmann. "A hitchhiker's guide to automatic differentiation". In: *Numerical Algorithms* 72.3 (2016), pages 775–811.

[Hor77] Berthold K. P. Horn. "Understanding image intensities". In: *Artificial intelligence* 8.2 (1977), pages 201–231.

[HWF19] Morten Hannemose, Jakob Wilm, and Jeppe Revall Frisvad. "Superaccurate camera calibration via inverse rendering". In: *Modeling Aspects in Optical Metrology VII*. Volume 11057. International Society for Optics and Photonics. SPIE, 2019, pages 252–260. DOI: 10.1117/12.2531769.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[Jen+19] Janus Nørtoft Jensen*, Morten Hannemose*, Jakob Wilm, Anders Bjorholm Dahl, Jeppe Revall Frisvad, and Serge Belongie. "Generating spatial attention cues via illusory motion". In: *Third Workshop on Computer Vision for AR/VR*. 2019. URL: http://people.compute.dtu.dk/jnje/illusory-motion/.

[Jen+20] Janus Nørtoft Jensen*, Morten Hannemose*, J. Andreas Bærentzen, Jakob Wilm, Jeppe Revall Frisvad, and Anders Bjorholm Dahl. "Surface Reconstruction from Structured Light Images using Differentiable Rendering". In: *Peer Reviewed Journal (To be decided)* (2020).

[Kat+20] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. "Differentiable Rendering: A Survey". In: *arXiv preprint arXiv:2006.12057* (2020).

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson surface reconstruction". In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Volume 7. 2006.

[KH13] Michael Kazhdan and Hugues Hoppe. "Screened poisson surface reconstruction". In: *ACM Transactions on Graphics (ToG)* 32.3 (2013), pages 1–13.

[KPC93] John K. Kawai, James S. Painter, and Michael F. Cohen. "Radioptimization: goal based rendering". In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 1993, pages 147–154.

[KUH18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3d mesh renderer". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pages 3907–3916.

[LB14] Matthew M. Loper and Michael J. Black. "OpenDR: An approximate differentiable renderer". In: *European Conference on Computer Vision*. Springer. 2014, pages 154–169.

[Lev44] Kenneth Levenberg. “A method for the solution of certain non-linear problems in least squares”. In: *Quarterly of Applied Mathematics* 2.2 (1944), pages 164–168.

[LHJ19] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. “Reparameterizing discontinuous integrands for differentiable rendering”. In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pages 1–14.

[Li+18] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. “Differentiable Monte Carlo Ray Tracing through Edge Sampling”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 37.6 (2018), 222:1–222:11.

[Liu+17] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. “Material editing using a physically based rendering network”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pages 2261–2269.

[Liu+19] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. “Soft rasterizer: A differentiable renderer for image-based 3d reasoning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pages 7708–7717.

[Low04] David G. Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pages 91–110.

[Luh+13] Thomas Luhmann, Stuart Robson, Stephen Kyle, and Jan Boehm. *Close-range photogrammetry and 3D imaging*. Walter de Gruyter, 2013.

[Mar63] Donald W. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters”. In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pages 431–441.

[Mar98] Stephen Robert Marschner. “Inverse rendering for computer graphics”. PhD thesis. Cornell University, 1998.

[Max+19] Maxim Maximov, Laura Leal-Taixé, Mario Fritz, and Tobias Ritschel. “Deep appearance maps”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pages 8729–8738.

[Ngu+18] Thu H. Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. “Rendernet: A deep convolutional network for differentiable rendering from 3d shapes”. In: *Advances in Neural Information Processing Systems*. 2018, pages 7891–7901.

[Nim+20] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. “Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pages 146–1.

[NL20] Simon Niklaus and Feng Liu. “Softmax Splatting for Video Frame Interpolation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pages 5437–5446.

[NM65] John A. Nelder and Roger Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pages 308–313.

[NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[Pal+18] Andrea Palazzi, Luca Bergamini, Simone Calderara, and Rita Cucchiara. “End-to-end 6-DoF Object Pose Estimation through Differentiable Rasterization”. In: *Second Workshop on 3D Reconstruction Meets Semantics (3DRMS)*. 2018.

[Pla+14] Simon Placht, Peter Fürsattel, Etienne Assoumou Mengue, Hannes Hofmann, Christian Schaller, Michael Balda, and Elli Angelopoulou. “Rochade: Robust checkerboard advanced detection for camera calibration”. In: *European Conference on Computer Vision*. Springer. 2014, pages 766–779.

[Pus96] Gordon D. Pusch. “Jet space as the geometric arena of automatic differentiation”. In: *Computational Differentiation: Techniques, Applications, and Tools, M. Berz, CH Bischof, GF Corliss, and A. Griewank, eds., SIAM, Philadelphia, Penn* (1996), pages 53–62.

[Reh+17] Joern Rehder, Janosch Nikolic, Thomas Schneider, and Roland Siegwart. “A direct formulation for camera calibration”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pages 6479–6486.

[Rho+15] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. “A versatile scene model with differentiable visibility applied to generative pose estimation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pages 765–773.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pages 533–536.

[RRT97] Carsten Reich, Reinhold Ritter, and Jan Thesing. “White light heterodyne principle for 3D-measurement”. In: *Sensors, Sensor Systems, and Sensor Data Processing*. Volume 3100. International Society for Optics and Photonics. 1997, pages 236–244.

[RS17] Joern Rehder and Roland Siegwart. “Camera/IMU calibration revisited”. In: *IEEE Sensors Journal* 17.11 (2017), pages 3257–3268.

[RW97] Alyn P. Rockwood and Jim Winget. “Three-dimensional object reconstruction from two-dimensional images”. In: *Computer-Aided Design* 29.4 (1997), pages 279–285.

[Sch+20] 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*. 2020, pages 2535–2544.

[Sho85] Ken Shoemake. “Animating rotation with quaternion curves”. In: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. 1985, pages 245–254.

[Sme+02] Vadim N. Smelyansky, Robin D. Morris, Frank O. Kuehnel, David A. Maluf, and Peter Cheeseman. “Dramatic improvements to feature based stereo”. In: *European Conference on Computer Vision*. Springer. 2002, pages 247–261.

[Tri+99] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. “Bundle adjustment—a modern synthesis”. In: *International workshop on vision algorithms*. Springer. 1999, pages 298–372.

[Vel+18] Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. “Appearance capture and modeling of human teeth”. In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pages 1–13.

[Zha00] Zhengyou Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pages 1330–1334.

CONTRIBUTION A

Superaccurate Camera Calibration via Inverse Rendering

Superaccurate Camera Calibration via Inverse Rendering

Morten Hannemose[a], Jakob Wilm[b], and Jeppe Revall Frisvad[a]

[a]DTU Compute, Technical University of Denmark
[b]SDU Robotics, University of Southern Denmark

ABSTRACT

The most prevalent routine for camera calibration is based on the detection of well-defined feature points on a purpose-made calibration artifact. These could be checkerboard saddle points, circles, rings or triangles, often printed on a planar structure. The feature points are first detected and then used in a nonlinear optimization to estimate the internal camera parameters. We propose a new method for camera calibration using the principle of inverse rendering. Instead of relying solely on detected feature points, we use an estimate of the internal parameters and the pose of the calibration object to implicitly render a non-photorealistic equivalent of the optical features. This enables us to compute pixel-wise differences in the image domain without interpolation artifacts. We can then improve our estimate of the internal parameters by minimizing pixel-wise least-squares differences. In this way, our model optimizes a meaningful metric in the image space assuming normally distributed noise characteristic for camera sensors. We demonstrate using synthetic and real camera images that our method improves the accuracy of estimated camera parameters as compared with current state-of-the-art calibration routines. Our method also estimates these parameters more robustly in the presence of noise and in situations where the number of calibration images is limited.

Keywords: camera calibration, inverse rendering, camera intrinsics

1. INTRODUCTION

Accurate camera calibration is essential for the success of many optical metrology techniques such as pose estimation, white light scanning, depth from defocus, passive and photometric stereo, and more. To obtain sub-pixel accuracy, it can be necessary to use high-order lens distortion models, but this necessitates a large number of observations to properly constrain the model and avoid local minima during optimization.

A very commonly used camera calibration routine is that of Zhang.[1] This is based on detection of feature points, an approximate analytic solution and a nonlinear optimization of the reprojection error to estimate the internal parameters, including lens distortion. Oftentimes, checkerboard corners are detected using Harris' corner detector,[2] followed by sub-pixel saddle-point detection, such as that of Förstner and Gülch,[3] which is implemented in OpenCV's `cornerSubPix()` routine. This standard technique can be improved for example by more robust and precise sub-pixel corner detectors[4,5] or use of a pattern different from the prevalent checkerboard.[6,7] A different line of work aims at reducing perspective and lens-dependent bias of sub-pixel estimates.[8,9] In the work of Datta,[10] reprojection errors are reduced significantly by iteratively rectifying images to a frontoparallel view and re-estimating saddle points. Nevertheless, such techniques are still dependent on how accurately and unbiased the corners/features were detected in the first place. Perspective and lens-distortion are then not considered directly, as their parameters are known only after calibration. Instead, the common approach is to try to make the detector mostly invariant to such effects. However, for larger features such as circles, it is questionable whether these can be detected in an unbiased way without prior knowledge of lens parameters. In addition, the distribution of the localization error is unknown and least-squares optimization may not be optimal.

In this paper, instead of relying solely on the sub-pixel accuracy of points in the image, we render an image of the calibration object given the current estimate of calibration parameters and the pose of the object. This non-photorealistic rendering of the texture of the calibration object can be compared to the observed image, which lets us compute pixel-wise differences in the image domain without interpolation. Because we are comparing

Further author information: (Send correspondence to M.H.)
M.H. E-mail: mohan@dtu.dk

Morten Hannemose, Jakob Wilm, and Jeppe Revall Frisvad. "Superaccurate camera calibration via inverse rendering ," Proc. SPIE, Modeling Aspects in Optical Metrology VII, 1105717, (2019). https://doi.org/10.1117/12.2531769

differences in pixel intensities, we can model the errors as normally distributed which closely resembles the noise characteristics usually seen in camera images. This process is iterated in an optimization routine so that we are able to directly minimize the squared difference between the observed pixels and our rendered equivalent.

To ensure convergence of the optimization, the error must be differentiable with respect to camera parameters, object pose, and image coordinates. We ensure this by rendering slightly smoothed versions of the calibration object features.

2. RELATED WORK

We use a texture for our implicit rendering. This bears some resemblance to the version of texture-based camera calibration[11] where a known pattern is employed. We thus inherit some of the robustness and accuracy benefits that this method earns because it is not relying exclusively on feature extraction. Our optimization strategy is however simpler and more easily applied in practice as compared with their rank minimization problem with nonlinear constraints.

The work by Rehder *et al.*[12] is more closely related to ours. They argue that an initial selection of feature points (like corners) is an inadequate abstraction. As in our work, they use a standard calibration technique for initialization. With this calibration, they implicitly render the calibration target into selected pixels to get a more direct error formulation based on image intensities. This is then used to further refine different calibration parameters through optimization. Their approach results in little difference from the initial calibration values in terms of intrinsic parameters. Instead, they focus on the use of their technique for estimating line delay in rolling shutter cameras and for inferring exposure time from motion blur. Rehder *et al.* select pixels for rendering where they find large image gradients in the calibration image. Our pixel selection scheme is different from theirs: we use all the pixels that the target is projected to, and our objective function is different.

In more recent work, Rehder and Siegwart[13] extend their direct formulation of camera calibration[12] to include calibration of inertial measurement units (IMUs). In this work, the authors introduce blurring into their renderings to simulate imperfect focusing and motion blur. We also use blurring, and their objective function is more similar to ours in this work. However, they still only select a subset of pixels for rendering based on image gradients, and they, again, did well in estimating exposure time from motion blur but did not otherwise improve results over the baseline approach.

In terms of improved image features, Ha *et al.*[7] proposed replacing the traditional checkerboard with a triangular tiling of the plane (a deltille grid). They describe a method for detecting this pattern and checkerboards in an image and introduce a method for computing the sub-pixel location of corner points for deltille grids or checkerboards. This is based on resampling of pixel intensities around a saddle point and fitting a polynomial surface to these. We consider this approach state-of-the-art in camera calibration based on detection of interest points, and we therefore use it for performance comparison.

3. METHOD

Our method builds on top of an existing camera calibration method. This is used as a starting guess for the camera matrix $\mathbf{K}_0$, the distortion coefficients $\mathbf{d}_0$ and the poses of each calibration object $\mathbf{R}_{i0}$, $\mathbf{t}_{i0}$. We use this to render images of calibration objects, which we compare with images captured by the camera. Based on this comparison, the optimizer updates the camera calibration until the result is satisfactory. An outline of our method is in Figure 1.

The rendering is based on sampling a smooth function $\mathrm{T_G}$ that describes the texture of the calibration object. The initial calibration with a standard technique reveals which pixels the calibration target covers. Each of these pixels is undistorted and projected onto the surface of the calibration object to get the coordinates for where to sample $\mathrm{T_G}$. This inverse mapping from pixel coordinates to calibration object coordinates is advantageous with respect to calculating the gradient used in optimization. Each sampled $\mathrm{T_G}$ value is compared with the intensity of its corresponding pixel, and the sum of squared errors is the objective function that we minimize.

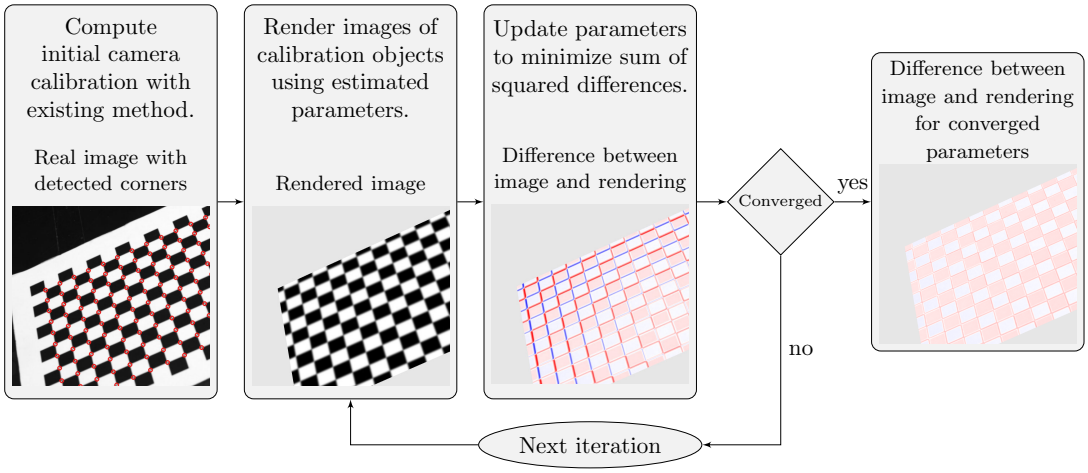


Figure 1: Overview of our method with a checkerboard as an example. All images are crops of a larger image. Difference images: Red represents positive values and blue represents negative values. For clear visualization, we have multiplied the focal length by 1.01 in our initial guess. Note that the converged difference image still resembles a checkerboard pattern because we do not compensate for the board's albedo.

3.1 Projection of points

Let us introduce a function that projects points in $\mathbb{R}^3$ to the image plane of a camera, including distortion

$$\mathrm{P}\left(\mathbf{K}, \mathbf{d}, \mathbf{p}\right) = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{1}$$

Here, $\mathbf{K}$ is a camera matrix, $\mathbf{d}$ is a vector of distortion coefficients, and $\mathbf{p}$ is the point we are projecting, where the elements are

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{2}$$

P is then implemented as follows. First, the points are projected to normalized image coordinates:

$$\begin{bmatrix} x_\mathrm{n} \\ y_\mathrm{n} \end{bmatrix} = \frac{1}{p_z} \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \quad r^2 = x_\mathrm{n}^2 + y_\mathrm{n}^2. \tag{3}$$

The normalized points are distorted using the distortion model of Brown-Conrady.[14, 15]

$$\begin{bmatrix} x_\mathrm{nd} \\ y_\mathrm{nd} \end{bmatrix} = \begin{bmatrix} x_\mathrm{n} \\ y_\mathrm{n} \end{bmatrix} \left(1 + k_1 r^2 + k_2 r^4\right) + \begin{bmatrix} 2p_1 x_\mathrm{n} y_\mathrm{n} + p_2 \left(r^2 + 2x_\mathrm{n}^2\right) \\ 2p_2 x_\mathrm{n} y_\mathrm{n} + p_1 \left(r^2 + 2y_\mathrm{n}^2\right) \end{bmatrix}, \tag{4}$$

and the distorted points are converted to pixel coordinates

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_x x_\mathrm{nd} + x_0 \\ f_y y_\mathrm{nd} + y_0 \end{bmatrix}. \tag{5}$$

3.2 Rendering

Let each calibration board have its own u, v coordinate system, and let $\mathbf{R}_i$ and $\mathbf{t}_i$ describe the pose of the ith board. Let $\mathbf{r}_{i_j}$ denote the ith column of $\mathbf{R}_i$. The 3D position of a point on a board is then

$$\mathbf{p}(i,u,v) = \begin{bmatrix}\mathbf{R}_i & \mathbf{t}_i\end{bmatrix}\begin{bmatrix}u\\v\\0\\1\end{bmatrix} = \begin{bmatrix}\mathbf{r}_{i_1} & \mathbf{r}_{i_2} & \mathbf{t}_i\end{bmatrix}\begin{bmatrix}u\\v\\1\end{bmatrix}. \tag{6}$$

Using a camera matrix $\mathbf{K}$ and distortion coefficients $\mathbf{d}$, we can project this point to the image plane

$$\begin{bmatrix}x\\y\end{bmatrix} = \mathrm{P}(\mathbf{K},\mathbf{d},\mathbf{p}(i,u,v)). \tag{7}$$

We can solve for u, v in terms of x, y in the above expression and obtain a new function:

$$\begin{bmatrix}u\\v\end{bmatrix} = \mathrm{P}^{-1}(\mathbf{K},\mathbf{d},i,x,y). \tag{8}$$

As mentioned, we use a function to define the texture of the calibration board (checkerboard, circles, deltille grid, or likewise). Let us call this texture function $\mathrm{T}(\mathbf{p}_{uv})$. Because natural images are not always sharp, we introduce a blurry version of the texture map by convolving it with a Gaussian kernel in u and v. This has the additional advantage that the texture map becomes smooth, which makes the objective function differentiable.

$$\mathrm{T_G}(\mathbf{p}_{uv},\sigma_u,\sigma_v) = (G_{\sigma_u} * G_{\sigma_v} * \mathrm{T})(\mathbf{p}_{uv}). \tag{9}$$

This blur is applied in texture space, but we actually want it to be uniform around the interest point in image space. Thus, the standard deviations σ_u and σ_v, need to be corrected according to the length of the positional differential vector of the projection to the image plane. We introduce this quantity as

$$M_u = \left|\left|\frac{\partial \mathrm{P}(\mathbf{K},\mathbf{d},\mathbf{p}(i,u,v))}{\partial u}\right|\right|_2. \tag{10}$$

Inserting Equations (8) and (10) in Equation (9), we obtain a function that enables the rendering of an image of the calibration object:

$$\mathrm{C}_i(x,y,\sigma_{i,j}) = \mathrm{T_G}\left(\mathrm{P}^{-1}\left(\mathbf{K},\mathbf{d},i,x,y\right),\sigma_{i,j}/M_u,\sigma_{i,j}/M_v\right), \tag{11}$$

where M_v is the same as M_u but with respect to v and $\sigma_{i,j}$ is a measure of how blurry the image is around the jth interest point on the ith calibration board. This implies that the formula is only valid in the neighborhood of this point, and therefore we introduce

$$\mathcal{N}_{i,j} \tag{12}$$

to describe the set of pixel coordinates where the rendering is accurate. We choose $\mathcal{N}_{i,j}$ to be the pixels where the corresponding u, v coordinate is no further away than one half of the interest point spacing in Manhattan distance given by the initial camera calibration. For convenience of notation, let us define a set containing all $\mathbf{R}_i$, $\mathbf{t}_i$, and $\sigma_{i,j}$

$$\beta = \left\{\mathbf{R}_i,\mathbf{t}_i : i \in \{1,\ldots,n_i\}\right\} \cup \left\{\sigma_{i,j} : i \in \{1,\ldots,n_i\}, j \in \{1,\ldots,n_j\}\right\}, \tag{13}$$

where n_i is the number of calibration boards and n_j is the number of interest points on each calibration board. Using Equations (11) to (13), our optimization problem is then

$$\hat{\mathbf{K}},\hat{\mathbf{d}},\hat{\beta} = \underset{\mathbf{K},\mathbf{d},\beta}{\arg\min} \sum_{i=1}^{n_i}\sum_{j=1}^{n_j}\sum_{x,y\in\mathcal{N}_{i,j}} \Big(\mathrm{C}_i(x,y,\sigma_{i,j}) - I_i(x,y)\Big)^2, \tag{14}$$

where $I_i(x,y)$ is the intensity of the pixel at x, y in the image containing the ith calibration board. We parameterize $\mathbf{R}_i$ as quaternions and solve Equation (14) using the Levenberg-Marquardt algorithm.[16,17] Because Equation (9) is defined to give values between 0 and 1, in the case where $\sigma = 0$, our optimization problem is equivalent to maximizing the sum of pixels on white parts of T while minimizing the sum of pixels corresponding to black parts of T.

3.3 Computation of P^{-1}

Recall that P^{-1} is the function that, given a camera calibration and the pose of a calibration board, transforms from x, y in pixel space to u, v coordinates on the board. The first step in computing this is to invert Equation (5) by normalizing the pixel coordinates

$$\begin{bmatrix} x_{\mathrm{nd}} \\ y_{\mathrm{nd}} \end{bmatrix} = \begin{bmatrix} \frac{x - x_0}{f_x} \\ \frac{y - y_0}{f_y} \end{bmatrix}. \tag{15}$$

Inverting Equation (4) is not possible to do analytically, so we use an iterative numerical approach.[18] Note however that we can compute analytical derivatives of the inverse of Equation (4) by applying the inverse function theorem. To map the undistorted normalized coordinates to the calibration object, we combine Equations (3) and (6):

$$s \begin{bmatrix} x_{\mathrm{n}} \\ y_{\mathrm{n}} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{r}_{i_1} & \mathbf{r}_{i_2} & \mathbf{t}_i \end{bmatrix}}_{\mathbf{H}_i} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{16}$$

From this, it is clear that $\mathbf{H}_i$ is a homography transforming from the space of the ith calibration board to the normalized image plane. We invert the homography to perform the mapping

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{H}_i^{-1} \begin{bmatrix} x_{\mathrm{n}} \\ y_{\mathrm{n}} \\ 1 \end{bmatrix}. \tag{17}$$

Because the Levenberg-Marquardt algorithm is gradient-based, we need derivatives. We designed our texture function $\mathrm{T_G}$ to be smooth and differentiable, and fortunately the function P^{-1} is also differentiable, which implies that C_i is differentiable. Our implementation uses dual numbers for computing analytical derivatives.

4. RESULTS

When comparing a camera calibration to the ground truth, one could measure errors of each parameter individually,[11] but this is difficult to interpret, especially for distortion parameters as they can counteract each other. Motivated by this, we introduce *per-pixel reprojection error*, which measures the root mean squared distance in pixels between points projected with the true and estimated camera intrinsics. For each pixel, the image plane coordinates x, y define a line in $\mathbb{R}^3$ along which we select a point $\mathbf{q}_{xy}$ that projects to this pixel:

$$\mathrm{P}(\mathbf{K}, \mathbf{d}, \mathbf{q}_{xy}) = \begin{bmatrix} x \\ y \end{bmatrix}, \tag{18}$$

where $\mathbf{K}$ is the true camera matrix and $\mathbf{d}$ are the true distortion coefficients. We can now compute the per-pixel reprojection error E by using the estimated parameters to project the same points. Computing the differences, we have

$$E = \sqrt{\sum_{x,y} \left|\left| \mathrm{P}(\hat{\mathbf{K}}, \hat{\mathbf{d}}, \mathbf{q}_{xy}) - \begin{bmatrix} x \\ y \end{bmatrix} \right|\right|_2^2}, \tag{19}$$

where $\hat{\mathbf{K}}$ is the estimated camera matrix, $\hat{\mathbf{d}}$ are the estimated distortion coefficients and x, y sum over all possible pixel locations.

4.1 Synthetic data

We generate a set of 500 images of size 1920×1080 with a virtual camera with focal length $f = 1000$ each containing a single 17×24 checkerboard. The images are rendered so that the pixel intensities lie in the range $[0.1; 0.9]$. Figure 2 shows examples of these images. Each image is blurred by filtering it with a Gaussian kernel with zero mean and standard deviation σ. After this we add normally distributed noise to each pixel with zero mean and standard deviation σ_n, examples of this can be seen in Figure 3.

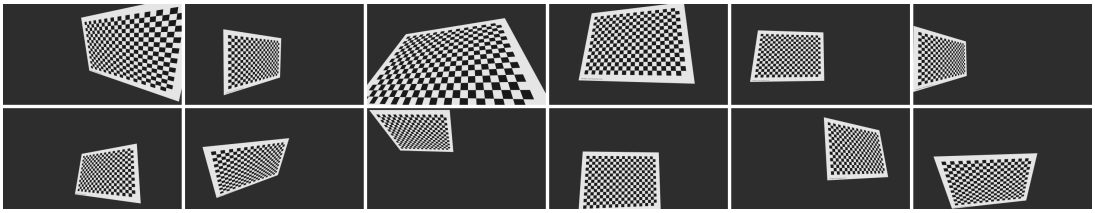


Figure 2: Sample images from our synthetic image dataset.

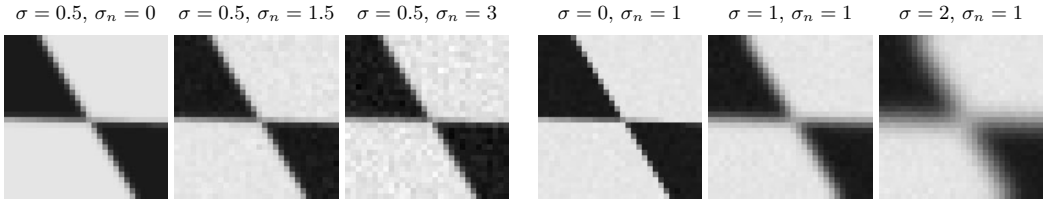


Figure 3: A corner from a checkerboard in a synthetic image with various levels of blur and noise added.

We select n random images from these and use the checkerboard detector from Ha *et al.*[7] with default parameters to detect points. After this, we use the standard method by Zhang[1] to compute the camera calibration. This is a calibration we compare with (Ha *et al.*), but also our initial guess for Equation (14). We do this for $n \in \{3, 20, 50\}$ and for varying values of σ and σ_n. For each n, σ and σ_n we perform 25 trials with randomly sampled images. The results of these experiments are in Figure 4. For comparison with OpenCV,[18] we use the detected points as initialization for `cornerSubPix`,[3] with a 5×5 window. As the images are rendered without distortion, we do the calibration without distortion as well.

We observe that our method performs better than Ha *et al.*[7] and OpenCV[3,18] for each n across various levels of noise and blur, except for $n = 3$ in cases with much blur. We also observe that our method is consistently better in noisy situations.

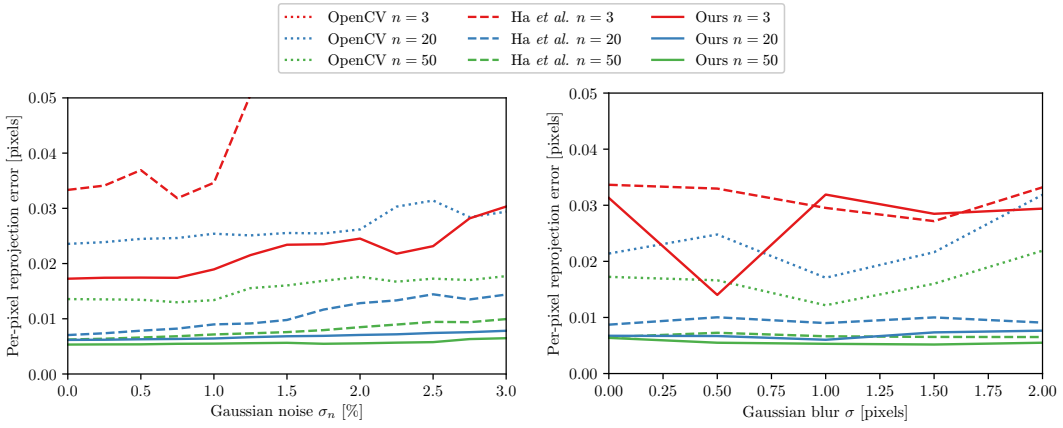


Figure 4: Comparison of our method with Ha *et al.*[7] and OpenCV[3,18] for varying number of images used in calibration (n). Left: Varying σ_n with fixed $\sigma = 0.5$. Right: Varying σ with fixed $\sigma_n = 1\%$. OpenCV $n = 3$ lies beyond the plotted area.

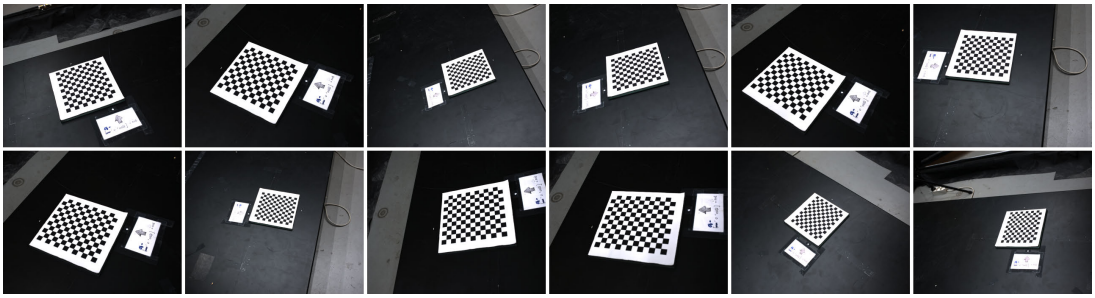


Figure 5: Sample images from our real image dataset.

4.2 Real data

When comparing camera calibrations on real data, an often reported measure is the reprojection error of all points. That is however not something we are able to do as our method incorporates the constraint of the calibration object geometry, and the reprojection error will thus per definition always be zero. Based on Figure 4, the points detected by the detector from Ha *et al.*[7] are clearly quite accurate, which motivates us to use it as a pseudo-ground truth.

We use a dataset of 128 images at a resolution of 3376×2704, each containing a 12×13 checkerboard. We randomly select 64 images to use as our test set, and detect points in them with Ha *et al.*[7] which we use as pseudo-ground truth. Then we select n of the images not in the test set and use them to compute the camera intrinsics. For each image in the test set, we use the already detected points together with our camera calibration, to compute the pose of the checkerboard, which in turn allows us to project points to the camera, and thereby measure a reprojection error. For each n we partition the 64 images in the training set into non-overlapping sets of size n and do the camera calibration for each of them. Figure 6 and Table 1 show the performance of our method compared to Ha *et al.*[7] and OpenCV.[3] For $n < 5$ our method performs better and has a lower standard deviation. For large n, Ha *et al.*[7] achieve an extremely similar reprojection error, but the points we are computing the reprojection error against are also detected by their method. It can also be seen that the reprojection error of their method on the training data approaches the same values from below, so the best achievable test set reprojection error is limited either by the camera model or the accuracy of the detected points.

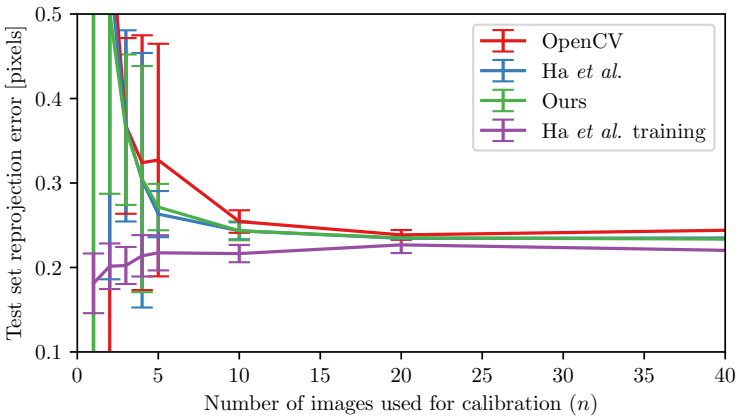


Figure 6: Reprojection error for images in the test set of our real dataset as a function of n. Bars on each point show ± 1 standard deviation.

Table 1: Data from Figure 6. ± indicates the standard deviation of the reprojection error.

n	2	3	4	5
OpenCV	0.61 ± 0.52	0.37 ± 0.10	0.32 ± 0.15	0.33 ± 0.14
Ha *et al.*	0.55 ± 0.36	0.37 ± 0.11	0.30 ± 0.15	0.26 ± 0.03
Ours	0.50 ± 0.22	0.36 ± 0.09	0.30 ± 0.13	0.27 ± 0.03

5. DISCUSSION

Although we have only used this method to compute intrinsics of a single camera in this paper, it is straightforward to extend to intrinsics of multiple cameras and their extrinsics. The homography $\mathbf{H}_i$ can easily incorporate the pose of the camera, and then all one needs is a separate set of parameters per camera.

Even though we have chosen to use the Brown-Conrady[14,15] distortion model in our work, this is a choice mostly motivated by being able to fairly compare with OpenCV.[18] Our method is not tailored to this distortion model, and one could replace it with another, such as the division model.[19]

We do not attempt to match the scaling of the image intensities in the rendering as in the work of Rehder *et al.*[12] We experimented with scaling the image intensities to match the rendering or including the local intensity of the rendering as a parameter in the optimization as well, but we did not observe any increase in accuracy when doing this.

Our method takes around three minutes to solve the optimization problem for 40 images from our real dataset, where each image is 9 Megapixels. We find this to be an acceptable computation time, especially given that even one such problem contains around 30 million residuals.

6. CONCLUSION

We have introduced a method for improving camera calibration based on minimizing the sum of squared differences between real and rendered images of textured flat calibration objects. Our rendering pipeline consists purely of analytically differentiable functions, which allows for exact gradients to be computed making the convergence of the optimization more robust and fast, while still allowing us to blur the image in the image space as would naturally occur. On synthetic data, our method outperforms state-of-the-art camera calibration based on point detection, for images distorted by Gaussian blur and noise.

On real data, our method exhibits a clear advantage when only a few images are available for calibration, and performs at least as well for a larger number of images, but we have not been able to verify whether our method outperforms the existing methods in this case, due to the difficulty of evaluating which of two estimated camera intrinsics is better.

REFERENCES

[1] Zhang, Z., "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence* **22** (2000).

[2] Harris, C. and Stephens, M., "A combined corner and edge detector," in [*Proceedings of the Alvey Vision Conference*], Taylor, C. J., ed., 23.1–23.6 (1988).

[3] Förstner, W. and Gülch, E., "A fast operator for detection and precise location of distinct points, corners and centres of circular features," in [*Proc. ISPRS intercommission conference on fast processing of photogrammetric data*], 281–305, Interlaken (1987).

[4] Geiger, A., Moosmann, F., Car, Ö., and Schuster, B., "Automatic camera and range sensor calibration using a single shot," in [*International Conference on Robotics and Automation (ICRA)*], 3936–3943, IEEE (2012).

[5] Chen, B., Xiong, C., and Zhang, Q., "CCDN: Checkerboard corner detection network for robust camera calibration," in [*International Conference on Intelligent Robotics and Applications (ICIRA)*], 324–334, Springer (2018).

[6] Ding, W., Liu, X., Xu, D., Zhang, D., and Zhang, Z., "A robust detection method of control points for calibration and measurement with defocused images," *IEEE Transactions on Instrumentation and Measurement* **66**(10), 2725–2735 (2017).

[7] Ha, H., Perdoch, M., Alismail, H., So Kweon, I., and Sheikh, Y., "Deltille grids for geometric camera calibration," in [*The IEEE International Conference on Computer Vision (ICCV)*], 5344–5352 (Oct 2017).

[8] Lucchese, L. and Mitra, S. K., "Using Saddle Points for Subpixel Feature Detection in Camera Calibration Targets," *Computer Engineering* , 191–195 (2002).

[9] Placht, S., Fürsattel, P., Mengue, E. A., Hofmann, H., Schaller, C., Balda, M., and Angelopoulou, E., "Rochade: Robust checkerboard advanced detection for camera calibration," in [*European Conference on Computer Vision (ECCV)*], 766–779, Springer (2014).

[10] Datta, A., Kim, J. S., and Kanade, T., "Accurate camera calibration using iterative refinement of control points," *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009* , 1201–1208 (2009).

[11] Zhang, Z., Matsushita, Y., and Ma, Y., "Camera calibration with lens distortion from low-rank textures," in [*Conference on Computer Vision and Pattern Recognition (CVPR)*], 2321–2328, IEEE (2011).

[12] Rehder, J., Nikolic, J., Schneider, T., and Siegwart, R., "A direct formulation for camera calibration," in [*International Conference on Robotics and Automation (ICRA)*], 6479–6486, IEEE (2017).

[13] Rehder, J. and Siegwart, R., "Camera/IMU calibration revisited," *IEEE Sensors Journal* **17**(11), 3257–3268 (2017).

[14] Brown, D. C., "Decentering distortion of lenses," *Photogrammetric Engineering and Remote Sensing* (1966).

[15] Conrady, A. E., "Decentred lens-systems," *Monthly notices of the royal astronomical society* **79**(5), 384–390 (1919).

[16] Levenberg, K., "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics* **2**(2), 164–168 (1944).

[17] Marquardt, D. W., "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics* **11**(2), 431–441 (1963).

[18] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools* (2000).

[19] Fitzgibbon, A. W., "Simultaneous linear estimation of multiple view geometry and lens distortion," in [*Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*], I–125–I–132, IEEE (2001).

CONTRIBUTION B

Design of Automated Robotic System for Draping of Prepreg Composite Fabrics

doi:10.1017/S0263574720000193

Design of Automated Robotic System for Draping Prepreg Composite Fabrics

L.-P. Ellekilde†, J. Wilm‡, O. W. Nielsen†, C. Krogh¶, E. Kristiansen¶*, G. G. Gunnarsson†, T. S. Stenvang†, J. Jakobsen¶, M. Kristiansen¶, J. A. Glud¶, M. Hannemose‡, H. Aanæs‡, J. de Kruijk§, I. Sveidahl||, A. Ikram# and H. G. Petersen†

†*University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark*
E-mails: lpe@mmmi.sdu.dk, own@mmmi.sdu.dk, gunu@mmmi.sdu.dk, thss@mmmi.sdu.dk, hgp@mmmi.sdu.dk
‡*Technical University of Denmark, Building 321, 2800 Lyngby, Denmark,*
E-mails: jakw@dtu.dk, mohan@dtu.dk, aanes@dtu.dk
¶*Aalborg University, Fibigerstræde 16, 9220 Aalborg Ø, Denmark*
E-mails: ck@m-tech.aau.dk, joj@m-tech.aau.dk, morten@m-tech.aau.dk, jag@m-tech.aau.dk
§*Netherlands Aerospace Centre, Voorsterweg 31, 8316 PR Marknesse, The Netherlands*
E-mail: joachim.de.kruijk@nlr.nl
||*RoboTool A/S,Vestermarksvej 10, 6600 Vejen, Denmark*
E-mail: is@robotool.com
#*Terma Aerostructures A/S, Fabrikvej 1, 8500 Grenaa, Denmark*
E-mail: aik@terma.com

(Accepted March 30, 2020)

SUMMARY
This paper presents a novel solution for precision draping of prepreg composite fabrics onto double curved molds. Our contributions relate to system design, including hardware and software components, and to system integration. On the hardware side, design and implementation of a drape tool with up to 120 suction cups positioned individually by linear actuators are described. On the software side, design and implementation of the software architecture are presented, along with necessary algorithms within sensor technologies and mathematical modeling. The essential system's components were verified individually, and the entire integrated system was successfully validated in the Proof-of-Concept experiments, performed on an experimental physical model of the system.

KEYWORDS: Industrial robotics; Composite manufacturing; System development; Automated draping process.

1. Introduction

Manual draping of woven composite fabrics onto one-of-a-kind molds is a time-consuming and labor-intensive process. When using fabrics, which are pre-impregnated with epoxy resin (prepreg material), the process also poses potential health risks to the operator. These factors hinder a more extensive application of composites in areas such as aerospace, wind turbines, automotive and shipbuilding. Hence, automation of the lay-up process could be highly beneficial. The FlexDraper

* Corresponding author. E-mail: ewa@m-tech.aau.dk

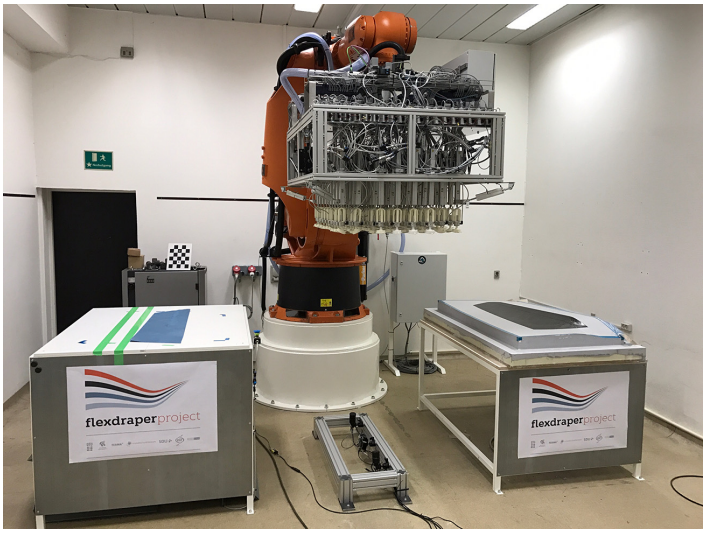


Fig. 1. Flexdraper, the developed experimental physical model of the system for draping the prepreg fabrics, which is comprised of an industrial robot, a drape tool with 80 suction cups on linear actuators, a picking table (on the left) and a mold table (on the right).

consortium was founded in 2013 to address this need and to develop a complete solution for automated draping of composite textiles. It consists of university and industrial partners with expertise in robotic systems, automation, vision, composite manufacturing and material modeling.

In this paper, we present a novel solution for high precision draping of prepreg composite fabrics onto one-of-a-kind double curved molds, using an automated robotic system, also refered to as a FlexDraper. Our contributions relate to the system design, including hardware and software components, and also to the system integration. The design of hardware components consists of mechanical design of the drape tool and design of a vision-based sensor system and relies on the kinematic and fabric modeling, verified in simulation. Special emphasis is put on design of software components for automatic planning of draping sequence and automatic planning of suction cup's trajectories for a given draping strategy. A strategy is defined here as a path initiated from a configuration where all suction cups are located above the mold (called the preshape) and end with all suction cups placed on the mold and the ply has been correctly draped. The draping strategy applied in this paper is inspired by manual draping in the wave-like pattern, where the drape starts by touching down at a well-chosen location from which the drape propagates over the lay-up surface with a wave-like movement. The aforementioned automatic planning requires an extensive modeling of fiber plies and a dynamic modeling of drape tool, which are both discussed in this article.

The drape task consists of picking pre-cut plies of prepreg fabric from a flat table surface and draping these onto the current stack of plies on a double curved mold. The pre-impregnated epoxy causes a certain stickiness, which makes it challenging to grip this material, and to complete the task without wrinkle formation. The developed FlexDraper system is shown in Fig. 1. The designed drape tool consists of up to 12×10 suction cups with individual linear actuators arranged in a rectangular grid, and it is moved by a six-axis industrial robot (Kuka KR360L2800). The integrated FlexDraper system for automated draping of prepreg composite fibers was successfully tested in the Proof-of-Concept experiments, performed on an experimental physical model of the system.

The paper is organized as follows: In the subsequent section, we discuss related approaches and existing systems for draping composite materials. Afterward, we present an overview of the proposed FlexDraper system. The succeeding section describes design and implementation of software architecture, followed by detailed insights into different system's components. Finally, the integrated system is validated in the Proof-of-Concept experiments, leading to a conclusion with an outlook toward future work.

2. State of the Art

For automating the manufacturing of large composite structures such as aircraft hulls, the two main technologies are applied: Automated Fiber Placement and Automated Tape Laying.[1] However, as documented in ref. [1], such systems are very expensive to acquire. Furthermore, their expected performance decreases significantly, when the parts become smaller. Therefore, the existing systems are not economically viable for smaller sized parts and in low volume production.

As stated in refs. [2, 3], there have been earlier initiatives that attempted to find the automated and competitive solutions for smaller parts. The majority of them focus on dry fabrics without resin. Compared to the prepreg fabric, the processing and avoiding wrinkle formation is much easier with such dry fabric because the dry fabric is not tacky and it is more flexible. Within our scope, previous designs occur only in connection with R&D environments.

Some examples of state-of-the-art robot end effectors for draping carbon fiber fabrics are described in refs. [2, 4–8], which are based mainly on tools with multiple actuated gripper grids, capable of approximating the desired 3D shape. The articles refs. [2, 4, 5, 8] focus on dry fiber material, whereas the work presented in refs. [6, 7] is also discussing prepreg fabrics and has many similarities to ours. However, it only considers the tool development and does not describe the necessary planning, modeling and sensor systems described in this paper. An alternative to the aforementioned multigripper grids is presented in refs. [9, 10], which describes a cylindrical tool capable of draping dry fiber plies through a rolling motion. The work shows promising results for single curved parts, but it is not generalized to cope with double curved parts considered in FlexDraper.

One of the very few comparable automated pick and drape developments, which demonstrated several prototype parts, was successfully demonstrated by Netherlands Aerospace Center and Umeco in the COALESCE project and presented in refs. [11, 12]. In COALESCE, a ply was picked up from a flat surface by a robot with an inflatable membrane end effector and the protective film was removed automatically. The material on the inflatable membrane was then slightly heated in order to improve both flexibility and tackiness. Afterward, the prepreg was draped onto a highly curved and complex mold. The system and process developed in COALESCE were found to be very cost-effective, but showed several drawbacks as well. Some of the plies needed to be cut into smaller pieces, which made it impossible to implement in the existing formal aerospace process qualification of the addressed part. Another disadvantage was that the robot was programmed manually, which did not allow for automatic adaptation to new parts.

To limit set-up and programming efforts, Bombardier and partners launched the *Rapid Dry Carbon Fiber Lay-up* program,[13] which uses Computer-aided design (CAD) data to aid the automated movement coordination of their multi-gripper grid. This program goes one step further than COALESCE in terms of using available CAD data for automated movement coordination. However, it also only demonstrates the less challenging dry fiber material lay-up. Hence, even though there is a very high pressure within the market itself to lower production costs and maintain or increase quality for smaller prepreg parts, so far no complete solution has been demonstrated.

3. Design of the FlexDraper System

In this section, the designed FlexDraper system is presented, including an overview of its components. The FlexDraper system is built according to the flow chart shown in Fig. 2.

First, the *suction cups are aligned* to a co-planar array in the horizontal direction and then a sensorial measurement is performed to *estimate the poses of suction cups*. Knowing the positions of each suction cup, the *ply is picked up* from a given position on a table and the *foil is removed*. Since we have measured the positions of the suction cups relative to the robot, knowledge is available about the positions of all the suction cups relative to the ply at the moment when it is picked. Based on the 2D ply contour and the shape of the 3D mold, where the ply is to be placed, a method has been implemented that *maps points* on the 2D ply to a desired corresponding 3D points on the mold (see Section 6.1). Hence, knowing the pick locations of suction cups, the corresponding desired cup placements on the mold can be computed.

The *Plan draping sequence* component generates a sequence of suction cups' target poses. The first target pose of the suction cups in the sequence is the preshape and gives the shape of the ply immediately before the placement of the first suction cups onto the mold surface. The subsequent steps consist of selecting a first point of contact between the ply and the mold and from there

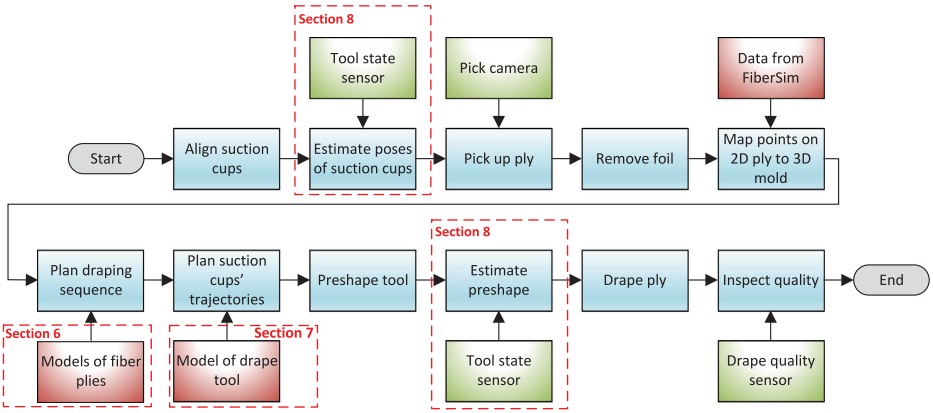


Fig. 2. Components in the FlexDraper system: the blue components represent the flow of actions, the green components are vision-based sensors and red components are data and computational models. The components encircled with the dashed, red line are explained in detail, in the referenced sections.

generating a wave-like pattern to perform the actual drape. In order to predict the draping outcome of this sequence, the system relies on the mathematical models of fiber plies presented in Section 6.2.

With a known sequence of suction cups' target poses, the *Plan suction cup' trajectories* component derives the trajectories of suction cups, using inverse kinematics and given the state of robot and all linear actuators. The drape tool mechanism is underactuated and with a non-trivial interrelation between the connected suction cups. Thus, a mathematical model (see Section 7) describing the dynamics of the tool has been developed and is used for simulating the tool behavior while computing the inverse kinematic solution, which minimizes the difference between the desired and actual suction cups' poses.

In order to account for inaccuracies in the mathematical models of the drape tool and the fiber plies, a custom vision-based sensor system is employed (see Section 8) and consists of *Tool state sensor*, *Drape quality sensor* and Pick camera. The Tool state sensor is employed to *Estimate poses of suction cups* and to *Estimate preshape*. The sensor data are providing the control variable for the actuators.

After the tool is preshaped and the preshape is estimated, the ply is moved to the mold and *draped* according to the planed sequence. Finally, the draped part is going through the *Quality inspection system*, which is relying on a *Drape quality sensor* and is intended for testing the draping quality with respect to specifications concerning location accuracy, avoidance of wrinkles and air intrusions.

4. Hardware Design and Implementation of a Drape Tool and a Suction Cup

In order to be able to lay-up plies that follow the very different contours of the lay-up mold accurately, it is necessary to have an end effector, which can be configured to the contours. For this purpose, we have developed an adaptive, actively manipulated gripper grid. Concerning the design of the gripper grid, several attributes and requirements had to be taken into account:

- The gripper must be able to handle plies of up to 1200 × 1000 mm in size with a height variation of up to 150 mm.
- Necessary lifting force of the individual suction cups must be sufficient.
- Distances between suction cups must be limited to be able to manipulate the material.
- Ability to comply with a wide range of lay-ups on double curved surfaces with small to medium curvature.
- The overall weight of the gripper must be within the payload limits of the robot.
- Electrical and mechanical complexity should be limited to improve robustness and reduce cost.

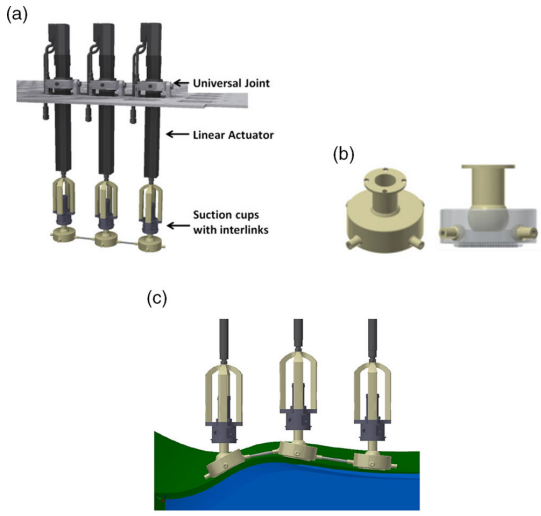


Fig. 3. Drape tool and suction cup design and functionality. (a) Part of tool showing the relative placement of the universal joints, actuators and suction cups. (b) Close-up of suction cups with ball joints for connecting with the actuator and interlink structure. (c) Illustration of interlink structure when placed at different heights.

Throughout the experimentation, we found out that to enable the preshape with medium curvature and details, the distances between suction cups need to be rather small. This conclusion also supports the criterion for necessary lifting force and limited free-hanging material. On the current FlexDraper tool, a grid spacing of approximately 110 mm between adjacent suction cups has been selected. The chosen spacing distance imposes the requirement of up to $12 \times 10 = 120$ grippers to handle the specified part dimensions. Considering the weight and complexity issue, the number of actuators should be limited. We have therefore chosen to use only one linear actuator per suction cup for up/down movements and employ passive joints and interlinks for allowing the cups to adjust the horizontal position and comply the orientation to the mold. Figure 3a highlights a part of the tool with three suction cups and interlinks.

We use SCHMALZ Composite Grippers (SCGs), which create suction by means of the Venturi effect. The grippers are equipped with a custom-designed suction cup housing (see Fig. 3b) with integral ball joints, which can turn up to ± 40 degrees. A set of so-called interlinks between the suction cups have been attached to force the position and orientation of suction cups and naturally adjust to a surface of changing height (see Fig. 3c).

The suction cups are positioned by electric linear actuators with spindle drive (FESTO EPCO-16-150-3P-ST-E). The actuators can be moved individually and can use position, velocity and force for feedback control. The stroke length of the FESTO actuators (Z-direction) is 150 mm. The gripper sub-assemblies of linear actuators and SCGs with custom suction cup housings are mounted equidistantly on a base plate. To passively account for the changes in horizontal distance of the suction cups from a 2D flat surface to a 3D-shaped contour, we mount the actuators on universal joints allowing them to rotate and thereby satisfy the constraints of interlink structure. Even with this drape tool design consisting of only one actuator per suction cup, the total weight of the whole assembly due to the actuators, control boxes, power supplies, pneumatic components, wiring and tubing is adding up to approximately 250 kg.

Finally, it should be mentioned that in order to be able to handle different ply shapes, suction will only be applied to the relevant areas of the multi gripper grid. Not only because of air/energy consumption but also due to the fact that the inactive grippers also cause loss of suction when they are on the same local air supply circuit as active grippers. Additionally, the suction control prevents the potential problem of laying down plies not covering the complete area of the previous ply, as inactive grippers with suction may get close to the previous yet not fully attached ply.

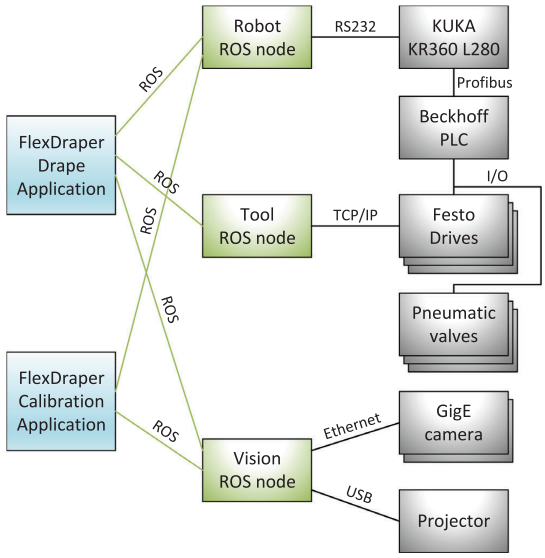


Fig. 4. System architecture showing the application components (blue), ROS components implementing functionality blocks and hardware abstractions (green) and hardware entities (grey).

5. Design and Implementation of Software Architecture for a FlexDraper System

This section describes the software architecture for the FlexDraper system. In order to bind its components together, the system uses ROS[14] as a middleware for enabling functionality to be distributed across multiple processes and computers. At the same time, ROS also provides a natural decoupling of components enabling project partners to work in parallel and update functionality as long as the components comply to the agreed ROS service interfaces.

The main system architecture is shown in Fig. 4 with the different application components shown to the left. These currently includes: (1) FlexDraper Drape Application, which implements the drape planning and execution strategy and (2) Calibration Application used for calibrating the robot to the sensors and to the mold.

The second column in Fig. 4 represents nodes responsible for a specific functionality, such as communication with the robot and the tool. The ROS interfaces for these components are designed to be hardware independent, which allows for future upgrades of hardware without changing the application components.

The right column in Fig. 4 represents the hardware entities and not ROS nodes. A *Kuka KR360L280* robot is controlled through an RS232 serial interface. The communication follows a client–server pattern with the robot being the client polling the server for tasks. The robot has the capability of buffering up to N tasks enabling it to blend smoothly between motions. Besides point-to-point and linear motions, the current interface also supports setting and reading I/O, reading the robot configuration and wait statements.

The main communication with the tool is done through Transmission Control Protocol/Internet Protocol (TCP/IP), where each of the controllers for the FESTO actuators has their own IP address. Using the TCP/IP connection, the controllers are configured individually from the ROS node with desired goals and velocity profiles. To ensure a synchronized start of the actuators, they are set up to start based on a digital I/O signal. The valves used for controling suction of the tool are likewise controlled through digital I/O. All mentioned I/O signals are connected to a Beckhoff PLC build into the tool and controlled from the robot using ProfiBus. This solution provides a great flexibility, as the Beckhoff PLC can easily be extended, and it does not require placement of additional wires along the robot in case of a need for more I/O signals.

Sensor systems and analysis are performed on a separate computer, which serves ROS nodes, so that the measurement updates can be requested and the results can be returned to the FlexDraper Drape Application.

6. Models of Fiber Plies

The first purpose of modeling the fiber plies is to calculate the optimal, final 3D location $r_{mold}(u, v)$, for any point (u, v) on the ply. The second purpose of modeling the fiber plies is to create an engine for planning draping strategies and for predicting the result of trial draping strategies. Since many trial strategies may be required before obtaining the correctly draped ply, it is important to have fast computable, but not necessarily highly accurate models for the initial planning. On the other hand, at or near the final solution, the models must be accurate, potentially at the expense of a longer computation time. Therefore, the studies are conducted on both less accurate, but fast approximate models and highly accurate, but computationally expensive models.

6.1. Estimating the transformation $r_{mold}(u, v)$

An initial estimation of the transformation of a point (u, v) on the flat ply on the picking table to a corresponding point on the curved mold geometry $r_{mold}(u, v)$ is done by a surface mapping algorithm. The algorithm is based on work by ref. [15] and extended with ply boundary constraints. It serves to compute the position and orientation of the suction cups on the mold based on the contact points between suction cup and ply during the pick up. The algorithm is based on a minimization of the energy density in the fibers of the ply material. Therefore, the solution seeks to minimize fiber deformations. Prior to initiating the minimization routine, a starting point is singled out and then a fiber together with its orthogonal at that point are mapped to the mold. A discretization size is chosen, and the energy minimization of the domain described by $r_{mold}(u, v)$ is executed.

6.2. Computationally fast models

Computationally fast models of the fiber ply deformations during draping are required for efficient execution of the initially broad search space of the drape planning. The typical approach for fast models of fabrics and other flexible materials is a damped mass spring model (see for example[16]). However, for the prepreg material used in this project, the stiffness in the fiber direction leads to very stiff springs and hence requires a very small time step, leading to slow computations. Rather than using the mass spring model, a static geometric model has been developed. This model takes advantage of the very high arc length stiffness in the fiber directions and computes only the equilibrium solution and not the internal modes, which are due to, for example, accelerations of the drape tool.

The following assumptions have been made in our model.

- Gravity can be neglected compared to material forces.
- Vibrational modes can be neglected, so only the equilibrium solution is needed.
- The material is first-order continuous at all points.
- The arc length in fiber directions between adjacent suction cups is preserved.
- There is no sliding of material relative to the suction cups.

With these assumptions, the ply can be modeled by a fiber length preserving polynomial interpolation. A combination of third- and fourth-order polynomials has been used.

$$\mathbf{r}^*(s) = \mathbf{a}_3^* s^3 + \mathbf{a}_2^* s^2 + \mathbf{a}_1^* s + \mathbf{a}_0^* \tag{1}$$

The $\mathbf{a}^*$ parameters can be found using boundary conditions. By adding a length constraint a fourth-order polynomial can be described as:

$$\mathbf{r}(s) = \mathbf{r}^*(s) + a_4^* \mathbf{d} s^2 (L - s)^2. \tag{2}$$

L denotes the length of a fiber between two constrained points and $\mathbf{d}$ is defined as:

$$\mathbf{d} = \mathbf{r}(\tfrac{L}{2}) - \tfrac{1}{2}\big(\mathbf{r}(0) + \mathbf{r}(L)\big). \tag{3}$$

In order to compute a_4^*, it is assumed that L is known and that the arc length in parametric form is given by:

$$L \equiv \int_0^L \sqrt{\mathbf{r}' \cdot \mathbf{r}'} ds. \tag{4}$$

Using Eqs. (1)–(4) for selected fibers connecting suction cups in both directions expands a grid. The enclosed areas can then be interpolated by a location-dependent weighed average, resulting in a function of the form. $r(u, v) = \{x(u, v); y(u, v); z(u, v)\}$, where u, v describe the $2D$ location on the ply and $r(u, v)$ describes the corresponding $3D$ location as a function of the position and orientation of the four nearest suction cups. For a more detailed derivation refer to ref. [17]. The advantage of this model is that it fits the experimental data well and that it is computational cheap. For an 10×8 gripper grid and with non-optimized MATLAB code, the model is using approximately 0.6 s on a conventional PC. The fact of working with an equilibrium solution makes the found parameters path invariant. It reduces the search space drastically, as it allows for the draping strategies to be tested by simulating subparts of the strategy separately. This contributes to further reduction of computational time.

6.3. Advanced FEM type models

In order to simulate the outcome of a finalized draping sequence and to confirm that the planned suction cups' trajectories that were learned and programmed using the computationally fast model are acceptable, a highly accurate model is developed. The accurate model considers the peculiar mechanical responses of the prepreg. Due to the arrangement of the fibers in the ply, the fibers can rotate in the weave, i.e. shear, such that the initial 90° fiber angles change. As a consequence, the material is highly anisotropic, i.e. the stiffness in the fiber direction is higher than in diagonal directions by several orders of magnitude. In addition, the stiffness is non-linear in the deformation (strain) as well as dependent on the rate of deformation due to the presence of the uncured resin (viscoelasticity). At large straining, permanent deformations, such as plasticity can come into effect. The tackiness of the ply affects the frictional properties in the interfaces between the ply and the suction cups as well as in the interface between the ply and the mold. This in turn affects whether the ply slides or deforms when being manipulated with the suction cups. These transient aspects of the draping must therefore also be considered in a highly accurate model.

A closed form solution that can represent the mechanical response for arbitrary ply geometries is not available, and thus the Finite Element (FE) method is deployed. Numerous accounts of successful application of FE simulations of woven reinforcement drapings are found in the literature.[18] Here, a so-called continuous model is applied, where homogenization theory and shell elements are used for modeling the otherwise non-homogeneous nature of the plies. Our material model is based on the measured material response from standardized tests. The in-plane ply response can be measured using a universal testing machine, where the force required to elongate a ply sample is recorded. For the fiber direction response, the fibers must be aligned with the direction of loading. To measure the shear, i.e. change in fiber angles, the sample must be oriented 45° to the loading direction as seen in Fig. 5a. This 45°-test is known as the bias-extension test.[19] By means of kinematics, the sample elongation can be converted to the change in shear angle and the recorded load can be converted to the shear load in the ply. The results from a bias-extension test on the prepreg material for three different displacement rates are presented in Fig. 5b.

Relevant output of the FE model includes: undesired formation of wrinkles, resulting local fiber angles, amount of shear and reaction forces at the interfaces that can be used for identifying sliding effects. Although the FE model requires a substantial number of inputs and the solution time can be in the order of hours, it is a powerful tool for offline studies and for verification of draping strategies. An example of the results from the FE model is shown in Fig. 6. Details on the material characterization, together with the FE modeling and the results can be found in ref. [20].

7. Model of Drape Tool

Initially, there conducted a study on the resulting degrees of freedom related to an underactuated tool. Consider here an $M \times N$ array of suction cups and hence $M \times N$ actuators. As each suction cup has 6 degrees of freedom, there would be $6MN$ degrees of freedom if no constraints were present. With

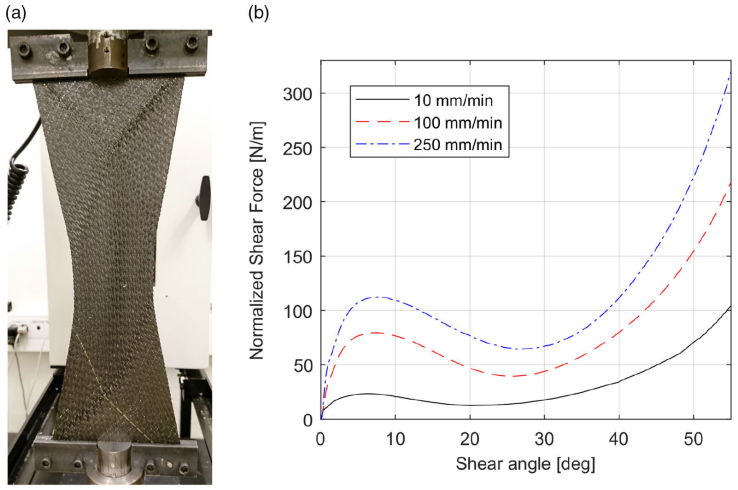


Fig. 5. The bias-extension test. (a) Test set-up with 120×270 mm sample. (b) Averaged shear force vs shear angle results. Three samples were tested for each rate.

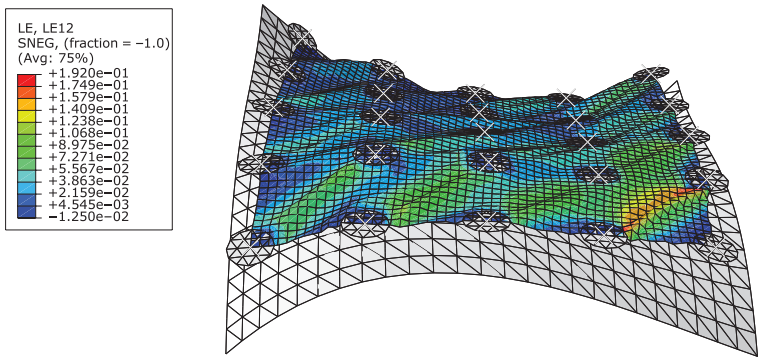


Fig. 6. A FE model of a 5×5 grid of suction cups in a free-hanging configuration. The colors indicate amount of shear.

the actuators at fixed configurations, there are MN actuator constraints and $M(N-1)+N(M-1)$ interlink constraints. This results with an underactuated system with $3MN+M+N$ internal degrees of freedom when the actuators are locked.

The underactuated drape tool has all the advantages previously listed, but the disadvantage is that the underactuation increases the difficulty of drape tool modeling. However, our experiments have indicated that at least there is a high degree of repeatability of the configurations when the plies are attached because the forces between the suction cups and the material determine the configuration.

This indicates that it is possible to model these forces correctly and thus predict the suction cup positions. To exploit that, a model is developed, which takes these forces as input and computes the resulting suction cup configuration. Additionally, a so-called *Tool state sensor*, described in the Section 8, is measuring the actual location of the suction cups, which are first used for performing an iterative fine tuning of the model and afterward for validating the accuracy of the final model.

Each of the MN suction cup assemblies, shown in Fig. 3, is modeled as three rigid bodies. These bodies are: the actuator housing, the actuator rod and the suction cup. The inertia is defined relative to the rigid bodies and the velocities as spatial twists.[21] The generalized coordinates, q_i, for assembly

i are: $q_i = \left[\theta_{y,i}, \theta_{x,i}, \eta_i, \phi_{z,i}, \phi_{y,i}, \phi_{x,i}\right]^T$. The θ's are denoting the angles of the universal joint at the mounting point, first rotating about the y-axis and then the resulting x axis. η is the extension of the actuator, and the ϕ's are the z-y-x Euler angles of the suction cup. The total state vector q is made by stacking all the q_i vectors. The total kinetic energy of the whole tool using generalized coordinates is then:

$$T = \frac{1}{2}\sum_{i=1}^{MN}\sum_{j=1}^{3}\left[m_j\|\dot{\mathbf{p}}_j(q_i)\|^2 + \omega_j^T(q_i)\cdot\mathbf{I}_j\cdot\omega_j(q_i)\right] = \frac{1}{2}\dot{q}^T\cdot\mathbf{M}(q)\cdot\dot{q}, \tag{5}$$

where m_j is the mass of the jth body, $\dot{\mathbf{p}}_j(q_i)$ is the time derivative of the center of mass of the body, $\omega_j(q_i)$ is the angular velocity and $\mathbf{I}_j$ is the inertia matrix. The matrix $\mathbf{M}(q)$ is the generalized inertia matrix.

The interlink constraints are modeled as point-to-point constraints between assemblies. The rotation point of the interlink on each suction cup is defined as $\mathbf{r}_\alpha$ and $\mathbf{s}_\alpha$ for each interlink α. Then, the interlink constraint is expressed as:

$$f_\alpha(q) = \frac{1}{2}\left(\|\mathbf{r}_\alpha(q) - \mathbf{s}_\alpha(q)\|^2 - d^2\right) = 0, \quad \alpha = 1, ..., M(N-1) + N(M-1), \tag{6}$$

where d denotes the length of the interlink.

The movement profiles of the actuators are known and are modeled as time-dependent constraints on the η coordinates:

$$g_\beta(t, \eta) = \eta_\beta - c_\beta(t) = 0, \quad \beta = 1, ..., MN, \tag{7}$$

where c_β is the movement of the β'th actuator.

The potential energy is denoted by $V(q)$, and the damping forces in the tool, denoted by $\mathcal{F}(\dot{q})$, is derived by defining a Rayleigh dissipation function.[22]

Using constrained Lagrangian mechanics,[23] the equations of motion become:

$$\frac{d}{dt}\partial_{\dot{q}}T(q,\dot{q}) - \partial_q T(q,\dot{q}) + \partial_q V(q) + \partial_{\dot{q}}\mathcal{F}(\dot{q}) = \sum_\alpha \lambda_\alpha \partial_q f_\alpha(q) + \sum_\beta \mu_\beta \partial_q g_\beta(t,q) \tag{8}$$

$$\partial_q f_\alpha(q)\cdot\ddot{q} + \dot{q}^T\cdot\partial_{qq}f_\alpha(q)\cdot\dot{q} = 0 \tag{9}$$

$$\ddot{\eta}_\beta - \ddot{c}_\beta(t) = 0 \tag{10}$$

In Eq. (8), the variables λ and μ are Lagrange multipliers for the interlink constraints and the actuator constraints, respectively. The operator ∂ denotes the partial derivative with respect to the variable in the subscript. Equations (9) and (10) are acceleration constraints due to the interlinks and actuator movement. The acceleration constraints introduce drift in the position constraints during integration, which is mitigated using mass orthogonal projection[24] at every timestep.

8. Sensor System

Automated optical (vision) measurements are needed for performing several sub-tasks in FlexDraper. More specifically, the sensor system shown in Fig. 7 was designed. It consists of:

- *Pick camera*. This is an industrial camera mounted above the pick table. The contour of a ply is found using image processing algorithms.
- *Tool state sensor*. This is a structured light vision system positioned between pick and drape tables. Details of the sensor are given below.
- *Drape quality sensor*. This is an additional higher resolution structured light vision system. This sensor will be mounted above the drape table to measure the ply surface after a drape. The allowable tolerance for ply placement is +/− 2.5 mm on the outer contour and +/− 3 degree for the fiber angle at a specified point. Hence, the measurement accuracy must be below these tolerances. The drape quality sensor has not been installed yet as the draping quality is still manually inspected.

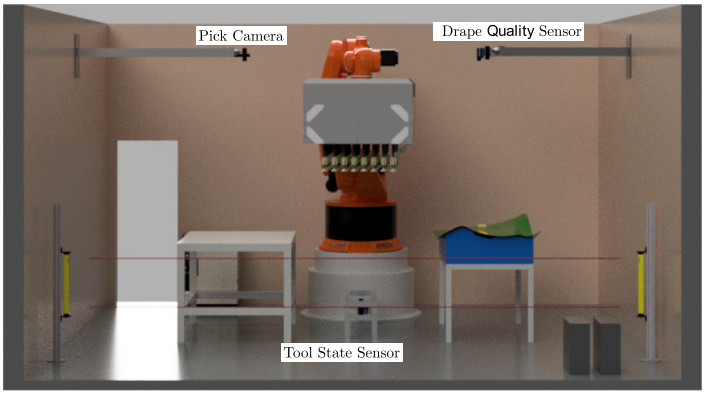


Fig. 7. Placement of sensors in the FlexDraper cell.

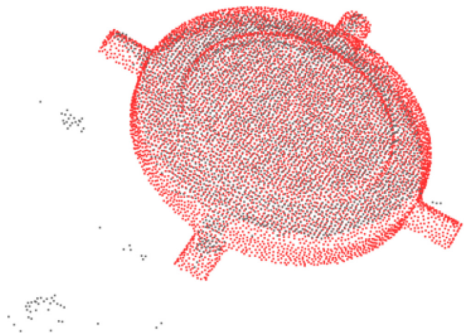


Fig. 8. Point cloud of one of the suction cups as captured by the tool state sensor.

The aforementioned tool state sensor serves two processes highlighted in Fig. 2:

- Estimate poses of suction cups
- Estimate preshape

The *estimation of poses of suction cups* is based on the measurements taken prior to picking the ply and it is necessary because due to the underactuation, the actual pose of each suction cup is unknown and impossible to model when no ply is attached. The estimated poses of suction cups are then used for predicting the locations, where the individual suction cups should pick up the ply.

For estimating the poses of suction cups, the 3D input data from the tool state sensor are acquired by scanning the suction cup array from below. The poses of the suction cups are recovered from the data by means of 3D pose estimation.

First, the 3D data are segmented into disjoint cells, corresponding to the individual suction cups, by means of a regular quadrilateral grid in camera space. Since the segmented regions remain constant, they are predefined in a manual operation. The pose of each suction cup is then recovered by robust alignment of the 3D CAD drawing of the suction cups.

Suction cups are found in the 3D data by fitting planes with the Random Sampling Consensus heuristic. The resulting locations and orientations are used for initializing point-to-plane-based Iterative closest point registration of the CAD suction cup to the data. Figure 8 shows alignment of a single suction cup, and Fig. 9 shows the resulting estimates of positions and orientations of the cups.

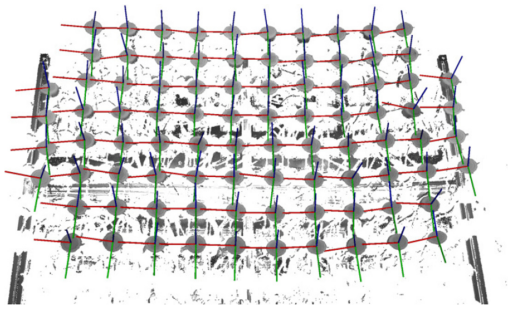


Fig. 9. Computed pose estimates of the suction cups.

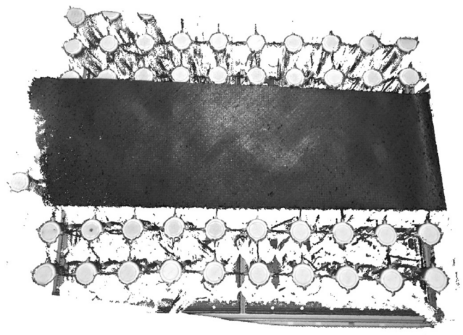


Fig. 10. Point cloud of a scanned fiber ply.

Estimate preshape is referring to the estimation of states of drape tool and ply after a ply is picked up. It is done based on the 3D scan of the ply, while it is held with the drape tool, combined with the states of drape tool and ply, which are predicted by the presented mathematical models.

The 3D data of the ply used in FlexDraper are acquired both from the tool state sensor and later on the drape quality sensor. Both sensors are structured light systems based on active stereo fringe projection,[25] which is able to achieve very high accuracy and precision while keeping scan times in the order of a few seconds.[26] The spatial encoding uses two frequency phase shifting with phase unwrapping based on the heterodyne principle.[27] This encoding method is particularly flexible and allows for the detection and elimination of falsely encoded surface points, which will inevitably occur due to the complex optical properties of carbon fiber composites.[28] Our experimentation shows that with careful consideration of these effects, excellent results of the 3D ply shapes can be achieved. Our solution is implemented with the tool state sensor, and Fig. 10 shows an example of a scanned ply.

The tool state sensor is comprised of two Gigabit Ethernet cameras (FLIR BFLY-PGE-23S6M-C) and a LightCrafter 4500 programmable pattern projector configured through USB. The projector emits blue light (465 nm). Hardware triggers the cameras for synchronized display and captures the structured light pattern sequence. In order to reduce ambient lighting effects, the cameras are equipped with narrow band-pass filters matching the illumination wavelength.

9. Proof-of-Concept Experimental Results

Most of the experiments performed until now on the experimental physical model of the system have had the aim to test and verify the electrical and mechanical robustness of the proposed solution to the integrated system. Initially, the pick and drape tests were taught manually and performed for making qualitative checks of system's repeatability, indicating that it can be modeled deterministically. The tests showed a position variation of the ply post drape of only a few millimeters, which is a promising result, since the tool state sensor is not yet used for correcting the positions. Moreover, the purpose

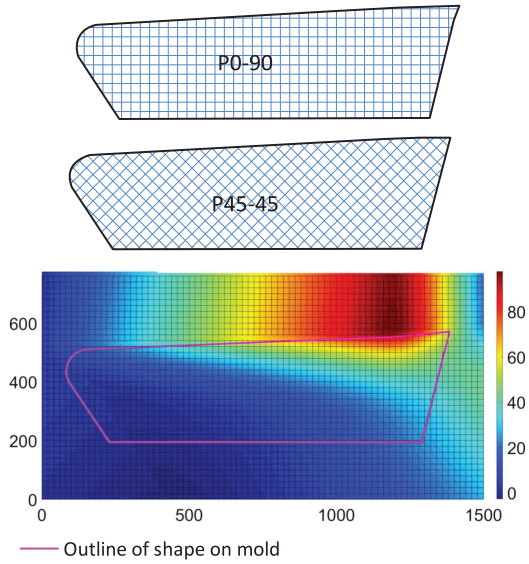


Fig. 11. Experimental set-up: two different orientations of prepreg material and a double curved mold. The dimensions of the mold are in mm and the height is color coded as shown on the scale bar.

of these manually taught tests has been to study if the system would be able to produce the required draping quality.

The conventional lay-up table of the part consists of a multilayer prepreg material with both full plies, covering the complete part surface, and partial plies, acting as local reinforcement layers in the assembly areas. The plies used in the Proof-of-Concept experiments are full pies, as shown in Fig. 11. They were draped on the mold with one layer of prepreg material, placed manually on the mold prior to the experiments and with no draping quality issues. A typical edge panel of 1200×350 mm with relatively low curvature has been selected. The chosen part has the shape and complexity representing a large product family of similar aerospace parts. The advantage of using this part over a more complex or larger and more material intensive part is also the possibility for quickly testing the different aspects of automation during the development of the automated process. The prepreg material used in the experiments had the 4 harness-satin weave fiber pattern. Two different orientations of the prepreg material with respect to the mold were tested: $P0-90$ and $P45-45$. The orientation $P0-90$ has the fibers aligned to the edges of the mold and the orientation $P45-45$ has the fibers rotated 45° with respect to the edges of the mold.

In the performed experiments with automated programming, the gripper grid preshaped the ply, in order to approximate the shape of the mold, and positioned the robot 20 mm above the drape location. Starting in one corner, the linear actuators of the tool created a wave-like motion, draping the ply onto the mold. Two types of wave strategies were tested: square wave strategy and skewed wave strategy, both shown in Fig. 12. The square wave strategy is staring from any single cup and moves outward in a ring-like wave. The skewed wave strategy is staring in a corner and moves on a straight line with the constant angle. Both strategies were applied to two different ply orientations $P0-90$ and $P45-45$, shown in Fig. 11.

The experimental results showed that the square wave strategy applied to $P0-90$ gives little to no wrinkling of the material. Whereas applying the same strategy to $P45-45$ resulted in significantly more wrinkling, which was caused by forces stretching the material in the diagonal direction of the fibers. Changing the draping strategy to the skewed wave reduced wrinkling of $P45-45$ samples. For validation purpose, the skewed strategy was also applied to $P0-90$, and as expected, it increased wrinkling in the drape quality.

A Proof-of-Concept experiment is demonstrated on the experimental physical model of the system, shown in Fig. 13. The results of an automated draping process can be seen in Fig. 14. Figure 14a

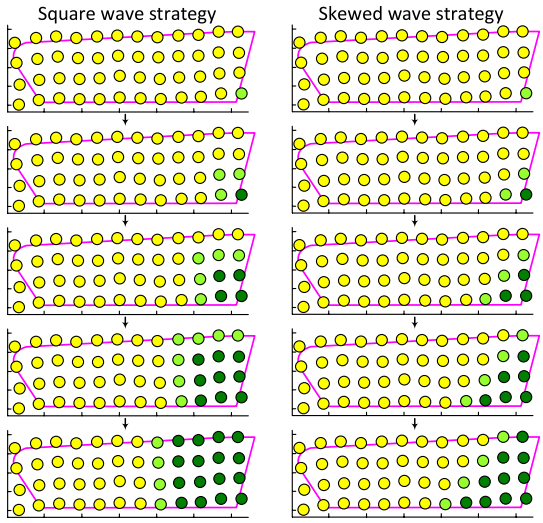


Fig. 12. Drape strategies used in the experiments, where color coding indicates position of the suction cup. Yellow is the preshape position, light green is the middle position and dark green is contact with the mold. The purple line illustrates the boundary of the ply.

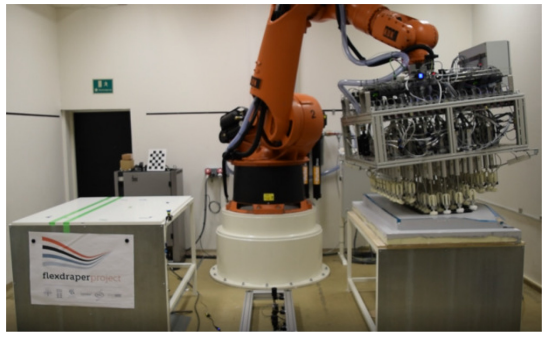


Fig. 13. (taken from Supplementary Video 1. Please refer to the supplementary material for the full video): Proof-of-Concept experiment showing automated draping process, performed on the experimental physical model of the system.

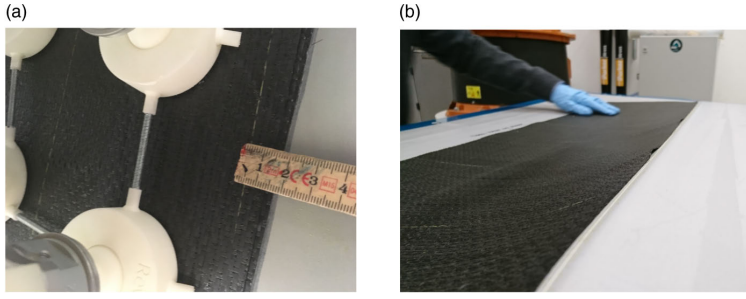


Fig. 14. Results of draping a 1200 × 350 mm ply onto a double curved low curvature surface. (a) After the automatic drape the outline of the ply is offset approximately 5 mm. (b) Draping result with only minor air intrusions, which are fixed during vacuum debulk.

shows the alignment of the ply. Its offset is approximately 5 mm. Figure 14b shows the case example of the overall draping result. It contains minor air intrusions, which can be handled by the sub-sequential vacuum debulking process.

10. Conclusion and Future Work

The main contribution of this paper is the novel design and implementation of a new system for automatic robotic draping of prepreg fiber fabrics onto double curved molds. The various hardware and software components have been described in detail, and the integration issue of these components to the complete solution has been discussed. Finally, Proof-of-Concept experiments are conducted on the experimental physical model of the system, and the results are presented.

The system has been tested on an actual, double curved part from an aircraft. The prepreg material used in the experiments had the 4 harness-satin weave pattern, and it was draped onto the mold using two different orientations: $P0 - 90$ and $P45 - 45$. For both orientations, two draping strategies were applied: square weave strategy and skewed wave strategy. The final drape was then inspected for quality, represented by the number of wrinkles and air intrusions. Based on the Proof-Of-Concept experimental results, it can be concluded that the draping strategies have to be adapted to the specific drapes, as the shape of the part and the orientation of fibers in the ply with regard to the mold have an impact on the resulting draping quality. This underlines also the importance of having realistic ply and tool models available for offline planning of suited drape strategies.

Various technologies still need to be developed before the system is ready for production. The current drape tool is a part of the experimental physical model of the system with many 3D printed parts that will be replaced. Moreover, it is desired to simplify the layout of the drape tool, further reduce the weight and study other kinematic structures. In addition, an open/close mechanism for the suction will be installed for each suction cup. Furthermore, the development of the tool state sensor will be finalized, and the two additional sensors, which are the Pick Camera and the drape quality sensor, will be installed and tested.

The models will be improved based on the inputs from the experimental tests with the tool state sensor and by direct tests of the material properties. Furthermore, the current version of applying the wave-like draping strategy is very premature and will be substantially improved and integrated with a learning strategy based on the feedback from the drape quality sensor.

Even though the system is still at an early stage of development, the initial trials have been positive. We therefore convinced that the FlexDraper system as presented constitutes a feasible approach toward fully automated draping of prepreg composite fabrics.

Acknowledgment

The authors would like to thank the Innovation Fund Denmark for their financial support to the FlexDraper project. The financial support from Manufacturing Academy of Denmark and Lockheed Martin is also gratefully acknowledged.

Supplementary Material

To view supplementary material for this article, please visit https://doi.org/10.1017/S0263574720000193.

References

1. D. H. J. Lukaszewicz, C. Ward and K. D. Potter, "The engineering aspects of automated prepreg layup: History, present and future," *Composites Part B* **43**(3), 997–1009 (2012). DOI: 10.1016/j.compositesb.2011.12.003.
2. R. Molfino, M. Zoppi, F. Cepolina, J. Yousef and E. E. Cepolina, "Design of a Hyper-Flexible Cell for Handling 3D Carbon Fiber Fabric," *Proceedings of the 2014 International Conference on Theoretical. Mechanics and Applied Mechanics (TMAM'14)*. Venice, Italy, pp. 165–170. https://pdfs.semanticscholar.org/da24/dbc48c622c8037b3188229bed051bf1734f7.pdf.
3. A. Björnsson and K. Johansen, "Automated material handling in composite manufacturing using pick-and-place systems review," *Robotics Comput. Integr. Manuf.* **51**, 222–229 (2018). DOI: 10.1016/j.rcim.2017.12.003.
4. M. T. Kordi, M. Husing and B. Corves, "Development of a Multifunctional Robot End-Effector System for Automated Manufacture of Textile Preforms," *Proceedings of the 2007 International Conference on Advanced Intelligent Mechatronic*.
5. T. Gerngross and D. Nieberl, "Automated manufacturing of large, three-dimensional CFRP parts from dry textiles," *CEAS Aeronaut J.* **7**, 241–257 (2016). DOI: 10.1007/s13272-016-0184-5.
6. F. Förster, F. Ballier, S. Coutandin, A. Defranceski and J. Fleischer, "Manufacturing of Textile Preforms with an Intelligent Draping and Gripping System," *Proceedings of the 1st Cirp Conference on Composite Materials Parts Manufacturing* (Elsevier) **66**, 39–44 (2017). DOI: 10.1016/j.procir.2017.03.370.

7. G. Gardiner, "Automated preforming: Intelligent automation in pick-and-place systems," *CompositesWorld* (2017); 10/22.
8. G. Gardiner, "Improving one-piece aerostructures by automating preforming," *CompositesWorld* (2017); 1/15.
9. A. Angerer, C. Ehinger, A. Hoffmann, W. Reif and G. Reinhart, "Design of an Automation System for Preforming Processes in Aerospace Industries," *Proceedings of the 2011 IEEE International Conference on Automation Science and Engineering* (IEEE). DOI: 10.1109/CASE.2011.6042411.
10. C. Ehinger and G. Reinhart, "Robot-based automation system for the flexible preforming of single-layer cut-outs in composite industry," *Product. Eng. Res. Dev.* **8**, 559–565 (2014). DOI: 10.1007/s11740-014-0546-y.
11. S. Sterk, "Low Cost Automated Manufacturing of an A320 Slat Fairing," *Proceedings of SAMPE Europe SEICO 2014 - 35th International Technical Conference & Forum*, Paris, France.
12. COALESCE. Cost efficient advanced leading edge structure (2009). http://www.structures.ethz.ch/research/computational-structural-mechanics/COALESCE2.html.
13. K. Campbell, "Tsb Affordable Composites Manufacturing - Grand Challenge," Presentation at ACCIS Annual Conference (2012). http://www.axillium.com/i-composites/sites/default/files/i-Composites%20-%20Picking%20and%20Placing%20of%20Dry%20Fibre.pdf.
14. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "Ros: An Open-Source Robot Operating System," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan (2009). https://pdfs.semanticscholar.org/d45e/aee8b2e047306329e5dbfc954e6dd318ca1e.pdf?_ga=2.240291008.2127394806.1587036108-362228064.1578833333.
15. F. Van Der Ween, "Algorithms for draping fabrics on doubly-curved surfaces," *Int. J. Numer. Methods Eng.* **31**(7), 1415–1426 (1991).
16. T. Liu, A. Bargteil, J. O'Brien and L. Kavan, "Fast simulation of mass-spring systems," *ACM Trans. Graphics* **32**(6), Article 214 (2013). DOI: 10.1145/2508363.2508406.
17. O. W. Nielsen, C. Schlette and H. G. Petersen, "Fast and Simple Model for Free Hanging, Pre-Impregnated Carbon Fibre Material," *Icinco 2018* (2018) pp. 7–14.
18. T. Gereke, O. Döbrich, M. Hübner and C. Cherif, "Experimental and computational composite textile reinforcement forming: A review," *Compos. Part A Appl. Sci. Manuf.* **46**(1), 1–10 (2013). DOI: 10.1016/j.compositesa.2012.10.004.
19. J. Cao, R. Akkerman, P. Boisse, J. Chen, H. S. Cheng, E. F. de Graaf, J. L. Gorczyca, P. Harrison, G. Hivet, J. Launay, W. Lee, L. Liu, S. V. Lomov, A. Long, E. de Luycker, F. Morestin, J. Padvoikis, X. Q. Peng, J. Sherwood, T. Z. Stoilova, X. M. Tao, I. Verpoest, A. Willems, J. Wiggers, T. X. Yu and B. Zhu, "Characterization of mechanical behavior of woven fabrics: experimental methods and benchmark results," *Compos. Part A Appl. Sci. Manuf.* **39**(6), 1037–1053 (2008). DOI: 10.1016/j.compositesa.2008.02.016.
20. C. Krogh, J. A. Glud and J. Jakobsen, "Modeling the robotic manipulation of woven carbon fiber prepreg plies onto double curved molds: A path dependent problem," *J. Compos. Materials* **53**(15), 2149–2164 (2019).
21. R. M. Murray, Z. Li and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. 1 ed. (CRC Press, 1994). ISBN 9780849379819.
22. E. Minguzzi, "Rayleigh's dissipation function at work," *Eur. J. Phys.* **36**(3), 035014 (2015). doi: 10.1088/0143-0807/36/3/035014. 1409.4041.
23. J.E. Marsden and M. West, "Discrete mechanics and variational integrators," *Acta Numerica* **10**, 357–514 (2001). DOI: 10.1017/S096249290100006X.
24. O. A. Bauchau and A. Laulusa, "Review of contemporary approaches for constraint enforcement in multibody systems," *ASME J. Comput. Nonlinear Dyn.* **3**(1), 011005 (2008). DOI: 10.1115/1.2803258.
25. P. Kühmstedt, C. Munckelt, M. Heinze, C. Bräuer-Burchardt and G. Notni, "3D Shape Measurement with Phase Correlation Based Fringe Projection," *Proceedings of SPIE 6616*. DOI: 10.1117/12.726119.
26. E. R. Eiríksson, J. Wilm, D. B. Pedersen and H. Aanæs, "Precision and Accuracy Parameters in Structured Light 3-d Scanning," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-5/W8 (2016) pp. 7–15. DOI: 10.5194/isprs-archives-XL-5-W8-7-2016.
27. C. Reich, R. Ritter and J. Thesing, "White Light Heterodyne Principle for 3D-measurement," *SPIE 3100, Sensors, Sensor Systems, and Sensor Data Processing*, vol. 3100 (1997) pp. 236–244.
28. W. Palfinger, S. Thumfart and C. Eitzinger, "Photometric Stereo on Carbon Fiber Surfaces," *35th Workshop of the Austrian Association for Pattern Recognition.*

CONTRIBUTION C

Video Frame Interpolation via Cyclic Fine-Tuning and Asymmetric Reverse Flow

Video Frame Interpolation via Cyclic Fine-Tuning and Asymmetric Reverse Flow

Morten Hannemose[1], Janus Nørtoft Jensen[1], Gudmundur Einarsson[1], Jakob Wilm[2], Anders Bjorholm Dahl[1], and Jeppe Revall Frisvad[1]

[1] DTU Compute, Technical University of Denmark
[2] SDU Robotics, University of Southern Denmark

Abstract. The objective in video frame interpolation is to predict additional in-between frames in a video while retaining natural motion and good visual quality. In this work, we use a convolutional neural network (CNN) that takes two frames as input and predicts two optical flows with pixelwise weights. The flows are from an unknown in-between frame to the input frames. The input frames are warped with the predicted flows, multiplied by the predicted weights, and added to form the in-between frame. We also propose a new strategy to improve the performance of video frame interpolation models: we reconstruct the original frames using the learned model by reusing the predicted frames as input for the model. This is used during inference to fine-tune the model so that it predicts the best possible frames. Our model outperforms the publicly available state-of-the-art methods on multiple datasets.

Keywords: slow motion · video frame interpolation · convolutional neural networks.

1 Introduction

Video frame interpolation, also known as inbetweening, is the process of generating intermediate frames between two consecutive frames in a video sequence. This is an important technique in computer animation [19], where artists draw keyframes and lets software interpolate between them. With the advent of high frame rate displays that need to display videos recorded at lower frame rates, inbetweening has become important in order to perform frame rate up-conversion [2]. Computer animation research [9, 19] indicates that good inbetweening cannot be obtained based on linear motion, as objects often deform and follow nonlinear paths between frames. In an early paper, Catmull [3] interestingly argues that inbetweening is "akin to difficult artificial intelligence problems" in that it must be able understand the content of the images in order to accurately handle e.g. occlusions. Applying learning-based methods to the problem of inbetweening thus seems an interesting line of investigation.

Some of the first work on video frame interpolation using CNNs was presented by Niklaus et al. [17, 18]. Their approach relies on estimating kernels to jointly represent motion and interpolate intermediate frames. Concurrently, Liu

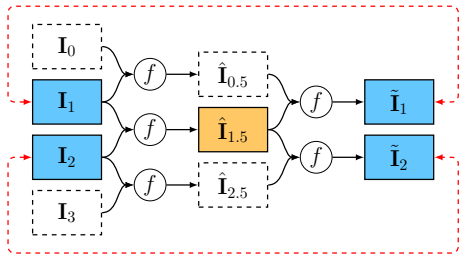


Fig. 1. Diagram illustrating the cyclic fine-tuning process when predicting frame $\hat{\mathbf{I}}_{1.5}$. The model is first applied in a pairwise manner on the four input frames $\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2$, and $\mathbf{I}_3$, then on the results $\hat{\mathbf{I}}_{0.5}$,$\hat{\mathbf{I}}_{1.5}$, and $\hat{\mathbf{I}}_{2.5}$. The results of the second iteration, $\tilde{\mathbf{I}}_1$ and $\tilde{\mathbf{I}}_2$, are then compared with the input frames and the weights of the network are updated. This process optimizes our model specifically to be good at interpolating frame $\hat{\mathbf{I}}_{1.5}$.

et al. [11] and Jiang et al. [6] used neural networks to predict optical flow and used it to warp the input images followed by a linear blending.

Our contribution is twofold. Firstly, we propose a CNN architecture that directly estimates asymmetric optical flows and weights from an unknown intermediate frame to two input frames. We use this to interpolate the frame in-between. Existing techniques either assume that this flow is symmetric or use a symmetric approximation followed by a refinement step [6, 11, 16]. For non-linear motion, this assumption does not hold, and we document the effect of relaxing it. Secondly, we propose a new strategy for fine-tuning a network for each specific frame in a video. We rely on the fact that interpolated frames can be used to estimate the original frames by applying the method again with the in-between frames as input. The similarity of reconstructed and original frames can be considered a proxy for the quality of the interpolated frames. For each frame we predict, the model is fine-tuned in this manner using the surrounding frames in the video, see Figure 1. This concept is not restricted to our method and could be applied to other methods as well.

2 Related work

Video frame interpolation is usually done in two steps: motion estimation followed by frame synthesis. Motion estimation is often performed using optical flow [1, 4, 25], and optical flow algorithms have used interpolation error as an error metric [1, 12, 23]. Frame synthesis can then be done via e.g. bilinear interpolation and occlusion reasoning using simple hole filling. Other methods use phase decompositions of the input frames to predict the phase decomposition of the intermediate frame and invert this for frame generation [14, 15], or they use local per pixel convolution kernels on the input frames to both represent motion and synthesize new frames [17, 18]. Mahajan et al. [13] determine where each pixel in an intermediate frame comes from in the surrounding input frames by

solving an expensive optimization problem. Our method is similar but replaces the optimization step with a learned neural network.

The advent of CNNs has prompted several new learning based approaches. Liu et al. [11] train a CNN to predict a symmetrical optical flow from the intermediate frame to the surrounding frames. They synthesize the target frame by interpolating the values in the input frames. Niklaus et al. [17] train a network to output local 38x38 convolution kernels for each pixel to be applied on the input images. In [18], they are able to improve this to 51x51 kernels. However, their representation is still limited to motions within this range. Jiang et al. [6] first predict bidirectional optical flows between two input frames. They combine these to get a symmetric approximation of the flows from an intermediate frame to the input frames, which is then refined in a separate step. Our method, in contrast, directly predicts the final flows to the input frames without the need for an intermediate step. Niklaus et al. [16] also initially predict bidirectional flows between the input frames and extract context maps for the images. They warp the input images and context maps to the intermediate time step using the predicted flows. Another network blends these to get the intermediate frame.

Liu et al. [10] propose a new loss term, which they call cycle consistency loss. This is a loss based on how well the output frames of a model can reconstruct the input frames. They retrain the model from [11] with this and show state-of-the-art results. We use this loss term and show how it can be used during inference to improve results. Meyer et al. [14] estimate the phase of an intermediate frame from the phases of two input frames represented by steerable pyramid filters. They invert the decomposition to reconstruct the image. This method alleviates some of the limitations of optical flow, which are also limitations of our method: sudden light changes, transparency and motion blur, for example. However, their results have a lower level of detail.

3 Method

Given a video containing the image sequence $\mathbf{I}_0, \mathbf{I}_1, \cdots, \mathbf{I}_n$, we are interested in computing additional images that can be inserted in the original sequence to increase the frame rate, while keeping good visual quality in the video. Our method doubles the frame rate, which allows for the retrieval of approximately any in-between frame by recursive application of the method. This means that we need to compute estimates of $\mathbf{I}_{0.5}, \mathbf{I}_{1.5}, \cdots, \mathbf{I}_{n-0.5}$, such that the final sequence would be:

$$\mathbf{I}_0, \mathbf{I}_{0.5}, \mathbf{I}_1, \cdots, \mathbf{I}_{n-0.5}, \mathbf{I}_n.$$

We simplify the problem by only looking at interpolating a single frame $\mathbf{I}_1$, that is located temporally between two neighboring frames $\mathbf{I}_0$ and $\mathbf{I}_2$. If we know the optical flows from the missing frame to each of these and denote them as $\mathbf{F}_{1\to0}$ and $\mathbf{F}_{1\to2}$, we can compute an estimate of the missing frame by

$$\hat{\mathbf{I}}_1 = \mathbf{W}_0 \mathcal{W}(\mathbf{F}_{1\to0}, \mathbf{I}_0) + \mathbf{W}_2 \mathcal{W}(\mathbf{F}_{1\to2}, \mathbf{I}_2), \tag{1}$$

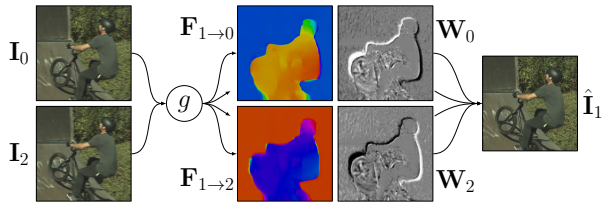


Fig. 2. Illustration of the frame interpolation process with g from Equation (2). From left to right: Input frames, predicted flows, weights and final interpolated frame.

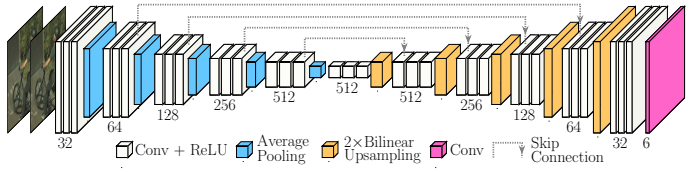


Fig. 3. The architecture of our network. Input is two color images $\mathbf{I}_0$ and $\mathbf{I}_2$ and output is optical flows $\mathbf{F}_{1\to0}, \mathbf{F}_{1\to2}$, and weights $\mathbf{W}_0, \mathbf{W}_2$. Convolutions are 3×3 and average pooling is 2×2 with a stride of 2. Skip connections are implemented by adding the output of the layer that arrows emerge from to the output of the layers they point to.

where $\mathcal{W}(\cdot, \cdot)$ is the backward warping function that follows the vector to the input frame and samples a value with bilinear interpolation. $\mathbf{W}_0$ and $\mathbf{W}_2$ are weights for each pixel describing how much of each of the neighboring frames should contribute to the middle frame. The weights are used for handling occlusions. Examples of flows and weights can be seen in Figure 2. We train a CNN g with a U-Net [20] style architecture, illustrated in Figure 3. The network takes two images as input and predicts the flows and pixel-wise weights

$$g(\mathbf{I}_0, \mathbf{I}_2) \to \mathbf{F}_{1\to0}, \mathbf{F}_{1\to2}, \mathbf{W}_0, \mathbf{W}_2. \tag{2}$$

Our architecture uses five 2×2 average pooling layers with stride 2 for the encoding and five bilinear upsampling layers to upscale the layers with a factor 2 in the decoding. We use four skip connections (addition) between layers in the encoder and decoder. It should be noted that our network is fully convolutional, which implies that it works on images of any size, where both dimensions are a multiple of 32. If this is not the case, we pad the image with boundary reflections.

Our model for frame interpolation is obtained by combining Equations (1) and (2) into

$$f(\mathbf{I}_0, \mathbf{I}_2) = \hat{\mathbf{I}}_1, \tag{3}$$

where $\hat{\mathbf{I}}_1$ is the estimated image. The model is depicted in Figure 2. All components of f are differentiable, which means that our model is end-to-end trainable. It is easy to get data in the form of triplets $(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2)$ by taking frames from videos that we use as training data for our model.

3.1 Loss-functions

We employ a number of loss functions to train our network. All of our loss functions are given for a single triplet $(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2)$, and the loss for a minibatch of triplets is simply the mean of the loss for each triplet. In the following paragraphs, we define the different loss functions that we employ.

Reconstruction loss models how well the network has reconstructed the missing frame:

$$\mathcal{L}_1 = \left|\left|\mathbf{I}_1 - \hat{\mathbf{I}}\right|\right|_1 . \tag{4}$$

Bidirectional reconstruction loss models how well each of our predicted optical flows is able to reconstruct the missing frame on its own:

$$\mathcal{L}_b = ||\mathbf{I}_1 - \mathcal{W}(\mathbf{F}_{1\to 0}, \mathbf{I}_0)||_1 + ||\mathbf{I}_1 - \mathcal{W}(\mathbf{F}_{1\to 2}, \mathbf{I}_2)||_1 . \tag{5}$$

This has similarities to the work of Jiang et al. [6] but differs since the flow is estimated from the missing frame to the existing frames, and not between the existing frames.

Feature loss is introduced as an approximation of the perceptual similarity by comparing feature representation of the images from a pre-trained deep neural network [7]. Let ϕ be the output of `relu4_4` from VGG19 [21], then

$$\mathcal{L}_f = \left|\left|\phi(\mathbf{I}_1) - \phi(\hat{\mathbf{I}}_1)\right|\right|_2^2 . \tag{6}$$

Smoothness loss is a penalty on the absolute difference between neighboring pixels in the flow field. This encourages a smoother optical flow [6,11]:

$$\mathcal{L}_s = ||\nabla \mathbf{F}_{1\to 0}||_1 + ||\nabla \mathbf{F}_{1\to 2}||_1 , \tag{7}$$

where $||\nabla \mathbf{F}||_1$ is the sum of the anisotropic total variation for each (x, y) component in the optical flow $\mathbf{F}$. For ease of notation, we introduce a linear combination of Equations (4) to (7):

$$\mathcal{L}_r(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2) = \lambda_1 \mathcal{L}_1 + \lambda_b \mathcal{L}_b + \lambda_f \mathcal{L}_f + \lambda_s \mathcal{L}_s . \tag{8}$$

Note that we explicitly express this as a function of a triplet. When this triplet is the three input images, we define

$$\mathcal{L}_\alpha = \mathcal{L}_r(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2) . \tag{9}$$

Similarly, for ease of notation, let the bidirectional loss from Equation (5) be a function

$$\mathcal{L}_B(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2, \mathbf{F}_{1\to 0}, \mathbf{F}_{1\to 2}) = \mathcal{L}_b \tag{10}$$

where, in this case, $\mathbf{F}_{1\to 0}$ and $\mathbf{F}_{1\to 2}$ are the flows predicted by the network.

Pyramid loss is a sum of bidirectional losses for downscaled versions of images and flow maps:

$$\mathcal{L}_p = \sum_{l=1}^{l=4} 4^l \mathcal{L}_B\Big(A_l(\mathbf{I}_0), A_l(\mathbf{I}_1), A_l(\mathbf{I}_2), A_l(\mathbf{F}_{1\to 0}), A_l(\mathbf{F}_{1\to 2})\Big), \tag{11}$$

where A_l is the $2^l \times 2^l$ average pooling operator with stride 2^l.

Cyclic loss functions. We can apply our model recursively to get another estimate of $\mathbf{I}_1$, namely

$$\tilde{\mathbf{I}}_1 = f\left(\hat{\mathbf{I}}_{0.5}, \hat{\mathbf{I}}_{1.5}\right) = f\Big(f\left(\mathbf{I}_0, \mathbf{I}_1\right), f\left(\mathbf{I}_1, \mathbf{I}_2\right)\Big). \tag{12}$$

Cyclic loss is introduced to ensure that outputs from the model work well as inputs to the model [10]. It is defined by

$$\mathcal{L}_c = \mathcal{L}_r\left(\hat{\mathbf{I}}_{0.5}, \mathbf{I}_1, \hat{\mathbf{I}}_{1.5}\right). \tag{13}$$

Motion loss is introduced in order to get extra supervision on the optical flow and utilizes the recursive nature of our network.

$$\mathcal{L}_m = ||\mathbf{F}_{1\to 0} - 2\mathbf{F}_{1\to 0.5}||_2^2 + ||\mathbf{F}_{1\to 2} - 2\mathbf{F}_{1\to 1.5}||_2^2 \tag{14}$$

This is introduced as self-supervision of the optical flow, under the assumption that the flow $\mathbf{F}_{1\to 0}$ is approximately twice that of $\mathbf{F}_{1\to 0.5}$ and similarly for $\mathbf{F}_{1\to 2}$ and $\mathbf{F}_{1\to 1.5}$, and assuming that the flow is easier to learn for shorter time steps.

3.2 Training

We train our network using the assembled loss function

$$\mathcal{L} = \mathcal{L}_\alpha + \mathcal{L}_c + \lambda_m \mathcal{L}_m, \tag{15}$$

where $\mathcal{L}_\alpha$, $\mathcal{L}_c$ and $\mathcal{L}_m$ are as defined in Equations (9), (13) and (14) with $\lambda_r = 1$, $\lambda_b = 1$, $\lambda_f = 8/3$, $\lambda_s = 10/3$ and $\lambda_m = 1/192$. The values have been selected based on the performance on a validation set.

We train our network using the Adam optimizer [8] with default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and with a minibatch size of 64.

Inspired by Liu et al. [10], we first train the network using only $\mathcal{L}_\alpha$. This is done on patches of size 128×128 for 150 epochs with a learning rate of 10^{-5}, followed by 50 epochs with a learning rate of 10^{-6}. We then train with the full loss function $\mathcal{L}$ on patches of size 256×256 for 35 epochs with a learning rate of 10^{-5} followed by 30 epochs with a learning rate of 10^{-6}. We did not use batch-normalization, as it decreased performance on our validation set. Due to the presence of the cyclic loss functions, four forward and backward passes are needed for each minibatch during the training with the full loss function.

Training data. We train our network on triplets of patches extracted from consecutive video frames. For our training data, we downloaded 1500 videos in 4k from youtube.com/4k and resized them to 1920×1080. For every four frames in the video not containing a scene cut, we chose a random 320×320 patch and cropped it from the first three frames. If any of these patches were too similar or if the mean absolute differences from the middle patch to the previous and following patches were too big, or too dissimilar, they were discarded to avoid patches that either had little motion or did not include the same object. Our final training set consists of 476,160 triplets.

Data augmentation. The data is augmented while we train the network by cropping a random patch from the 320×320 data with size as specified in the training details in Section 3.2. In this way, we use the training data more effectively. We also add a random translation to the flow between the patches, by offsetting the crop to the first and third patch while not moving the center patch [18]. This offset is ± 5 pixels in each direction. Furthermore, we also performed random horizontal flips and swapped the temporal order of the triplet.

3.3 Cyclic fine-tuning (CFT)

We introduce the concept of fine-tuning the model during inference for each frame that we want to interpolate and refer to this as cyclic fine-tuning (CFT). Recall that the cyclic loss $\mathcal{L}_c$ measures how well the predicted frames are able to reconstruct the original frames. This gives an indication of the quality of the interpolated frames. The idea of CFT is to exploit this property at inference time to improve interpolation quality. We do this by extending the cyclic loss to ± 2 frames around the desired frame and fine-tuning the network using these images only.

When interpolating frame $\mathbf{I}_{1.5}$, we would use surrounding frames $\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2$, and $\mathbf{I}_3$ to compute $\hat{\mathbf{I}}_{0.5}, \hat{\mathbf{I}}_{1.5}$, and $\hat{\mathbf{I}}_{2.5}$, which are then used to compute $\tilde{\mathbf{I}}_1$ and $\tilde{\mathbf{I}}_2$ as illustrated in Figure 1 on page 2. Note that the desired interpolated frame $\hat{\mathbf{I}}_{1.5}$ is used in the computation of both of the original frames. Therefore by fine-tuning of the network to improve the quality of the reconstructed original frames, we are improving the quality of the desired intermediate frame indirectly.

Specifically, we minimize the loss for each of the two triplets $(\hat{\mathbf{I}}_{0.5}, \mathbf{I}_1, \hat{\mathbf{I}}_{1.5})$ and $(\hat{\mathbf{I}}_{1.5}, \mathbf{I}_2, \hat{\mathbf{I}}_{2.5})$ with the loss for each triplet given by

$$\mathcal{L}_{CFT} = \mathcal{L}_c + \lambda_p \mathcal{L}_p, \tag{16}$$

where $\lambda_p = 10$, and $\mathcal{L}_p$ is the pyramid loss described in Section 3.1. In order for the model to be presented with slightly different samples, we only do this fine-tuning on patches of 256×256 with flow augmentation as described in the previous section. For computational efficiency, we only do this for 50 patch-triplets for each interpolated frame.

More than ± 2 frames can be applied for fine-tuning, however, we found that this did not increase performance. This fine-tuning process is not limited to our model and can be applied to any frame interpolation model taking two images as input and outputting one image.

4 Experiments

We evaluate variations of our method on three diverse datasets: UCF101 [22], SlowFlow [5] and See You Again [18]. These have previously been used for frame interpolation [6, 11, 18]. UCF101 contains different image sequences of a variety of actions, and we evaluate our method on the same frames as Liu et al. [11], but

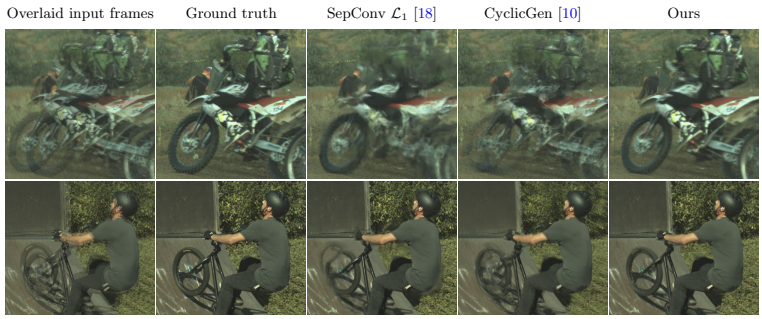


Fig. 4. Qualitative examples on the SlowFlow dataset. Images shown are representative crops taken from the full images. Note that our method performs much better for large motions. The motion of the dirt bike is approximately 53 pixels, and the bike tire has a motion of 34 pixels.

Table 1. Overview of the datasets we used for evaluation.

Dataset	Number of sequences	Resolution	Number of frames		Avg. sequence length
			Interpolated	Total	
SlowFlow [5]	34	$1280 \times \{1024, 720\}$	17,871	20,458	602
See You Again	117	1920×1080	2,503	5,355	46
UCF101 [22]	379	256×256	379	1,137	3

did not use any motion masks as we are interested in performing equally well over the entire frame. SlowFlow is a high-fps dataset that we include to showcase our performance when predicting multiple in-between frames. For this dataset, we have only used every eighth frame as input and predicted the remaining seven in-between in a recursive manner. All frames in the dataset have been debayered, resized to 1280 pixels on the long edge and gamma corrected with a gamma value of 2.2. See You Again is a high-resolution music video, where we predict the even-numbered frames using the odd-numbered frames. Furthermore, we have divided it into sequences by removing scene changes. A summary of the datasets is shown in Table 1.

We have compared our method with multiple state-of-the-art methods [6,10, 11,18] which either have publicly available code and/or published their predicted frames. For the comparison with SepConv [18], we use the $\mathcal{L}_1$ version of their network for which they report their best quantitative results. For each evaluation, we report the Peak Signal to Noise Ratio (PSNR) and the Structural Similarity Index (SSIM) [24].

Table 2. Interpolation results on SlowFlow, See You Again and UCF101. PSNR, SSIM: higher is better. Our baseline is our model trained only with $\mathcal{L}_1 + \mathcal{L}_b + \mathcal{L}_s$ and constrained to symmetric flow. Elements are added to the model cumulatively. Larger patches means training on 256×256 patches. Bold numbers signify that a method performs significantly better than the rest on that task with $p < 0.02$.

	SlowFlow		See You Again		UCF101	
Method	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
DVF [11]	-	-	-	-	34.12	0.941
SuperSloMo [6]	-	-	-	-	34.75	0.947
SepConv $\mathcal{L}_1$ [18]	34.03	0.899	42.49	0.983	34.78	0.947
CyclicGen [10]	31.33	0.839	41.28	0.975	**35.11**	**0.949**
Our baseline	34.28	0.903	42.50	0.984	34.39	0.946
+ asymmetric flow	34.33	0.904	42.54	0.985	34.40	0.946
+ feature loss	34.29	0.901	42.62	0.984	34.60	0.948
+ cyclic loss	34.33	0.900	42.73	0.984	34.62	0.947
+ motion loss	34.31	0.900	42.74	0.984	34.61	0.948
+ larger patches	34.60	0.907	43.14	0.986	34.69	0.948
+ CFT	**34.91**	**0.912**	**43.21**	**0.986**	34.94	**0.949**

Comparison with state-of-the-art. Table 2 shows that our best method, with or without CFT, clearly outperforms the other methods on SlowFlow and See you Again, which is also reflected in Figure 4. On UCF101 our best method performs better than all other methods except CyclicGen, where our best method has the same SSIM but lower PSNR. We suspect this is partly due to the fact that our CFT does not have access to ± 2 frames in all sequences. For some of the sequences, we had to use $-1, +3$ as the intermediate frame was at the beginning of the sequence. Visually, our method produces better results as seen in Figure 5. We note that CyclicGen is trained on UCF101, and their much worse performance on the two other datasets could indicate overfitting.

Effect of various model configurations. Table 2 reveals that an asymmetric flow around the interpolated frame slightly improves performance on all three datasets as compared with enforcing a symmetric flow. There is no clear change in performance when we add feature loss, cyclic loss and motion loss.

For all three datasets, performance improves when we train on larger image patches. Using larger patches allows for the network to learn larger flows and the performance improvement is correspondingly seen most clearly in SlowFlow and See You Again which, as compared with UCF101, have much larger images with larger motions. We see a big performance improvement when cyclic fine-tuning is added, which is also clearly visible in Figure 6.

Discussion. Adding CFT to our model increases the run-time of our method by approximately 6.5 seconds per frame pair. This is not dependent on image

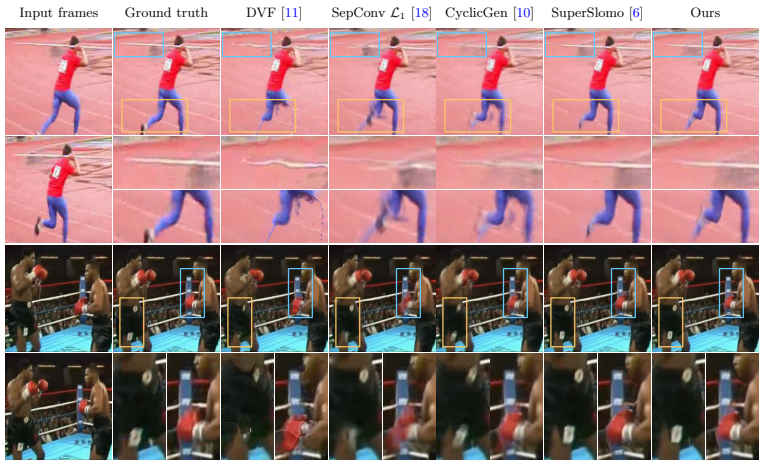


Fig. 5. Qualitative comparison for two sequences from UCF101. Top: Our method produces the least distorted javelin and retains the detailed lines on the track. All methods perform inaccurately on the leg, however, SuperSlomo and our method are the most visually plausible. Bottom: Our method and SuperSlomo create accurate white squares on the shorts (left box). Our method also produces the least distorted ropes and white squares on the corner post, while creating foreground similar to the ground truth (right box).

size, as we only fine-tune on 256×256 patches for 50 iterations per frame pair. For reference, our method takes 0.08 seconds without CFT to interpolate a 1920×1080 image on an NVIDIA GTX 1080 TI. It should be noted that CFT is only necessary to do once per frame pair in the original video, and thus there is no extra overhead when computing multiple in-between frames.

Training for more than 50 iterations does not necessarily ensure better results, as we can only optimize a proxy of the interpolation quality. The best number of iterations remains to be determined, but it is certainly dependent on the quality of the pre-training, the training parameters, and the specific video.

As of now, CFT should only be used if the target is purely interpolation quality. Improving the speed of CFT is a topic worthy of further investigation. Possible solutions of achieving similar results include training a network to learn the result of CFT, or training a network to predict the necessary weight changes.

5 Conclusion

We have proposed a CNN for video frame interpolation that predicts two optical flows with pixelwise weights from an unknown intermediate frame to the frames

Fig. 6. Representative example of how the cyclic fine-tuning improves the interpolated frame. It can be seen that the small misalignment of the tire and yellow "41" is corrected by the cyclic fine-tuning.

before and after. The flows are used to warp the input frames to the intermediate time step. These warped frames are then linearly combined using the weights to obtain the intermediate frame. We have trained our CNN using 1500 high-quality videos and shown that it performs better than or comparably to state-of-the-art methods across three different datasets. Furthermore, we have proposed a new strategy for fine-tuning frame interpolation methods for each specific frame at evaluation time. When used with our model, we have shown that it improves both the quantitative and visual results.

Acknowledgements. We would like to thank Joel Janai for providing us with the SlowFlow data [5].

References

1. Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R.: A database and evaluation methodology for optical flow. International Journal of Computer Vision **92**(1), 1–31 (2011)
2. Castagno, R., Haavisto, P., Ramponi, G.: A method for motion adaptive frame rate up-conversion. IEEE Transactions on Circuits and Systems for Video Technology **6**(5), 436–446 (October 1996)
3. Catmull, E.: The problems of computer-assisted animation. Computer Graphics (SIGGRAPH '78) **12**(3), 348–353 (August 1978)
4. Herbst, E., Seitz, S., Baker, S.: Occlusion reasoning for temporal interpolation using optical flow. Tech. rep., Microsoft Research (August 2009)
5. Janai, J., Güney, F., Wulff, J., Black, M., Geiger, A.: Slow Flow: Exploiting high-speed cameras for accurate and diverse optical flow reference data. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 1406–1416 (2017)
6. Jiang, H., Sun, D., Jampani, V., Yang, M.H., Learned-Miller, E., Kautz, J.: Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 9000–9008 (2018)

7. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision (ECCV). pp. 694–711 (2016)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
9. Lasseter, J.: Principles of traditional animation applied to 3D computer animation. Compututer Graphics (SIGGRAPH '87) **21**(4), 35–44 (August 1987)
10. Liu, Y.L., Liao, Y.T., Lin, Y.Y., Chuang, Y.Y.: Deep video frame interpolation using cyclic frame generation. In: AAAI Conference on Artificial Intelligence (2019)
11. Liu, Z., Yeh, R.A., Tang, X., Liu, Y., Agarwala, A.: Video frame synthesis using deep voxel flow. In: International Conference on Computer Vision. pp. 4463–4471 (2017)
12. Long, G., Kneip, L., Alvarez, J.M., Li, H., Zhang, X., Yu, Q.: Learning image matching by simply watching video. In: ECCV. pp. 434–450. Springer, Cham (2016)
13. Mahajan, D., Huang, F.C., Matusik, W., Ramamoorthi, R., Belhumeur, P.: Moving gradients: A path-based method for plausible image interpolation. ACM Transactions on Graphics **28**(3), 42:1–42:11 (July 2009)
14. Meyer, S., Djelouah, A., McWilliams, B., Sorkine-Hornung, A., Gross, M., Schroers, C.: Phasenet for video frame interpolation. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 498–507 (2018)
15. Meyer, S., Wang, O., Zimmer, H., Grosse, M., Sorkine-Hornung, A.: Phase-based frame interpolation for video. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1410–1418 (2015)
16. Niklaus, S., Liu, F.: Context-aware synthesis for video frame interpolation. In: Conference on Computer Vision and Pattern Recognition. pp. 1701–1710 (2018)
17. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive convolution. In: Conference on Computer Vision and Pattern Recognition. pp. 670–679 (2017)
18. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive separable convolution. In: International Conference on Computer Vision. pp. 261–270 (2017)
19. Reeves, W.T.: Inbetweening for computer animation utilizing moving point constraints. Computer Graphics (SIGGRAPH '81) **15**(3), 263–269 (August 1981)
20. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
22. Soomro, K., Zamir, A.R., Shah, M., Soomro, K., Zamir, A.R., Shah, M.: UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR **abs/1212.0402** (2012)
23. Szeliski, R.: Prediction error as a quality metric for motion and stereo. In: IEEE International Conference on Computer Vision (ICCV). vol. 2, pp. 781–788 (1999)
24. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004)
25. Werlberger, M., Pock, T., Unger, M., Bischof, H.: Optical flow guided TV-L^1 video interpolation and restoration. In: Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR). pp. 273–286. Springer (2011)

CONTRIBUTION D

Generating Spatial Attention Cues via Illusory Motion

Generating Spatial Attention Cues via Illusory Motion

Janus Nørtoft Jensen[1*] jnje@dtu.dk
Morten Hannemose[1*] mohan@dtu.dk
Jakob Wilm[2] jaw@mmmi.sdu.dk
Anders Bjorholm Dahl[1] abda@dtu.dk
Jeppe Revall Frisvad[1] jerf@dtu.dk
Serge Belongie[3] sjb344@cornell.edu

[1]Technical University of Denmark [2]University of Southern Denmark [3]Cornell Tech

Abstract

For many applications in augmented reality (AR), the user has a much more enjoyable experience if the AR system is able to properly guide the user's attention. In this extended abstract, we explain how to create patterns of light that when projected onto an object are perceived as if the object itself is moving. This can be used as a spatial attention cue. We accomplish this with a calibrated projector-camera setup to synthesize an image from the projector's point of view. This image is filtered to create local phase changes that are then projected back onto the object and perceived as motion. Our method will be shown as a live demonstration at the CV4AR/VR workshop at CVPR 2019.

(a) Left camera (b) Projector (c) Right camera

Figure 1. Example of a synthesized image from the projector's point of view (b), along with the camera images the intensities are sampled from (a, c).

1. Introduction

Visual attention is important as it affects our performance in many visual tasks [2]. To successfully accomplish tasks such as visual search or tele-assistance we need effective spatial attention cues to attract the attention of a user. What constitutes an effective cue is in part determined by the task at hand. A very obvious cue, such as a big bouncing red arrow, might be the best, if the purpose is to alert the user of possible danger. In many other situations, however, more subtle cues might be preferred. This could be in the setting of an escape room game where we would guide players towards the next clue if they are stuck. A very obvious cue could possibly ruin the fun of the game, however a subtle one, which would take longer to notice, could be useful.

As described by Carrasco and Barbot [2], our attention is involuntarily captured by sudden changes in the environment. We seek to exploit this effect by creating apparent motion through light projection. While the human ability to detect motion is not better in the peripheral vision [10], the speed of visual processing does increase in the peripheral vision [3]. When something moves differently compared to the movement of the observer, it becomes a powerful cue to attract attention, especially in the periphery of the visual field [12]. This is referred to as a relative-motion cue. Our idea is to guide attention by creating apparent relative-motion cues in a scene by means of a light projector that modifies the appearance of a physical object to make it look as if it were moving, thereby creating illusory motion.

2. Projecting Illusory Motion

Prior work in guiding visual attention has mainly been focused on head-mounted displays and images displayed on a screen. The spatial attention cues used include flickering [16, 17], blurring [8], and color manipulation [9]. Spatial projection has been shown to be more effective than head-mounted displays in providing spatial instructions in assembly tasks [1]. Research in attention cues for spatial projection is however limited. Some use laser projection [14, 7] as a simple cue. Similar to our approach Taki-

*These authors contributed equally to this work.

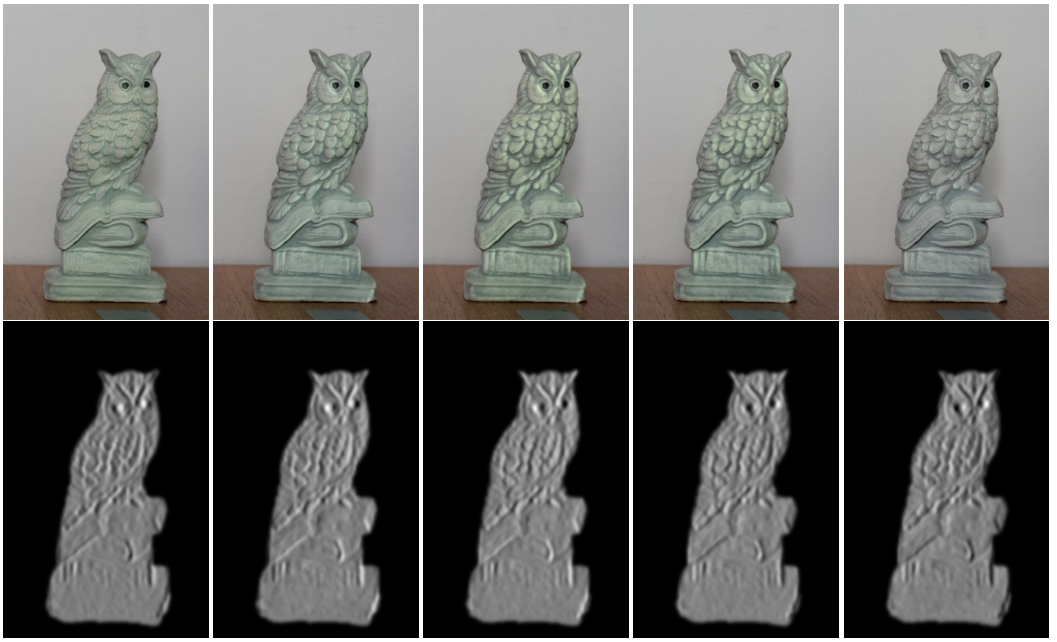


Figure 2. Five frames sampled from the continuous loop of motion our method produces. Top row: Picture of object taken with camera. Bottom row: Image projected by projector to create corresponding picture. Videos showing the effect can be seen at http://people.compute.dtu.dk/jnje/illusory-motion.

moto et al. [15] uses a calibrated projector-camera setup. They modulate color information recorded with the camera and project the result back onto the object. For humans, the sensitivity to color variations declines faster compared with the sensitivity to luminance [6]. It thus seems natural to investigate modifying the luminance instead.

Our approach to create the illusion of motion is based on adapting the work by Freeman et al. [4] to a projector-camera setup. In short, they apply local filters with continuously varying phase over time to the image. This is based on the observation that local phase changes are interpreted as global motion. We synthesize an image from the viewpoint of the projector (Figure 1) and use it as input for their method. The filter response is then projected back onto the object. In Figure 2, examples of the object with the filter response projected onto it are shown along with the images projected.

2.1. Projector-camera calibration

We use two cameras and a projector mounted in a fixed setup, and we model the projector and cameras as pinhole cameras with radial distortion. To calibrate the system we encode the projector's pixel coordinates using structured light [13], detect corners in images of a checkerboard and convert these to the projector's pixel space by local homographies [11]. The projector and cameras are then calibrated with Zhang's method [18].

2.2. Synthesizing image from projector's point of view

To synthesize an image from the projector's point of view we use structured light to create two 3D-scans of the object - one based on each camera. The resulting point clouds and the pixel intensities associated with the points are then projected back to the projector to form the desired image (Figure 1). Creating two separate 3D-scans enables us to scan all points visible in the projector and at least one camera, thereby getting better coverage of the object. After the points from each scan have been projected to the projector's pixel space they are rounded to the nearest integer pixel. Because the projector has a lower resolution than the cameras, each pixel in the projector contains multiple measurements. We compute medians of these to obtain a single value per pixel.

2.3. Creating Illusory Motion

With the image from the projector's point of view, we can directly apply the method of Freeman et al. [4] to this

image. As the filter responses contain both positive and negative values, and the projective setting has the constraint of only adding light, we add a constant value to the filter response to make all values positive. The filtered image is multiplied with a mask to restrict light to the object, and the resulting image is projected onto the object. Examples of the projected image and the resulting effect are in Figure 2.

3. Future work

We can perhaps utilize the 3D information obtained through our process to make the projected patterns less view dependent. Because we know the 3D position of each pixel in addition to its intensity, we can compute a normal at each point. If we, based on these normals, choose a consistent orthonormal basis at each point, such that the basis changes smoothly over the object [5], and assuming that the object is locally planar, we can project the filter onto this plane, and thereby approximate convolution along the surface of the object instead of in image space.

Our current work has been focused on creating the illusion of motion. Further work is needed to determine under which circumstances it is perceived as motion and to determine its effectiveness as a spatial attention cue. Furthermore, examining how it affects the user's experience of interacting with the AR system, and how our cue compares with using alternative spatial attention cues is important to examine. Reasonable variables to look into would be how much light is necessary to make the effect noticeable and how this varies across the visual field.

References

[1] Sebastian Büttner, Markus Funk, Oliver Sand, and Carsten Röcker. Using head-mounted displays and in-situ projection for assistive systems: A comparison. In *Proceedings of the 9th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '16, pages 44:1–44:8. ACM, 2016. 1

[2] Marisa Carrasco and Antoine Barbot. Spatial attention alters visual appearance. *Current opinion in psychology*, 29:56–64, 2019. 1

[3] Marisa Carrasco, Brian McElree, Kristina Denisova, and Anna Marie Giordano. Speed of visual processing increases with eccentricity. *Nature Neuroscience*, 6(7):699, 2003. 1

[4] William T Freeman, Edward H Adelson, and David J Heeger. *Motion without movement*, volume 25. Citeseer, 1991. 2

[5] Jeppe Revall Frisvad. Building an orthonormal basis from a 3d unit vector without normalization. *Journal of Graphics Tools*, 16(3):151–159, 2012. 3

[6] Thorsten Hansen, Lars Pracejus, and Karl R Gegenfurtner. Color perception in the intermediate periphery of the visual field. *Journal of Vision*, 9(4):26–26, 2009. 2

[7] Harald Haraldsson, Doron Tal, Karla Polo-Garcia, and Serge Belongie. Pointar: Augmented reality for tele-assistance. In *CVPR Workshop on Embedded Computer Vision*, Salt Lake City, UT, 2018. 1

[8] Hajime Hata, Hideki Koike, and Yoichi Sato. Visual guidance with unnoticed blur effect. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 28–35. ACM, 2016. 1

[9] Victor A. Mateescu and Ivan V. Bajić. Attention retargeting by color manipulation in images. In *Proceedings of the 1st International Workshop on Perception Inspired Video Processing*, PIVP '14, pages 15–20. ACM, 2014. 1

[10] Suzanne P McKee and Ken Nakayama. The detection of motion in the peripheral visual field. *Vision research*, 24(1):25–32, 1984. 1

[11] Daniel Moreno and Gabriel Taubin. Simple, accurate, and robust projector-camera calibration. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 464–471. IEEE, 2012. 2

[12] Dorothe A Poggel, Hans Strasburger, and Manfred MacKeben. Cueing attention by relative motion in the periphery of the visual field. *Perception*, 36(7):955–970, 2007. 1

[13] Carsten Reich, Reinhold Ritter, and Jan Thesing. White light heterodyne principle for 3d-measurement. In *Sensors, Sensor Systems, and Sensor Data Processing*, volume 3100, pages 236–245. International Society for Optics and Photonics, 1997. 2

[14] Björn Schwerdtfeger, Daniel Pustka, Andreas Hofhauser, and Gudrun Klinker. Using laser projectors for augmented reality. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pages 134–137. ACM, 2008. 1

[15] Hironori Takimoto, Katsumi Yamamoto, Akihiro Kanagawa, Mitsuyoshi Kishihara, and Kensuke Okubo. Guiding visual attention based on visual saliency map with projector-camera system. In *HCI International 2017 – Posters' Extended Abstracts*, pages 383–390. Springer International Publishing, 2017. 2

[16] N. Waldin, M. Waldner, and I. Viola. Flicker observer effect: Guiding attention through high frequency flicker in images. *Computer Graphics Forum*, 36(2):467–476, 2017. 1

[17] M. Waldner, M. Le Muzic, M. Bernhard, W. Purgathofer, and I. Viola. Attractive flicker guiding attention in dynamic narrative visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2456–2465, Dec 2014. 1

[18] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000. 2

CONTRIBUTION E

Alignment of rendered images with photographs for testing appearance models

Alignment of Rendered Images with Photographs for Testing Appearance Models

MORTEN HANNEMOSE[1], MADS EMIL BRIX DOEST[1], ANDREA LUONGO[1], SØREN KIMMER SCHOU GREGERSEN[1], JAKOB WILM[2], AND JEPPE REVALL FRISVAD[1,*]

[1] Technical University of Denmark, Anker Engelunds Vej 1, 2800 Kongens Lyngby, Denmark
[2] University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
[*] Corresponding author: jerf@dtu.dk

Rendering images that match the appearance of real world objects is essential in methods for analysis by synthesis and in appearance prediction. We provide guidelines for step-by-step composition of a model to represent the appearance of a real object. This requires alignment of renderings with a corresponding photograph. We provide a practical method for alignment of a known object and a point-like light source with the configuration observed in a photograph. Our method is based on projective transformation of object edges and silhouette matching in the image plane. Our goal is to support development toward progressive refinement of appearance models through quantitative validation.

1. INTRODUCTION

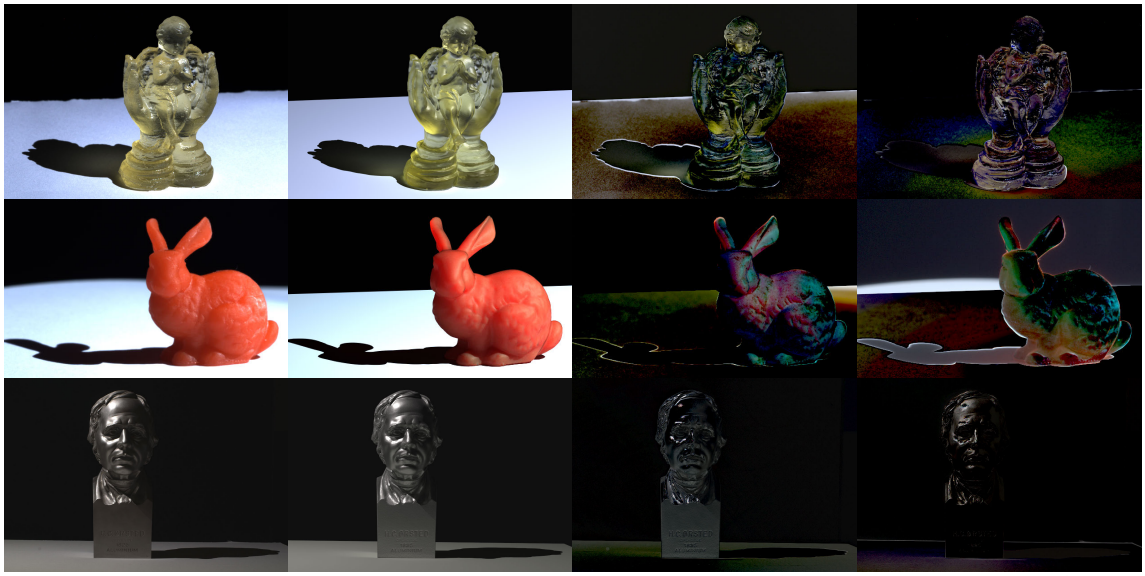
Photorealistic rendering has many applications: product appearance prediction, digital prototyping, inverse rendering to acquire optical properties, 3D soft proofing, etc. In most of these applications, it is important to validate the photorealism of the employed rendering technique. In graphics, side-by-side visual comparison of rendered and photographed images has traditionally been the validation method of choice. Phong [1], for example, qualitatively compared a rendered sphere with a photographed sphere as a final evaluation of his shading and lighting models. Similarly, the Cornell box [2, 3] was presented as a test scene for qualitative comparison of photographs and rendered images. Rushmeier [4] was seemingly the first to discuss quantitative comparison of photographed and rendered images, and Pattanaik et al. [5] then presented a difference image for rendering versus photograph of a version of the Cornell box. Differences in scene geometry and the view-light configuration tend to be the main difficulty in setting up such pixel-by-pixel comparisons [4, 6].

Alignment of rendered and photographed images has reached good precision in controlled setups for geometry and reflectance acquisition [7]. For images captured in less controlled settings, the main difficulties are pose estimation of an object from a given CAD model and light source estimation. These are most often considered two separate problems. For pose estimation, a large dataset is usually employed to train a statistical model [8, 9]. A multitude of techniques exist for light source estimation [10, 11]. However, as we estimate the object pose, we may as well use the pose for light source estimation. Moreover, if using the cast shadow for estimating the light position, we can use it to improve the estimate of the object pose as well.

Inverse rendering [12] enables recovery of both lighting and reflectance properties but often assumes a known object with a known pose. More recent inverse rendering techniques [13, 14] allow pose estimation and deformation of object geometry too. These techniques are based on differentiable rendering, where per pixel derivatives are computed as part of the rendering. While this is a powerful approach for estimating surface displacements and spatially varying reflectance [13], it is also a gradient-based optimization based on per pixel derivatives that requires careful initialization to avoid local minima [14]. In this landscape, we missed a practical method for estimation of both object pose and light source position to enable pixel-by-pixel comparison of a photograph with a rendering. We propose such a method and find that it delivers a good starting point for validating rendering techniques, estimating optical properties, and testing appearance models. In addition, our method is useful for initialization of inverse rendering techniques.

Our outset is a photograph of a single object of known geometry that has been captured with a known camera. We assume that the object is placed on a planar surface and illuminated by a point-like light source. In this scene configuration, we let the user approximately initialize the orientation of the object relative to the planar surface (this could be done using a physics engine), or we use a camera calibration. Our method then estimates the light source position and the camera and object poses. We do this by segmenting the photograph and matching the object and the shadow silhouettes to the silhouettes of the virtual object found by projective transformation of the edges.

We exemplify our method using three scanned objects (see Figure 1): the Stanford bunny [15], an angel figurine, and an aluminium bust of H. C. Ørsted (the scientist who discovered electromagnetism and who was also the first to isolate aluminium). The Stanford bunny was scanned by Greg Turk using a technique for zippering several range scans [16], and we 3D scanned



photograph (x) rendering (y) $(\max(y-x,0))^{1/\gamma}$ $(\max(x-y,0))^{1/\gamma}$

Fig. 1. Pixel-by-pixel comparison of renderings with a photograph enables a detailed investigation of the virtues and deficiencies of an appearance model. Our practical alignment technique is here used for testing different models: rough transparent (top), rough translucent (middle), and metallic (bottom). Difference images were brightened with $\gamma = 2.2$ to clearly visualize the deficiencies.

the other two objects using structured light and stereo vision [17]. We use a translucent 3D printed version of the Stanford bunny, the angel figurine was 3D printed using an almost transparent photopolymer, while we used the aluminium bust as is. This enabled us to take photographs and test appearance models for both subsurface scattering, rough refraction, and metallic rough reflection. We quantitatively test the ability of such models to match the appearance of object samples from the real world (Figure 1), and we suggest improved models based on our findings. Notably, we for the first time integrate rough surface scattering [18] with the directional dipole model for subsurface scattering [19].

2. RELATED WORK

In many side-by-side comparisons of renderings with photographs [1–3, 6, 12, 20], alignment is done manually. This is usually a time-consuming process with an imprecise result. When a comparison is done in the context of 3D acquisition, alignment is given with good accuracy because the object geometry was acquired in a calibrated setup [21, 22]. We are however looking for an alignment method that does not require concurrent 3D scanning of the object. Differentiable rendering [13, 23, 24] is useful for improving a manual alignment, but we would like to avoid the initial manual alignment. We thus use vision techniques to do the alignment and think of our technique as an enabler for an inverse (differentiable) rendering system, which is then free to focus on estimation of parameters not related to alignment.

Our work is related to CAD-based vision [25], where the CAD model of a 3D object is used to recognise the physical version of the object in an image. An important part of such recognition is pose estimation of the object. In a view-based approach [26, 27], multiple views of the object are used for the training of a statistical model to recognise the object and suggest an initial pose. The views can be obtained from photographs captured in a calibrated robot setup [26] or from rendered images of object edges [8, 27, 28]. After estimating an initial pose using a statistical model, the pose is typically refined using iterative shape matching [27, 29]. We combine some of these ideas. Petit et al. [28] suggest a method based on foreground/background segmentation in the case of a moving object. Our method is also based on such a segmentation but for a static object. As in the discussed previous work, we use the edges of the CAD model for pose estimation, specifically the silhouette [8], but we avoid the training of a statistical model based on a dataset with many views.

Iterative methods for pose estimation [29] are good for pose refinement but also prone to local minima if not carefully initialised. An exhaustive search for initial parameters is then needed if we want to avoid the training of a statistical model, but such a search is infeasible for the full 6D pose of an object. An option is then to limit the dimensionality of the search space using invariants [30, 31]. Hu's moment invariants [32] are for example invariant to scale, rotation, and translation. For a 2D shape, this reduces the search space in pose estimation to two angular dimensions [30]. We use this concept for 3D shapes by applying it to the object silhouette found in the image plane.

If one is willing to generate a dataset of object silhouettes (for example) as observed across a view sphere, the pose estimation can be accomplished using shape descriptors even for cluttered scenes [33]. After image segmentation and initial pose estimation, refinement is still required using an iterative method. Several other learning-based techniques are available as well [34–

37]. These all require a large dataset for training and pose refinement after estimating the initial pose. Interestingly, Tekin et al. [38] report a fast learning-based method that does not require pose refinement, but then Li et al. [39] present an iterative learning-based method for pose refinement with improvements over Tekin et al. Peng et al. [9] present an improved method inspired by Tekin and others that indeed seems not to require *a posteriori* pose refinement. This is based on an extensive dataset augmented with 20,000 synthetic images of each object. These learning-based techniques contribute robustness with respect to object detection. This is however not important for our scenes which must, in any case, be uncluttered to enable photorealistic rendering of a corresponding digital scene.

A distinctive advantage of our silhouette matching approach is that we can estimate the light source position too. In this way, we avoid the traditional calibration of a point light by observing highlights in mirroring spheres [7]. Our method employs the shadow silhouette, which we find using Blinn's projection shadows [40]. In some related work [41], the shadow silhouette was detected in an input image with depth information (RGB-D) and used for estimating the position of one or more light sources. However, since we estimate the pose of a known object together with the position of the light, we do not need the depth information. In addition, our treatment of pose and light as a joint problem enables us to refine the estimation of both.

3. ALIGNMENT METHOD

Our method is based on the following input:

- image of an object on a uniform ground plane illuminated by a point-like light source
- segmentation of the image into object, shadow, and background
- 3D model of the object
- camera intrinsics (focal length / camera constant / field of view)
- approximate rotation of the object relative to the ground plane.

Any camera can be used to capture the input image, but we need to know the field of view. If this is not known for a given camera, we can obtain it through camera calibration, but we exclude images captured with an unknown and unavailable camera. In most cases, the segmentation can be accomplished by appropriate thresholds of the input image. In harder cases, such as transparent objects, a good segmentation can be obtained through background subtraction based on one image with and one without the object.

Although we work with one light source per view, we also illuminate a static object with multiple light sources in different positions one at a time. In this case, we use the additional information to improve the object pose and light source positions in a final refinement step.

To obtain object pose and light source position, we project the 3D model into the image plane of the camera and extract the silhouette. Our method aligns the silhouette in this plane with the corresponding silhouette in the input image. We obtain the latter from the segmentation of the input image. The silhouette is a useful representation that enables different comparisons of two silhouettes with options for being either exact or invariant to various measures such as rotation and translation, all while being differentiable.

We define a silhouette as a list of 2D point pairs each representing an edge with a direction. In analogy with a triangle mesh, we can use an indexed edge set to represent a silhouette or a set of lists of 2D points, where the points in each list are connected by edges. This works in general, as we can describe objects with holes (nonzero genus) by having both outer and inner perimeters. An inner perimeter should then be in the opposite direction.

Algorithm 1: Computing a silhouette from edges of a mesh projected to a plane. Each edge exists once in each direction.

$\mathbf{p} := \mathbf{p}_0$ (the leftmost point)
$\mathbf{e}$:= edge from $\mathbf{p}$ with the largest slope
repeat
  from $\mathbf{p}$ follow $\mathbf{e}$ until next intersection, $\mathbf{p}_{\text{new}}$
  $\mathbf{e}_{\text{new}}$:= choose from edges intersecting $\mathbf{p}_{\text{new}}$ such that angle($\mathbf{e}_{\text{new}}, \mathbf{e}$) is minimized
  $\mathbf{p} := \mathbf{p}_{\text{new}}$, $\mathbf{e} := \mathbf{e}_{\text{new}}$
until $\mathbf{p} = \mathbf{p}_0$

A. Silhouette Computation

To compute the silhouette of the real object, we enlarge the segmentation resolution by a factor of two using nearest neighbor sampling. We then use the algorithm by Suzuki and Abe [42, 43] to trace the perimeter of the object. We downscale the traced perimeter and round the coordinates so that they lie exactly on the border between object and background. After tracing the perimeter, we have an optional step to simplify the perimeter to accelerate computations later on. The optional simplification is done using the Ramer-Douglas-Peucker algorithm [44, 45].

We compute silhouettes of the 3D models without rasterization. This makes the silhouettes directly differentiable with respect to scene parameters, which is an advantage in a gradient-based optimization. Given a CAD model, we extract a polygonal mesh and build a half-edge representation of this for easy queries. For a given view matrix, we project the vertex positions to the image plane and connect them using the edges of the mesh polygons. To compute the silhouette, we traverse these edges using Algorithm 1. This algorithm assumes a fully connected object silhouette without holes. Extension to objects with holes is done by restarting the algorithm inside each hole.

For the silhouette computation in Algorithm 1, we find the signed angle between two vectors in 2D using

$$\text{angle}\left(\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) = \text{atan2}\left(a_1 b_2 - a_2 b_1,\ a_1 b_1 + a_2 b_2\right) . \quad \textbf{(1)}$$

The majority of time in Algorithm 1 is spent computing edge intersections [46]. Computational complexity thus depends on the number of edges. We significantly reduce this number by exploiting that an edge can only be part of the silhouette if it is shared by one face facing the camera and another facing away. If we let $\vec{n}_{e,1}$ and $\vec{n}_{e,2}$ denote the 3D surface normals of the faces bordering an edge e, the edge e can only be part of the silhouette if

$$\left(\vec{n}_{e,1} \cdot (\mathbf{v}_e - \mathbf{c})\right)\left(\vec{n}_{e,2} \cdot (\mathbf{v}_e - \mathbf{c})\right) \leq 0 , \quad \textbf{(2)}$$

where $\mathbf{v}_e$ is any point on the edge and $\mathbf{c}$ is the position of the camera. After removing all edges that cannot be part of the silhouette and building a bounding volume hierarchy for the remaining edges, intersection testing is inexpensive just like in ray tracing.

Another way to reduce the computation time of this algorithm is to use a mesh with a lower polygon count for the silhouette while retaining the original mesh for rendering. A modest

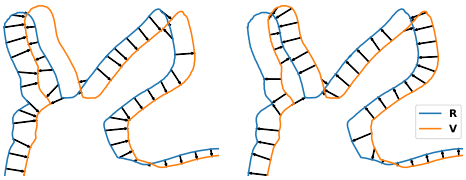


Fig. 2. Illustration of how $\mathrm{sim}(\mathbf{R}, \mathbf{V}, n)$ is computed for a small value of n. The arrows illustrate evaluations of $\mathrm{dist}(\cdot, \cdot)$.

mesh simplification often has a negligible influence on the silhouette.

We have several options when computing silhouette derivatives. For simplicity, we use finite differences. Exact derivatives can be obtained by using dual numbers.

B. Shadow Contours

To include the shadow of an object when considering its silhouette, we assume that the object is placed on a planar surface and use projection shadows [40]. This is also done without rasterization to keep our method valid for the entire image plane. We project the edges of the mesh to the ground plane to generate shadow edges. We then project both object and shadow edges to the image plane of the camera. After this, we use Algorithm 1 to compute the silhouette of the object including its shadow. The number of edges in the shadow that we need to consider is reduced early in the procedure by substituting $\mathbf{c}$ with the light position in Eq. (2).

C. Silhouette Matching

To be able to align silhouettes, we introduce a silhouette similarity metric. We refer to the silhouette of the real object observed by camera c as $\mathbf{R}_c$, and the union of object and shadow silhouettes as $\mathbf{R}_{c,\ell}$, where ℓ is the light source causing the shadow. Equivalently, we define for the virtual object $\mathbf{V}_c$ and $\mathbf{V}_{c,\ell}$. We now let $\mathrm{P}(\mathbf{X}, t)$ denote a parameterization of the silhouette $\mathbf{X}$ with $t \in [0, 1]$. We measure the similarity of two silhouettes by using a function (dist) that finds the shortest distance from a point to a silhouette. Taking n equidistantly sampled points on the silhouettes, we find the shortest distance to the other silhouette and take the sum. The similarity is then computed by

$$\mathrm{sim}(\mathbf{R}, \mathbf{V}, n) = \sum_{i=1}^{n} \left(\mathrm{dist}\left(\mathbf{R}, \mathrm{P}\left(\mathbf{V}, \tfrac{i}{n}\right)\right) + \mathrm{dist}\left(\mathbf{V}, \mathrm{P}\left(\mathbf{R}, \tfrac{i}{n}\right)\right)\right), \tag{3}$$

A visualization of what sim computes is in Figure 2. We can again use a spatial data structure to obtain an efficient implementation of the dist function [47]. Our similarity metric (sim) has the advantage that it has a nonzero gradient even for non-intersecting silhouettes, which enables the use of our method with a poor initial guess.

Our final goal is to minimize the difference between the silhouettes of the real and virtual objects. For a silhouette without shadow, we measure the similarity by

$$E_c = \mathrm{sim}(\mathbf{R}_c, \mathbf{V}_c, \lceil \|\mathbf{R}_c\| \rceil), \tag{4}$$

where $\|\cdot\|$ denotes the length of a silhouette in pixels. Ideally, we would like to sample as many points as possible. In this performance vs. accuracy trade-off, we have chosen $n = \lceil \|\mathbf{R}_c\| \rceil$ to place the sampled points approximately one pixel apart.

To compare silhouettes including shadows, we introduce a similarity measurement $E_{c,\ell}$. As mentioned previously, we would like to refine estimates using multiple cameras and light sources as long as only one is active per image. We compute the sum of comparisons of silhouettes over one or more configurations as follows:

$$E_s = \sum_{\ell} \sum_{c} \Big(\underbrace{\mathrm{sim}(\mathbf{R}_{c,\ell}, \mathbf{V}_{c,\ell}, \lceil \|\mathbf{R}_{c,\ell}\| \rceil)}_{E_{c,\ell}} + E_c \Big). \tag{5}$$

In the following, we describe how we estimate object pose and light source position using these silhouette similarity measurements.

D. Pose Estimation

We compute the pose of the object independently for each camera. We do this in camera space, where the camera is fixed at the origin. In the end, we can then use the known relation between object and ground plane to position each camera in world space.

Starting in camera space, the first step of the pose estimation is to find an initial guess for the position of the object. We do this by minimizing E_c with respect to the position, which places the virtual object approximately in the same position as the real object.

To find a good initial guess of the rotation, we randomly sample rotations. For each rotation, we compare the silhouette of the digital object to the real object using Hu's moment invariants [32]. These are calculated from image moments but are invariant to scale, rotation, and translation. For an image of pixel values $I(x, y)$, the image moments are defined by

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q I(x, y)\, dx\, dy, \tag{6}$$

where the p and q exponents are the moment orders and integration is across the image plane. Since the silhouette can be considered a polygon, the image moments can be computed efficiently by applying Green's theorem [48]. Hu's moment invariants are seven polynomial combinations of image moments that we store in a vector and compare using the sum of squared differences. Using the Hu moment invariants, the search space of the rotation is practically reduced to two dimensions. The rotation giving the silhouette that best matches the Hu moment invariants of the real silhouette is chosen as the initial guess of the rotation. We parameterize the rotation using quaternions and use the centroid of the object as the rotation centre.

With these initial guesses for position and rotation, we minimize E_c, which gives an object pose for each view. We use Levenberg-Marquardt [49, 50] for the minimization. This is possible as E_c is a sum of squares. As part of our input, we know the object pose in relation to the ground plane. We use this to convert the per camera object poses into camera poses in world space.

E. Light Positions and Final Refinement

To estimate the position of each light, we randomly sample positions and then choose the one with the lowest $E_{c,\ell}$ for each light. Following this, $E_{c,\ell}$ is minimized using Levenberg-Marquardt.

The last step of our method is a joint optimization where we minimize E_s with respect to object pose, camera pose(s), light position(s), and a non-uniform scaling of the mesh. The non-uniform scaling of the mesh is to compensate for some of the shrinkages that may occur during 3D printing. The final optimization of the object pose is beneficial as the inclusion of

the shadow silhouette(s) enables us to use more information from the input image.

F. Known Camera Poses

If camera poses are known in advance, for example from a stereo calibration of the camera rig, we can use the same steps as in Sec. D to find the pose for all cameras jointly. When finding the rotation, it is then no longer desirable to have rotational invariance for all cameras. Instead, we propose to rotate the object to align the normalized image moments of the virtual and digital objects in the best way possible along a randomly chosen camera's viewing direction. The rotation is found by aligning the principal components of the two silhouettes [51]. We choose the rotation that best matches the normalized image moments across all cameras as the best rotation.

An initial guess of the object's scale is required, but if the camera poses are known in relation to the ground plane, we need not know the rotation of the object relative to the ground plane. The method for light source estimation is as with unknown camera poses.

4. COMPOSING APPEARANCE MODELS FOR REAL OBJECTS

Rendering systems provide a multitude of rendering techniques that we need to choose among when composing an appearance model for a real object. We start from a very approximate model at the most macroscopic scale. We then gradually increase complexity by reconsidering the involved optical properties [52] and what types of materials and visual effects that they can model.

At the most macroscopic scale, we have the bidirectional reflectance/transmittance distribution function (BRDF/BTDF) and the simplest models at this scale are the ones for perfectly diffuse and perfectly specular materials [53]. To cover a broad spectrum of different material types, we consider three different starting points: (a) diffuse, (b) metallic, or (c) transparent. In the following, we describe existing appearance models for these material types as well as model extensions (Sections A–C).

The perfectly diffuse (or Lambertian) material is a good starting point for objects that exhibit a significant amount of subsurface scattering (a). The BRDF of a perfectly diffuse material is $f_{r,d} = \rho_d/\pi$, where ρ_d is the bihemispherical diffuse reflectance, which we can set in an RGB renderer using a colour vector in $[0,1]^3$. This reflectance represents the subsurface scattering of the material. We can then add an interface to model highlights and switch to a bidirectional scattering-surface reflectance distribution function (BSSRDF) to model translucency.

The Fresnel equations for reflection are an excellent starting point for metallic and transparent objects (b-c). Given an interface between two media of refractive indices n_i and n_t, the Fresnel equations provide complex ratios of reflected to incident wave amplitudes for the part of the light polarized perpendicularly ($\perp$) and parallelly ($\parallel$) to the plane of incidence [54]:

$$\tilde{r}_\perp = \frac{n_i\cos\theta_i - n_t\cos\theta_t}{n_i\cos\theta_i + n_t\cos\theta_t}, \quad \tilde{r}_\parallel = \frac{n_t\cos\theta_i - n_i\cos\theta_t}{n_t\cos\theta_i + n_i\cos\theta_t}. \tag{7}$$

Here, i is subscript for incidence and t is for transmission. For unpolarized light incident on a perfectly smooth interface, the bidirectional reflectance is

$$F = \frac{1}{2}\left(|\tilde{r}_\perp|^2 + |\tilde{r}_\parallel|^2\right), \tag{8}$$

and we find the cosine of the angle of refraction by

$$\cos\theta_t = \sqrt{1 - \left(\frac{n_i}{n_t}\right)^2 (1-\cos^2\theta_i)}. \tag{9}$$

The cosine resulting from this formula may be a complex number, and refractive indices of conductors (metals) and absorbing materials are also complex. Taking absolute values in Eq. (8) is thus important to obtain a real-valued bidirectional reflectance. These formulas return the right result both in the case of total internal reflection and in the case of metals/conductors.

The BRDF/BTDF of a perfectly smooth or a rough interface are available from Walter et al. [18]. The BRDFs presented by these authors work just as well for metals as long as we use the complex index of refraction of the metals to find the Fresnel factor F. The key difficulty in use of the Fresnel equations is that indices of refraction are physical parameters that are defined as a spectrum rather than colours. We can convert a spectrum to a representative RGB vector using weighted averages based on RGB colour matching functions [55, 56]. Assuming known (complex) index of refraction, the key parameter for metallic and transparent objects is the surface roughness (which is different for different surface microfacet distributions [18, 57]).

A natural extension of the diffuse model (a) is to introduce a refractive interface. The BRDF then becomes a sum of a specular and a diffuse component [58]. We can think of the specular term as in-surface scattering and of the diffuse term as subsurface scattering. The Fresnel equations are then useful for ensuring energy conservation (and reciprocity) both for smooth surfaces [59] and for rough surfaces [60]. The trick is to sample the BRDF/BTDF of a transparent surface [18] and then let incident light that refracts into the material reflect diffusely before it refracts back out of the material using the BTDF of the surface again but this time for the outgoing direction. This enables addition of glossy reflections and highlights to an object with an otherwise matte appearance.

A natural extension of the transparent model (b) is to account for absorption based on the distance d that a ray travels through the interior of the object. This is done using an (RGB) absorption coefficient σ_a and Bouguer's law of exponential attenuation of light (attenuation factor $e^{-\sigma_a d}$). The absorption coefficient is directly linked to the imaginary part of the index of refraction [54]:

$$\sigma_a = \frac{4\pi\,\mathrm{Im}(n_t)}{\lambda}, \tag{10}$$

where λ is the wavelength in a vacuum and Im takes the imaginary part of a complex number. The index of refraction was assumed known, and for metals σ_a is very large. We can thus assume that all light transmitted into a metal is absorbed. However, for transparent objects, $\mathrm{Im}(n_t)$ is often very small and σ_a may need some adjustment to account for dissolved substances [55] or impurities [56]. The absorption coefficient then becomes an RGB parameter in the model that controls the colour of transmitted light.

A further extension of the diffuse model (a) is to replace $f_{r,d}$ with proper subsurface scattering, where light may be incident at one surface position and observed at another. In terms of input parameters, this requires knowledge of the (RGB) scattering coefficient σ_s and the phase function. The latter is the distribution of the scattered light, which is often represented by an analytical model taking an (RGB) asymmetry parameter (g) as input. Several rendering techniques are available for evaluating the volumetric light transport between two surface positions [61].

For highly scattering materials, however, a full-fledged unbiased path tracing technique [62] is unpractical due to long rendering times. We need faster rendering when tuning parameters based on comparison of renderings with a reference photograph. A more practical rendering technique for subsurface scattering is then to use an analytical approximation of the BSSRDF [19, 20].

The standard dipole approximation for subsurface scattering [20] does not model how the direction of the incident light influences the subsurface scattering. To include this component, we can use a directional dipole approximation [19]. However, these models use Fresnel terms that assume a perfectly smooth interface. Donner and Jensen [63] explained how to account for a rough surface with a distribution of microfacet normals [57, 58]. In the following, we describe how to account for a rough surface in the case of a model that accounts for the directional dependency of the subsurface scattering. We also describe simplistic models that we use to account for spatial variation in the surface roughness of our example objects.

A. Directional Subsurface Scattering for Rough Surfaces

The BSSRDF depends on the object geometry X, the position $\mathbf{x}_i$ and the direction $\vec{\omega}_i$ of the incident light as well as the position $\mathbf{x}_o$ and the direction $\vec{\omega}_o$ of the observed light. The normals at the points of incidence and observation $\vec{n}_i$ and $\vec{n}_o$ are known from the object geometry. An analytic BSSRDF model developed for a material with a smooth surface then usually has the form

$$S(X;\mathbf{x}_i,\vec{\omega}_i;\mathbf{x}_o,\vec{\omega}_o) = F_t(\vec{\omega}_o\cdot\vec{n}_o)(S_d+S^*)F_t(\vec{\omega}_i\cdot\vec{n}_i)\,, \quad \textbf{(11)}$$

where $F_t = 1-F$ is Fresnel transmittance, S_d is the diffusive part, which is typically modeled by a dipole, and S^* is the remaining light transport. The number of arguments used with S_d and S^* is different for different models.

To incorporate a rough surface in a BSSRDF model of this kind, we add a BRDF in the special case where the point of incidence equals the point of emergence, and we insert hemispherical transmittance integrals in place of the Fresnel terms:

$$\begin{aligned} S(X;\mathbf{x}_i,\vec{\omega}_i;\mathbf{x}_o,\vec{\omega}_o) &= \delta(\mathbf{x}_o-\mathbf{x}_i)f_r(\mathbf{x}_o,\vec{\omega}_i,\vec{\omega}_o) \\ &+ \int_{2\pi}\int_{2\pi} f_t(\mathbf{x}_o,\vec{\omega}_{21},\vec{\omega}_o)(-\vec{n}_o\cdot\vec{\omega}_{21})(S_d+S^*)\,d\omega_{21} \\ &\quad f_t(\mathbf{x}_i,\vec{\omega}_i,\vec{\omega}_{12})(-\vec{n}_i\cdot\vec{\omega}_{12})\,d\omega_{12}\,, \end{aligned} \quad \textbf{(12)}$$

where f_r is the BRDF and f_t is the BTDF of the surface, δ is a Dirac delta function, $\vec{\omega}_{12}$ is the direction of a ray transmitted into the volume, and $\vec{\omega}_{21}$ is the direction of a ray to be transmitted out of the volume. The directions $\vec{\omega}_{12}$ and $\vec{\omega}_{21}$ would thus be the ones to use as arguments for the S-functions.

The S^* term is usually fully directional, and the integrations over BTDFs at $\mathbf{x}_i$ and $\mathbf{x}_o$ are evaluated using regular volume path tracing with rough refraction at the interfaces. In the case of the standard dipole [20], $S^* = S^{(1)}$ includes evaluation of single scattering in the volume. In the case of the directional dipole [19], $S^* = S_{\delta E}$ is evaluated in the same way as absorption in a transparent material, but with a modified coefficient in the exponential attenuation. One should note that analytic expressions are available for the Fresnel transmittance integrals in cases where S_d is independent of $\vec{\omega}_i$ and/or $\vec{\omega}_o$ [63, 64]. Some care must be taken as some models [19, 64] assume a diffuse distribution of the light at $\mathbf{x}_o$ and then include the integration over $\vec{\omega}_{21}$ in their formulation. In the case of the directional dipole, our expression becomes

$$\begin{aligned} S(X;\mathbf{x}_i,\vec{\omega}_i;\mathbf{x}_o,\vec{\omega}_o) &= \delta(\mathbf{x}_o-\mathbf{x}_i)f_r(\mathbf{x}_o,\vec{\omega}_i,\vec{\omega}_o) + S^*_{\delta E} \\ &+ \int_{2\pi} S_{d,\mathrm{dir}}(\mathbf{x}_i,\vec{\omega}_{12};\mathbf{x}_o)f_t(\mathbf{x}_i,\vec{\omega}_i,\vec{\omega}_{12})(-\vec{n}_i\cdot\vec{\omega}_{12})\,d\omega_{12}\,, \end{aligned} \quad \textbf{(13)}$$

where $S_{d,\mathrm{dir}}$ is the diffusive part of the BSSRDF in the directional dipole model, but taking the transmitted direction directly as input instead of $\vec{\omega}_i$, and $S^*_{\delta E}$ is the modified reduced intensity term appearing in this model, but here including the BTDF integrations (rough refractions at the interfaces).

Comparing Eq. (12) to common illumination models [1, 58], the first term corresponds to the specular term and the second term corresponds to the diffuse term. The BRDF f_r to be used for the first term should therefore not include an added diffuse term. The BSDF (collective name for BRDF and BTDF) used in Eq. (12) should rather depend only on surface properties, such as a distribution of microfacet normals, see the work of Walter et al. [18] for examples. In particular, we use the so-called GGX distribution developed by these authors. This distribution has a width parameter α_g that we refer to as the GGX roughness.

B. Surface Roughness of a 3D Printed Object

Since most 3D printers print in layers, the surface of a printed object is usually rougher when the intended surface normal points in a direction aligned with layer edges in the voxel cubes of the print volume. If the z-axis is the print direction, we can use the following function to control the GGX roughness (α_g) based on the z-component of the surface normal (n_z):

$$\alpha_g = \rho + (1-\rho)\frac{|\sin(2\theta)|^s}{s} = \rho + (1-\rho)\frac{\left(2|n_z|\sqrt{1-n_z^2}\right)^s}{s}, \quad \textbf{(14)}$$

where θ is the angle of the surface normal $\vec{n}$ with the z-axis. We can think of the user parameters as follows: $\rho \in [0,1]$ is the minimum roughness and $s > 0$ is the shininess, which controls the height and width of the bumps in the curve around angles of $\pm 45°$, $\pm 135°$.

C. Surface Roughness of a Polished Metal Object

Quick hand polishing of a metallic object can result in an object with a rougher surface in curved areas and a smoother surface in flat areas. One way to specify the curvature of an object is using the mean curvature normal $\mathbf{H}$ [65]. This is a quantity that we can precompute for a triangle mesh using vertex circulators and store as a vertex attribute. The dot product of the outward-pointing surface normal $\vec{n}$ and the mean curvature normal $\mathbf{H}$ provides a signed measure of the curvature, where positive is a concavity and negative is a convexity. We use the absolute value of this dot product as an indicator of areas that were maybe not as easy to polish. To reduce noise from the surface scan and set a high roughness for curved areas, we employ a sigmoid function. Our use of the mean curvature normal is demonstrated in Figure 3, and the formula is

$$\alpha_g = \rho + \frac{1-\rho}{1+\exp(s\,(1-30\,|\mathbf{H}\cdot\vec{n}|))}\,, \quad \textbf{(15)}$$

where ρ is again minimum roughness and s is a sort of shininess while $\mathbf{H}$ is the mean curvature normal after division by the length of the longest mean curvature normal in the triangle mesh.

5. RENDERING

We implemented a progressive unidirectional path tracer using OptiX [66]. To include subsurface scattering, we sample a new set of surface positions for each progressive update. For each update and within each pixel, the ray tracer generates a random

Fig. 3. Model of spatially varying roughness α_g for an aluminium bust that has from time to time been subjected to hand polishing. We use the dot product of the mean curvature normal $\mathbf{H}$ and the surface normal $\vec{n}$. The model correctly marks eyes, hair, nostrils, and engraved letters as rough, but also incorrectly marks edges along the box-like base of the bust as being very rough.

position $\mathbf{x}_p$ in pixel coordinates. With the rotation of the camera relative to the object $\mathbf{R}$ and the camera intrinsic matrix $\mathbf{K}$, we get the direction of the corresponding ray using

$$\vec{\omega} = (\mathbf{KR})^{-1}\mathbf{S}\,\mathbf{x}_p = \mathbf{R}^T\mathbf{K}^{-1}\mathbf{S}\,\mathbf{x}_p\,. \tag{16}$$

Since the intrinsic matrix $\mathbf{K}$ is locked to the resolution of the camera ($W_c \times H_c$), which is usually very high, we use the scaling matrix $\mathbf{S} = \mathrm{diag}(W_r/W_c, H_r/H_c, 1)$ to enable rendering in a different resolution ($W_r \times H_r$).

6. RESULTS

The three objects of interest are (a) a translucent 3D print of the Stanford bunny, (b) an aluminium bust, and (c) a cupped angel figurine 3D scanned and printed using almost transparent resin. Two of our test objects (b-c) were 3D scanned using structured light based on phase shifting [17]. Our 3D printed objects (a, c) were produced using vat photopolymerization additive manufacturing processes. In our pose estimation and renderings, we used the geometry of these objects without correction for print artifacts. The Stanford bunny was printed by Luongo et al. [67] using red Industrial Blend resin (manufactured by Fun To Do) and a digital light processing (DLP) printer developed for research. The object was kindly lent to us. The angel was printed using a commercial stereolithography (SLA) 3D printer (Peopoly Maoi), and the general-purpose resin IM2.0 GP1 (manufactured by AddiFab). We use a real index of refraction of 1.54 for the printed objects as this is in the middle of the range of commercial acrylic resins with low shrinkage after photopolymerization [68]. Our three objects all have a rough surface and exhibit different types of spatial variation in this roughness.

We used our method to align renderings of the objects of interest with their photographs. We then tested different appearance models following the presented guidelines, where we started from a simplistic model and gradually added complexity. In each case, our end result is an appearance model and a rendering paired with a photograph for validation that would serve as a suitable starting point for an inverse rendering technique. The optical properties that we estimated for our different objects are in Table 1. The reference photographs and the associated CAD files and relative camera and light source alignments will be available as a supplement. We encourage the reader to use this dataset for testing preferred appearance models and rendering software. Another option is to use the dataset for finding better optical properties including better spatial variation of surface roughness by means of inverse rendering.

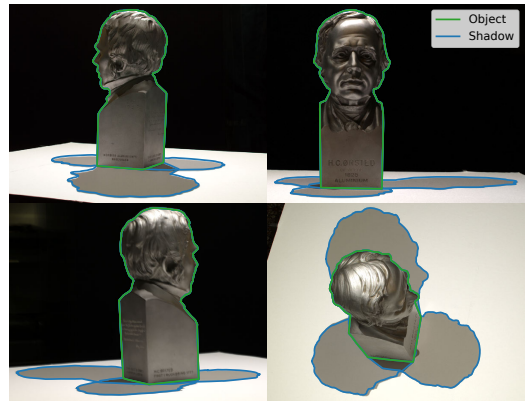


Fig. 4. Each image is an additive blend of three photos of the bust illuminated by the light source at different positions and overlaid with aligned silhouettes of the digital object.

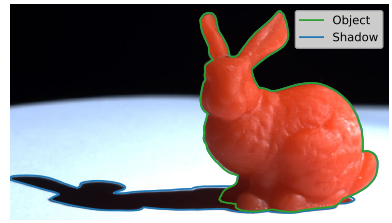


Fig. 5. Photo of the bunny overlaid with the aligned silhouette of the digital object.

A. Acquisition

The objects were placed on a flat piece of paper and illuminated by a Thorlabs MNWHL4 LED light source. This source is neutral white with a reasonably point-like radiation distribution. The bunny (a) and the angel (c) were captured using a FLIR Grasshopper3 GS3-U3-60QS6C-C camera, while the bust (b) was captured using D3200, D7000, D7500 and D750 cameras from Nikon. We used four cameras to cover all angles of the object while also taking multiple images from the same positions with different light positions. As different cameras were used, the images of the bust were colour calibrated using a ColorChecker from X-Rite. We performed camera calibration [43, 69] using a ChArUco board which is a checkerboard with ArUco markers [70]. For the bunny (a), we did not use the estimated extrinsics and only used the estimated focal length from the intrinsics.

B. Alignment

To segment the photographs as required by our alignment method, we used thresholding followed by hole closing and selected the largest connected component. For the images of the bunny and the angel, some manual cleaning of the segmentation was necessary due to caustics.

Our test cases span different setups to showcase the flexibility

Table 1. Estimated optical properties.

Material	n	σ_a	σ_s	ρ	s
Bunny (FTD, red Industrial Blend)	1.54	$(0.33, 25, 67) \cdot 10^3$ m^{-1}	$(10, 21, 0.083) \cdot 10^3$ m^{-1}	0.20	2.4
Angel (AddiFab, IM2.0 GP1)	1.54	$(0.032, 32, 640)$ m^{-1}	0	0.15	5.0
Bust (aluminium)	$(1.04, 0.76, 0.49) + i\,(6.45, 5.73, 4.76)$	$1.3 \cdot 10^8$	n/a	0.22	4.5

Fig. 6. Photos of the angel overlaid with the aligned silhouette of the digital object.

of our alignment method. For the bunny (a), we use just a single picture with unknown camera pose to align the scene. For the angel (c), we use two camera poses and a single light position to do the estimation. Finally, the bust (b) was captured from four camera poses, each with four different light source positions, yielding a total of 16 images that we used to do the alignment. The more light source positions, the more information we have available for the pose estimation. This comes at the small cost of increasing the dimensionality of the optimization problem. If we again consider our method an enabler for inverse rendering, it is an advantage to have multiple light positions as these provide additional samples for estimation of BRDFs, for example.

Outputs from our alignment method are in Figures 4 to 6. We achieve good alignment of the outlines of the bust, which makes sense as this is the only object in our collection for which the geometry is directly from the photographed object. Both the angel and the bunny have a quite good alignment, but especially the bunny has noticeable differences between the rendered silhouette and the object. We presume these mostly stem from non-linear shrinkages during printing that our method cannot account for. For the angel, our method estimated shrinkages of 3%, 6%, 1% in the x, y, z directions as compared to the size of an ideal 3D print.

C. Appearance

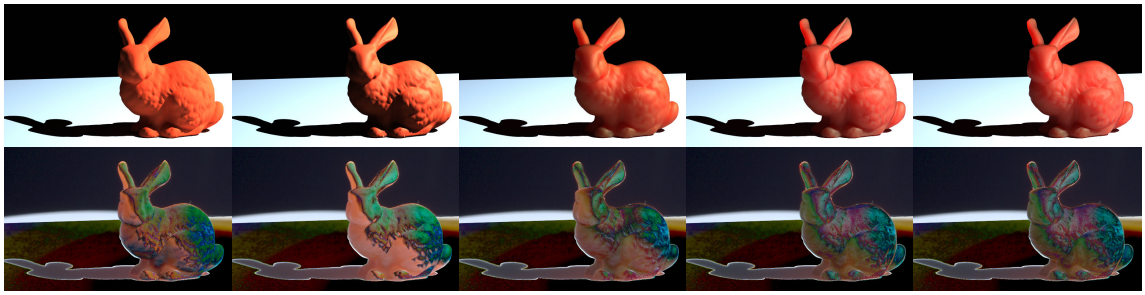
Since our objects are placed on a piece of paper assumed to be flat, we place a quad in the ground plane and resize it manually to approximately fit the paper observed in the photograph. Precise alignment of the paper could be part of the object alignment, but we find that it is not so important with respect to testing the appearance model applied to the object. To start simple, we consider the paper to be a diffuse surface. More complexity could easily be added to the paper appearance model [71], but we focus our attention on the objects of interest.

We initialise the diffuse reflectance of the paper to $\rho_d = (0.8, 0.8, 0.8)$ and select the simplest shading model for the material category of the object in question. We then use the intensity of the light reflected from the paper to estimate the intensity of the point light. Since our source is neutral white, we use the same intensity in all colour bands. An easy way to do a comparison is using two coloured difference images: one for positive difference and one for negative difference (see examples in Figure 1). Once the light intensity has been set, we modify the reflectance values until each colour appears equally in the positive and the negative difference image. We also evaluate our results quantitatively using root-mean-squared error (RMSE) (lower is better) and structural similarity (SSIM) index [72] (higher is better). The initial results for each of our three test cases are leftmost in Figures 7 to 9.

To estimate absorption and scattering coefficients (σ_a and σ_s), we need the physical size of the object as these optical properties are measured per distance unit that a ray has travelled through the material. Using the physical dimensions of the object, we get the coefficients in Table 1. We decided to leave the phase function as isotropic ($g = 0$) since the analytic BSSRDF models mostly use the reduced scattering coefficient $\sigma_s' = \sigma_s(1 - g)$ and thus do not distinguish much between a reduction in σ_s and an increase of g. The directional dipole is not exclusively based on the reduced scattering coefficient, but the role of g seems limited. When estimating the coefficients, 10 over the length of the bounding box diagonal is usually a good value to start with for the absorption or the scattering to have a reasonable effect.

Refinement of the model for the rough translucent bunny (a). Figure 7. We first add an interface to the model [59, 60] to enable rendering of highlights. However, this also directs a lot of energy into a glossy reflection lobe meaning that the missing transport of light from the point of incidence to a different point of emergence becomes apparent and RMSE and SSIM both worsen. As soon as we switch to a model that accounts for this subsurface light transport [20], the result becomes better than the Lambertian model. This is true even without single scattering and assuming that the surface is perfectly smooth. The directional dipole [19] and our spatially varying roughness from Sec. B further improve the result. However, the models cannot fully represent the scattering process. This is probably due to limiting assumptions such as diffuse emergent light and a locally flat, convex object. It should be mentioned that the bunny was printed using greyscale values to reduce staircasing artefacts [67]. These staircasing artefacts due to layered printing are significantly less pronounced for the bunny as compared with the angel (which was not printed using greyscale values). Nevertheless, the bunny object still exhibits some spatial variation in its roughness that we have modelled.

Refinement of the model for the aluminium bust (b). Figure 9. We use the complex index of refraction of aluminium from McPeak et al. [73] (this is available for download at refractiveindex.info). Since we have a dark scene with a point light, the appearance is off without surface roughness (as highlights then disappear). Adding a microfacet normal distribution was thus essential for this case, and we found that the GGX distribution [18] provided a good result. When adding spatially varying roughness based on the curvature, we found that SSIM would improve for a larger shininess s at the cost of a poorer RMSE. The SSIM-improved result is in Figure 1. The RMSE probably suffers from a slight misplacement of the highlight peak in the forehead of the bust.



RMSE: 0.1152 SSIM: 0.8108 Lambertian	RMSE: 0.1237 SSIM: 0.7931 interfaced	RMSE: 0.1136 SSIM: 0.8145 standard SSS	RMSE: 0.1127 SSIM: 0.8177 directional SSS	RMSE: 0.1124 SSIM: 0.8180 SV roughness

Fig. 7. Renderings (top) and gamma-corrected absolute difference images (bottom, $\gamma = 2.2$) to test appearance models for the rough translucent bunny. The interfaced model adds a rough surface with a GGX microfacet normal distribution [18, 59, 60]. The standard subsurface scattering (SSS) model is the standard dipole including path traced single scattering and a rough surface [20, 63]. The directional SSS model uses the directional dipole [19] and incorporates a rough surface (Sec. A). The model with spatially varying (SV) roughness uses Eq. (14). Further comparison of the input image with the end result is in Figure 1. The not quite so flat paper worsens both RMSE and SSIM by approximately 0.05.

Refinement of the model for the rough transparent angel (c). Figure 8. Using the convention that surface normals always point outwards, absorption is easily included by applying Bouguer's law of exponential attenuation to all rays that hit the surface from the inside. Accounting for absorption and a rough interface is highly important when modelling the appearance of the angel. Apart from this, the print layers are visually obvious, especially in highlights. We, therefore, tried to model the layers by calculating a layer index based on the point of intersection and using an increased roughness for every second layer. This represents the rougher layer edges more explicitly. Visually, we find this layered result more convincing and it also has lower RMSE, but SSIM disagrees. We tried adding single scattering to the material, but this only seemed to worsen RMSE and SSIM. Thus, it seems that the remaining deviations from the reference are mostly due to geometric print artefacts and inaccuracies in the spatial variation of the surface roughness.

7. DISCUSSION

While we use a pinhole camera model, one should note that our method can also work for more advanced camera models as we can apply the necessary transformations to the object edges before computing the silhouette. Extending to area lights is however challenging and left for future work.

A disadvantage of using silhouettes is their simplicity. In some cases, they describe the features of an object inadequately, which can cause ambiguities in the pose estimation. An example of this could be a bowl with contents, where the silhouette only contains information enough to pose estimate the bowl. To have more information, some methods [28] also use features on the object itself. In cases where the segmentation has inaccuracies and our pose may have small errors, our method is still useful for obtaining a good initial guess that can be refined by other methods (such as differentiable rendering).

8. CONCLUSION

We presented a practical method for aligning photographs with rendered images. Our method is based on silhouette matching and estimates both object pose and the position of a point-like light source. If multiple images have been captured from different views and/or with light sources in different positions, our method can include this added information in the pose estimation. As opposed to differentiable rendering techniques, our method works not only in pixel space but in the entire image plane. This means that we can estimate a pose from a very poor initial guess. Thus we find our work a practical enabling technique for inverse rendering that could be based on differentiable rendering.

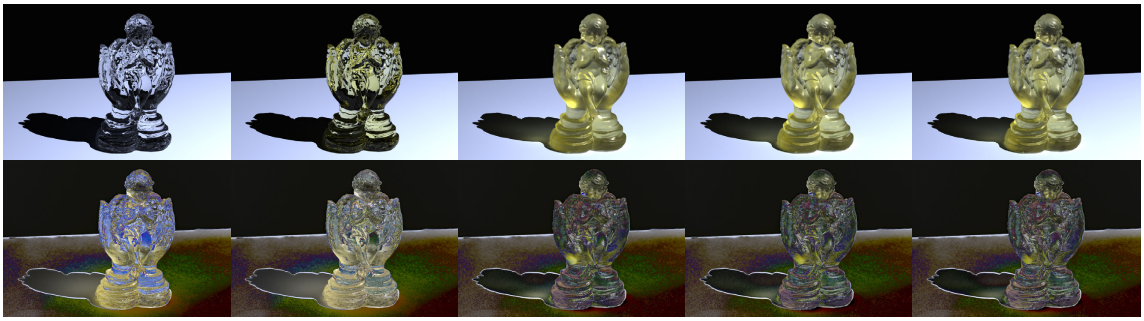
Given an alignment, we proposed a procedure for composing an appearance model suitable for the photographed object. The concept is to start from a simplistic model and gradually increase the complexity of appearance models guided by difference images and quantitative metrics such as RMSE and SSIM. As a consequence of this approach, we presented extensions of existing models providing improved photorealism. One extension was the combination of a rough surface with directional subsurface scattering. We believe that practical alignment of photographs with renderings is an important step in furthering the predictive abilities of appearance models.

ACKNOWLEDGMENTS

Thanks to Macarena Mendez Ribo for 3D printing the transparent angel and to Andreas Bærentzen for useful discussions on finding a silhouette given projected polygon edges.

REFERENCES

1. B. T. Phong, "Illumination for computer generated pictures," Communications of the ACM **18**, 311–317 (1975).
2. C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," Computer Graphics (SIGGRAPH '84) **18**, 213–222 (1984).
3. G. W. Meyer, H. E. Rushmeier, M. F. Cohen, D. P. Greenberg, and K. E. Torrance, "An experimental evaluation of computer graphics imagery," ACM Transactions on Graphics **5**, 30–50 (1986).
4. H. Rushmeier, G. Ward, C. Piatko, P. Sanders, and B. Rust, "Comparing real and synthetic images: Some ideas about metrics," in "Rendering Techniques '95," (Springer, 1995), pp. 82–91.



RMSE: 0.1267 SSIM: 0.7699 glass | RMSE: 0.1135 SSIM: 0.8051 absorbing | RMSE: 0.0741 SSIM: 0.8876 rough | RMSE: 0.0733 SSIM: 0.8870 SV roughness | RMSE: 0.0730 SSIM: 0.8858 layered

Fig. 8. Renderings (top) and gamma-corrected absolute difference images (bottom, $\gamma = 2.2$) to test appearance models for the rough transparent angel. We use the GGX microfacet normal distribution [18] and add absorption through analysis by synthesis [56] and spatially varying (SV) roughness (Sec. B). We also tested a layered variation of the roughness in the print direction (every second layer is rougher to model a staircase). SSIM is sensitive to structure and takes a hit because the layers do not perfectly match the real layers. Further comparison of the input image with the end result is in Figure 1.

RMSE: 0.0311 SSIM: 0.9339 metallic | RMSE: 0.0159 SSIM: 0.9693 rough | RMSE: 0.0134 SSIM: 0.9713 SV roughness

Fig. 9. Renderings (top) and gamma-corrected absolute difference images (bottom, $\gamma = 2.2$) to test appearance models for the aluminium bust. We test spatially varying (SV) roughness as depicted in Figure 3 and use high dynamic range when computing differences. The input image is in Figure 1, where it is compared with an SSIM-improved result (RMSE: 0.0148, SSIM: 0.9725).

5. S. N. Pattanaik, J. A. Ferwerda, K. E. Torrance, and D. P. Greenberg, "Validation of global illumination solutions through CCD camera measurements," in "Proceedings of Color Imaging Conference (CIC 1997)," (1997), pp. 250–253.
6. C. Ulbricht, A. Wilkie, and W. Purgathofer, "Verification of physically based rendering algorithms," Computer Graphics Forum **25**, 237–255 (2006).
7. M. Weinmann and R. Klein, "Advances in geometry and reflectance acquisition (course notes)," in "Proceedings of SIGGRAPH Asia 2015 Courses," (ACM, 2015).
8. C. Reinbacher, M. Ruther, and H. Bischof, "Pose estimation of known objects by efficient silhouette matching," in "Proceedings of International Conference on Pattern Recognition (ICPR 2010)," (IEEE, 2010), pp. 1080–1083.
9. S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNet: Pixel-wise voting network for 6DoF pose estimation," in "Proceedings of CVPR 2019," (2019), pp. 4561–4570.
10. A. Panagopoulos, C. Wang, D. Samaras, and N. Paragios, "Illumination estimation and cast shadow detection through a higher-order graphical model," in "Proceedings of CVPR 2011," (IEEE, 2011), pp. 673–680.
11. J. Lopez-Moreno, E. Garces, S. Hadap, E. Reinhard, and D. Gutierrez, "Multiple light source estimation in a single image," Computer Graphics Forum **32**, 170–182 (2013).
12. R. Ramamoorthi and P. Hanrahan, "A signal-processing framework for inverse rendering," in "Proceedings of SIGGRAPH 2001," (ACM, 2001), pp. 117—128.
13. G. Loubet, N. Holzschuch, and W. Jakob, "Reparameterizing discontinuous integrands for differentiable rendering," ACM Transactions on Graphics **38**, 228:1–228:14 (2019).
14. M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, "Mitsuba 2: a retargetable forward and inverse renderer," ACM Transactions on Graphics **38**, 203:1–203:17 (2019).
15. G. Turk, "The Stanford bunny," https://www.cc.gatech.edu/~turk/bunny/bunny.html (2000).
16. G. Turk and M. Levoy, "Zippered polygon meshes from range images," in "Proceedings of SIGGRAPH '94," (1994), pp. 311–318.
17. J. Geng, "Structured-light 3D surface imaging: a tutorial," Advances in Optics and Photonics **3**, 128–160 (2011).
18. B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in "Proceedings of Eurographics Symposium on Rendering (EGSR 2007)," (The Eurographics Association, 2007), pp. 195–206.
19. J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen, "Directional dipole model for subsurface scattering," ACM Transactions on Graphics **34**, 5:1–5:12 (2014).
20. H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in "Proceedings of SIGGRAPH 2001," (ACM, 2001), pp. 511–518.
21. H. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel, "Image-based reconstruction of spatial appearance and geometric detail," ACM Transactions on Graphics **22**, 234–257 (2003).
22. M. Holroyd, J. Lawrence, and T. Zickler, "A coaxial optical scanner for synchronous acquisition of 3D geometry and surface reflectance," ACM Transactions on Graphics **29**, 99:1–99:12 (2010).
23. M. M. Loper and M. J. Black, "OpenDR: An approximate differentiable renderer," in "Proceedings of ECCV 2014," (Springer, 2014), pp. 154–169.
24. T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable Monte Carlo ray tracing through edge sampling," ACM Transactions on Graphics

37, 222:1–222:11 (2018).
25. B. Bhanu, "CAD-based robot vision," Computer **20**, 13–16 (1987).
26. J. Byne and J. A. D. W. Anderson, "A CAD-based computer vision system," Image and Vision Computing **16**, 533–539 (1998).
27. M. Ulrich, C. Wiedemann, and C. Steger, "CAD-based recognition of 3D objects in monocular images," in "Proceedings of ICRA 2009," (IEEE, 2009), pp. 2090–2097.
28. A. Petit, E. Marchand, R. Sekkal, and K. Kanani, "3D object pose detection using foreground/background segmentation," in "Proceedings of ICRA 2015," (IEEE, 2015), pp. 1858–1865.
29. B. Rosenhahn, T. Brox, D. Cremers, and H.-P. Seidel, "A comparison of shape matching methods for contour based pose estimation," in "International Workshop on Combinatorial Image Analysis," (Springer, 2006), pp. 263–276.
30. O. Tahri and F. Chaumette, "Complex objects pose estimation based on image moment invariants," in "Proceedings of ICRA 2005," (IEEE, 2005), pp. 436–441.
31. O. Tahri, H. Araujo, Y. Mezouar, and F. Chaumette, "Efficient iterative pose estimation using an invariant to rotations," IEEE Transactions on Cybernetics **44**, 199–207 (2013).
32. M.-K. Hu, "Visual pattern recognition by moment invariants," IRE transactions on information theory **8**, 179–187 (1962).
33. M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single image 3D object detection and pose estimation for grasping," in "Proceedings of ICRA 2014," (IEEE, 2014), pp. 3936–3943.
34. Z. Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6DOF pose estimation for textureless objects," in "Proceedings of ICRA 2016," (IEEE, 2016), pp. 2441–2448.
35. E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold *et al.*, "Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image," in "Proceedings CVPR 2016," (2016), pp. 3364–3372.
36. W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SDD-6D: Making RGB-based 3D detection and 6D pose estimation great again," in "Proceedings of ICCV 2017," (2017), pp. 1521–1529.
37. M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," in "Proceedings of ICCV 2017," (2017), pp. 3828–3836.
38. B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6D object pose prediction," in "Proceedings CVPR 2018," (2018), pp. 292–301.
39. Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep iterative matching for 6D pose estimation," in "Proceedings of ECCV 2018," (2018), pp. 683–698.
40. J. Blinn, "Me and my (fake) shadow," IEEE Computer Graphics and Applications **8**, 82–86 (1988).
41. N. Chotikakamthorn, "Near point light source location estimation from shadow edge correspondence," in "Proceedings of Cybernetics and Intelligent Systems (CIS) and Robotics, Automation and Mechatronics (RAM)," (IEEE, 2015), pp. 30–35.
42. S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," Computer Vision, Graphics, and Image Processing **30**, 32–46 (1985).
43. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools (2000).
44. U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," Computer Graphics and Image Processing **1**, 244–256 (1972).
45. D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," Cartographica: The International Journal for Geographic Information and Geovisualization **10**, 112–122 (1973).
46. F. Antonio, "Faster line segment intersection," in "Graphics Gems III," D. Kirk, ed. (Academic Press, 1992), pp. 199–202.
47. P. Alliez, S. Tayeb, and C. Wormser, "3D fast intersection and distance computation," CGAL user and reference manual **3** (2016).
48. X. Y. Jiang and H. Bunke, "Simple and fast computation of moments," Pattern Recognition **24**, 801–806 (1991).
49. K. Levenberg, "A method for the solution of certain non-linear problems in least squares," Quarterly of Applied Mathematics **2**, 164–168 (1944).
50. D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," Journal of the Society for Industrial and Applied Mathematics **11**, 431–441 (1963).
51. R. Candelier, "Tracking object orientation with image moments," http://raphael.candelier.fr/?blog=Image%20Moments (2016).
52. J. R. Frisvad, S. A. Jensen, J. S. Madsen, A. Correia, L. Yang, S. K. S. Gregersen, Y. Meuret, and P.-E. Hansen, "Survey of models for acquiring the optical properties of translucent materials," Computer Graphics Forum **39** (2020). To appear.
53. F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis, "Geometrical considerations and nomenclature for reflectance," Tech. Rep. NBS MN-160, National Bureau of Standards (1977).
54. M. Born and E. Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light* (Cambridge University Press, 1999), seventh (expanded) ed.
55. J. R. Frisvad, N. J. Christensen, and H. W. Jensen, "Computing the scattering properties of participating media using Lorenz-Mie theory," ACM Transactions on Graphics **26**, 60:1–60:10 (2007).
56. J. D. Stets, A. Dal Corso, J. B. Nielsen, R. A. Lyngby, S. H. N. Jensen, J. Wilm, M. B. Doest, C. Gundlach, E. R. Eiriksson, K. Conradsen, A. B. Dahl, J. A. Bærentzen, J. R. Frisvad, and H. Aanæs, "Scene reassembly after multimodal digitization and pipeline evaluation using photorealistic rendering," Applied Optics **56**, 7679–7690 (2017).
57. R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," ACM Transactions on Graphics **1**, 7–24 (1982).
58. K. E. Torrance and E. M. Sparrow, "Theory for off-specular reflection from roughened surfaces," Journal of the Optical Society of America **57**, 1105–1114 (1967).
59. P. Shirley, B. Smits, H. Hu, and E. Lafortune, "A practitioners' assessment of light reflection models," in "Proceedings Pacific Graphics 97," (1997), pp. 40–49.
60. M. Ashikmin, S. Premože, and P. Shirley, "A microfacet-based BRDF generator," in "Proceedings of SIGGRAPH 2000," (ACM/Addison-Wesley, 2000), pp. 65–74.
61. M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation* (Morgan Kaufmann/Elsevier, 2017), 3rd ed.
62. M. Raab, D. Seibert, and A. Keller, "Unbiased global illumination with participating media," in "Monte Carlo and Quasi-Monte Carlo Methods 2006," (Springer, 2008), pp. 591–605.
63. C. Donner and H. W. Jensen, "Light diffusion in multi-layered translucent materials," ACM Transactions on Graphics **24**, 1032–1039 (2005).
64. E. d'Eon and G. Irving, "A quantized-diffusion model for rendering translucent materials," ACM Transactions on Graphics **30**, 56:1–56:13 (2011).
65. J. A. Bærentzen, J. Gravesen, F. Anton, and H. Aanæs, *Guide to Computational Geometry Processing: Foundations, Algorithms, and Methods* (Springer, 2012).
66. S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: A general purpose ray tracing engine," ACM Transactions on Graphics **29**, 66:1–66:13 (2010).
67. A. Luongo, V. Falster, M. B. Doest, M. M. Ribo, E. R. Eiriksson, D. B. Pedersen, and J. R. Frisvad, "Microstructure control in 3D printing with digital light processing," Computer Graphics Forum **39**, 347–359 (2020).
68. F. Aloui, L. Lecamp, P. Lebaudy, and F. Burel, "Refractive index evolution of various commercial acrylic resins during photopolymerization," eXPRESS Polymer Letters **12**, 966–971 (2018).
69. Z. Zhang, "A flexible new technique for camera calibration," IEEE Transactions on pattern analysis and machine intelligence **22**, 1330–1334 (2000).
70. 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**, 2280–2292 (2014).
71. M. Papas, K. de Mesa, and H. W. Jensen, "A physically-based BSDF for modeling the appearance of paper," Computer Graphics Forum **33**, 133–142 (2014).
72. Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing **13**, 600–612 (2004).
73. K. M. McPeak, S. V. Jayanti, S. J. Kress, S. Meyer, S. Iotti, A. Rossinelli, and D. J. Norris, "Plasmonic films can easily be better: rules and recipes," ACS Photonics **2**, 326–333 (2015).

CONTRIBUTION F

Surface Reconstruction from Structured Light Images using Differentiable Rendering

This contribution is a work in progress that we expect to revise further soon.

Surface Reconstruction from Structured Light Images using Differentiable Rendering

Janus N. Jensen[1]* · Morten Hannemose[1]* · J. Andreas Bærentzen[1] · Jakob Wilm[2] · Jeppe R. Frisvad[1] · Anders B. Dahl[1]

Fig. 1: Our method reconstructing the Stanford Bunny starting from a sphere. From left to right: Initial mesh (sphere), after 50 iterations, after 750 iterations, and the converged result. The final reconstruction has 13 780 vertices and a Δ_V error of 0.30%, when compared to the ground truth. The reconstruction was done for $k = 1$.

Abstract A triangle mesh is a good model to digitally represent a surface because it gives a closed object interface. Most surface scanning methods such as structured light scanning are, however, based on computing a point cloud. To obtain a triangle surface mesh then requires a second meshing step. Here, we investigate the effect of a one-step approach, where we compute the triangle mesh directly from structured light images. To do so, we propose a new method based on minimizing the least-squares error between real images and renderings of a triangle mesh, where the positions of the vertices of the mesh are the parameters of the minimization problem. Our experiments show that computing a triangle mesh in one step using our method has several advantages over the two-step approach with an intermediate point cloud. Our method can produce accurate reconstructions when initializing the optimization from a simple sphere. We also show that our method is especially good at reconstructing sharp edges, that it is robust with respect to image noise, and that it can improve the output from other reconstruction algorithms when we use these as initialization.

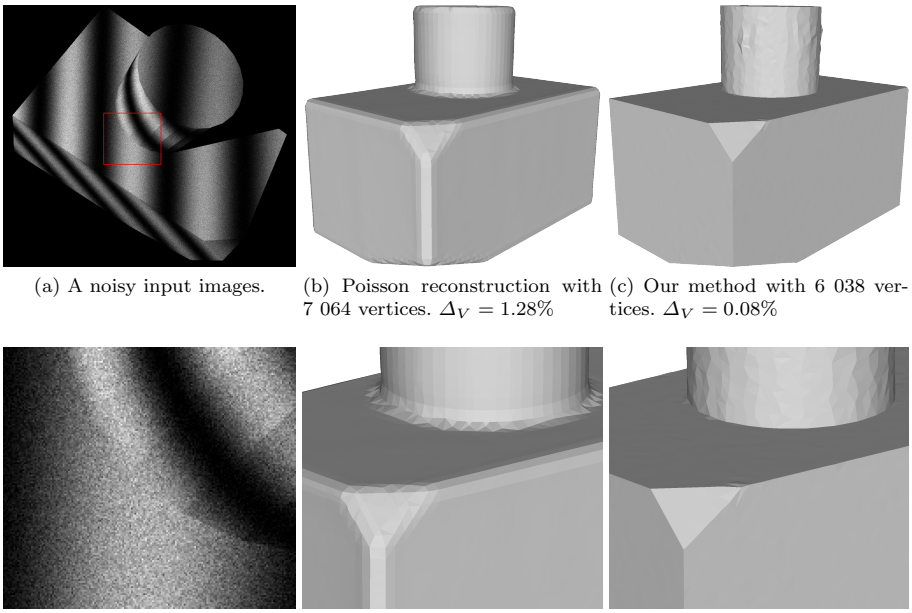
Keywords 3D surface reconstruction, structured light, differentiable rendering.

[1] DTU Compute, Technical University of Denmark, Lyngby, Denmark
E-mail: jnje@dtu.dk
[2] SDU Robotics, University of Southern Denmark, Odense, Denmark
* These authors contributed equally.

1 Introduction

Structured light 3D scanning of an object can be used to produce a point cloud from which we can reconstruct a triangle mesh. This mesh is then a digital representation of the surface of the scanned object. This has many applications including cultural heritage preservation and industrial quality control. For most applications, the accuracy of a recovered surface of the reconstruction is of great import. Typically, producing point clouds from phase shifting structured light images is rather cumbersome. It involves determining the phases, unwrapping these, re-sampling the unwrapped phases due to image distortion and rectification, finding point correspondences, and finally triangulating these. Afterwards, the point clouds from different sub-scans need to be merged before the final reconstruction of a triangle mesh. During this process of producing point clouds and subsequently a triangle mesh, the image noise propagates non-linearly to affect vertex positions in the reconstructed triangle mesh. We, therefore, propose to skip the point cloud, and the process of creating it, and instead reconstruct surfaces directly from image intensities to investigate how that affects the accuracy of the reconstructions. Using vertex positions as model parameters, we minimize the least-squares error between rendered and recorded images to obtain a triangle mesh directly.

Our method has three major implications: (1) explicit point triangulation from image correspondences is no longer needed; (2) we understand how noise in the



(a) A noisy input images. (b) Poisson reconstruction with 7 064 vertices. $\Delta_V = 1.28\%$ (c) Our method with 6 038 vertices. $\Delta_V = 0.08\%$

(d) Crop of noisy image. (e) Close-up of Poisson reconstruction. (f) Close-up of our method.

Fig. 2: Our method is able to reconstruct meshes with sharp edges from very noisy images ($k = 10^3$). Our method has a lower error (Δ_V) than a Screened Poisson reconstruction with a comparable number of vertices (spatial octree depth 7).

image data affects the final reconstruction; (3) the reconstruction is done for all image data simultaneously.

Similar approaches have been used for multi-view passive stereo but to the best of our knowledge not for structured light reconstructions. In passive stereo, dense correspondences can be established, usually yielding much higher accuracy in weakly textured areas.

Why are we solving this? In many reconstruction methods, the raw image data is not considered during the reconstruction. The reconstruction is instead done using unstructured point clouds that have been constructed from the image data. This means that any noise from the imaging process and from the point sampling process is not modeled in the reconstruction and may be propagated through to the end result.

2 Related work

Suppose we know the configuration of light and camera in a vision setup and the reflectance properties of the imaged object. Obtaining the shape of the object based on the observed shading of the object in an acquired image is then referred to as the shape-from-shading problem (Horn 1970). The original shape-from-shading method by Horn (1970, 1975) used so-called characteristic curves to describe the observed shape. The method was based on illumination from point-like sources and the shading would then only allow estimation of the gradient along a path. This was the reason for using a collection of curves to describe the object shape.

Curves are inconvenient in the sense that they require stitching to become a full surface description. One way to fit a mesh instead of curves is to use an optimization technique that does not require gradients. This has been done for a rectangular mesh using simulated annealing and simplex search (Rockwood and Winget 1997). Gradients are however preferable to ease the optimization problem. For triangular meshes, an approach has been developed based on image gradients (Zhang and Seitz 2000). Unfortunately, the shape can be hard to recover from image gradients due to color variance caused by normal variations in the surface. To keep gradient-based optimization while having a method more robust to surface reflectance deviating

from an assumption of a specific shading model, we use structured light with a differentiable pattern.

Combination of shape from shading with a structured light approach like phase shifting improves the performance of the shape estimation (Inagaki et al. 2001; Naganuma et al. 2003), and enables simultaneous acquisition of shape and object color (diffuse reflectance) (Naganuma et al. 2004). We do not include estimation of spatially varying reflectance in this study, but we note that this is an option. Only a height field was reconstructed in this previous work. We take the concept one step further and reconstruct a closed 3D object as in the work of Zhang and Seitz (2000) but also exploiting structured light.

The mesh-based reconstruction method of Zhang and Seitz (2000) was improved by Isidoro and Sclaroff (2002, 2003) through creation of a better initial mesh and by Yu et al. (2004, 2007) through use of a more physically based reflectance model. Another option is to use multiview stereo to acquire a good initial guess and then refine the mesh using a shape-from-shading approach Wu et al. (2011). Our method can be used in a similar way with the innovation of using structured light to improve the robustness of the mesh refinement.

Use of structured light has become a strong technique for point-based 3D reconstruction (Ha et al. 2015) but has to the best of our knowledge not previously been tested in mesh-based 3D reconstruction. We do this to have the benefits of a mesh-based technique. An important benefit is that the connectivity between points (vertices) is retained throughout the geometric refinement process.

The recent differentiable renderer from Loubet et al. (2019) uses ray tracing and reverse mode automatic differentiation. Use of such a framework for mesh refinement is an option. Every rendering is however quite computationally demanding, so the optimization would have a significant run time. Liu et al. (2019) introduced the soft rasterizer, which is a faster differentiable renderer based on a smoothed version of rasterization. This has shown promising results in other mesh reconstruction tasks, but to make it able to render the structured light of a projector is nontrivial.

3 Method

Our method fits a surface to a set of structured light images by minimizing the squared differences between rendered images and real images. We parameterize the object surface by a triangular mesh, and the parameters we optimize are thus the vertex positions $\mathbf{v}$. The real images are captured with structured light phase shifting from multiple camera-projector positions. We denote these images $\mathbf{I}_{c,p}$, where $c \in \mathcal{C}$ is the index of the camera-projector pair and $p \in \mathcal{P}$ is the index of the projected pattern. We render images $\widetilde{\mathbf{I}}_{c,p}(\mathbf{v})$ of our parameterized surface from the same camera-projector positions and find the optimal vertex positions by solving the following minimization problem

$$\underset{\mathbf{v}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{v}) = \underset{\mathbf{v}}{\operatorname{argmin}} \sum_{c\in\mathcal{C}} \sum_{p\in\mathcal{P}} \left\| \mathbf{I}_{c,p} - \widetilde{\mathbf{I}}_{c,p}(\mathbf{v}) \right\|_F^2, \quad (1)$$

where $\|\cdot\|_F$ is the Frobenius norm, i.e. we minimize the sum of squared differences over all pixels for all patterns and all camera-projector pairs.

We generate our structured light images by projecting phase shifted sinusoidal patterns of the form

$$\sin\left(2\pi n_p x + \phi_p\right), \quad (2)$$

where x is the x-coordinate of the projector normalized to $[0, 1]$, n_p is the number of periods in the pattern, and ϕ_p is a phase shift. With structured light images, such as phase shifted images made from Equation 2, the goal is usually to find the projectors x-coordinate, which subsequently can be used for triangulation.

Our method is not specific to phase shifting patterns, however we use differentiable patterns to make the minimization problem tractable.

3.1 Rendering images

To solve the minimization problem in Equation 1 we need to render the images $\widetilde{\mathbf{I}}_{c,p}(\mathbf{v})$ in each iteration. We want to adequately reproduce the structured light images that would have been obtained if our current parameterized surface was a real object. We achieve this by simulating the structured light process as seen from the viewpoint of the camera. We render the images using the formula

$$\widetilde{\mathbf{I}}_{c,p}(\mathbf{v}) = \mathbf{A}_c \sin\left(2\pi n_p \widetilde{\mathbf{X}}_c(\mathbf{v}) + \phi_p\right) + \mathbf{B}_c. \quad (3)$$

Here, $\sin(\cdot)$ is elementwise application of the sine function, n_p is the number of periods as in Equation 2, ϕ_p is the phase shift for the p^{th} pattern, and $\widetilde{\mathbf{X}}_c(\mathbf{v})$ contains the x-coordinates of the projector in the $[0, 1]$ range for each pixel. We use the x-coordinate of the projector as it is parallel to the offset between the camera and projector. The matrices $\mathbf{A}_c$ and $\mathbf{B}_c$ are amplitudes and biases that are estimated from the ground truth images by fitting sinusoids at each pixel location. We find the elements of $\widetilde{\mathbf{X}}_c(\mathbf{v})$ by tracing a ray from the camera through the center of each pixel and projecting

the point where it intersects the surface back to the projector. As we model the projector as a pinhole camera, the point is projected to the projector as follows

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \mathbf{P} \begin{bmatrix} \mathbf{r} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \mathbf{p}_4 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ 1 \end{bmatrix}, \tag{4}$$

where $\mathbf{P}$ is the projection matrix of the projector, $\mathbf{p}_i$ is the i^{th} column of $\mathbf{P}$, and $\mathbf{r}$ is the 3D point where the ray intersects the triangle face. The i, j^{th} element of $\widetilde{X}_c$ is then given by

$$\widetilde{X}_c^{i,j} = \frac{q_1}{q_3}. \tag{5}$$

If the ray does not intersect the surface, we treat the pixel as background and set $\widetilde{X}_c^{i,j}(\mathbf{v}) := X_c^{i,j}$, such that the corresponding term in the loss $\mathcal{L}(\mathbf{v})$ will be zero.

The amplitudes $\mathbf{A}_c$ are a measure of how much of the projector light is reflected into the camera, and $\mathbf{B}_c - \frac{1}{2}\mathbf{A}_c$ is a measure of the amount ambient light. We estimate these to have the renderings better resemble the true images. However, we only need to estimate these once for each viewpoint, and we are able to use these estimates repeatedly in each iteration of the optimization.

3.2 Optimizing the surface

We use gradient descent to solve the optimization problem in Equation 1. In order to do this, we need the gradient of $\mathcal{L}(\mathbf{v})$, which in turn depends on the gradient of the elements in $\widetilde{\mathbf{X}}_c(\mathbf{v})$. Recall that each of these elements is computed by tracing a single ray from the camera to the surface of the object. The gradient of $\widetilde{X}_c^{i,j}$ will therefore only have contributions from the vertices spanning the triangle face that intersects the ray. We can compute the derivative for one of these three vertices ($\mathbf{p}_a$) as follows

$$\frac{\partial \widetilde{X}_c^{i,j}}{\partial \mathbf{p}_a} = \left(\frac{\mathbf{p}_1 - \widetilde{X}_c^{i,j}\mathbf{p}_3}{q_3} \cdot \mathbf{d} \right) \frac{\lambda_a \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}, \tag{6}$$

where $\mathbf{d}$ is the ray direction, $\mathbf{n}$ is the normal of the face, and λ_a is the barycentric coordinate corresponding to $\mathbf{p}_a$. The equations for the remaining two vertices, $\mathbf{p}_b$ and $\mathbf{p}_c$, use λ_b and λ_c but are otherwise identical. For a derivation of Equation 6, see Appendix A.

As mentioned we use gradient descent to update the vertex positions, that is

$$\mathbf{v}_{i+1} = \mathbf{v}_i - \alpha_i \nabla \mathcal{L}(\mathbf{v_i}), \tag{7}$$

where $\mathbf{v}_i$, and $\mathbf{v}_{i+1}$ are the vertex positions in the i^{th} and $(i+1)^{\text{th}}$ iterations respectively. To choose the steplength α_i we use a simple backtracking line-search strategy to choose

$$\alpha_i := \frac{1}{2^n}\alpha, \tag{8}$$

where α is a fixed constant and n is largest integer such that $\mathcal{L}(\mathbf{v}_{i+1}) < \mathcal{L}(\mathbf{v}_i)$ when doing the update.

3.3 Initializing the optimization

Up until this point, we have described how the iterative part of the optimization problem works, but this is only one half of the problem. A good initial guess is extremely important to ensure convergence. In the next two sections, we will introduce two possible ways to obtain an initial guess for the minimization problem in Equation 1.

3.3.1 Using other reconstruction methods

One way of getting a good initial guess is using a reconstruction found via another reconstruction method. In this way, our method can be seen as a post-processing step which tries to adjust the reconstruction to better fit to the original image data. In some of our experiments we have used Screened Poisson Reconstructions (Kazhdan and Hoppe 2013) at various depths as initialization. When using Poisson reconstructions, we experience it often being beneficial to remesh the mesh before starting the optimization.

3.3.2 Using simple shape

Another way of getting a good initial guess is by solving a related, but simpler, minimization problem, and use the result as initialization for the minimization problem in Equation 1. If we denote the x-coordinates of the projector from the ground truth images by $\mathbf{X}_c$, we can solve the simpler problem given by

$$\underset{\mathbf{v}}{\operatorname{argmin}} \sum_{c \in \mathcal{C}} \left\| \mathbf{X}_c - \widetilde{\mathbf{X}}_c(\mathbf{v}) \right\|_F^2, \tag{9}$$

and use the solution as initialization of our problem in Equation 1. The method used to recover $\mathbf{X}_c$ depends on the patterns displayed, but for two sets of phase shifted patterns the heterodyne principle can be used (Reich et al. 1997).

To make the initial problem simpler, we start with a mesh that has few vertices and gradually increase the number of vertices by remeshing. This enables us to use a simple shape, e.g. a sphere as our initial mesh.

3.3.3 Remeshing

As described in Sections 3.3.1 and 3.3.2 we use a remeshing algorithm. The algorithm we use is adapted from the Python geometry processing library Pymesh (Zhou 2020). The outline of the algorithm is shown in algorithm 1.

Algorithm 1: Remeshing algorithm adapted from Zhou (2020)

Input: Mesh, target edge length ℓ
Result: Mesh
Remove degenerate triangles
Split edges longer than ℓ
while *mesh is changed* **do**
  Collapse edges shorter than ℓ
  Split obtuse triangles (angle bigger than 150°)
Remove self-intersections
Remove duplicate faces
Replace mesh with outer hull of mesh
Remove duplicated vertices
Split obtuse triangles (angle bigger than 179°)
Remove isolated vertices

4 Experiments

To demonstrate the usefulness of our method, we have carried out a few experiments, that we will briefly introduce. First, we show that our method is able to reconstruct an object starting from a sphere, which is in Figure 1. Secondly, we reconstruct 3 different objects at multiple levels of noise, which we show in Figure 3. Finally, we compare against a Poisson reconstruction at two levels of noise, and show that our method is able to produce sharp edges.

To understand how we did these experiments, we will go through how we generate our ground truth images.

4.1 Generating ground truth images

We did all our experiments using synthetic ground truth images, to have access to the ground truth shape of the mesh to compare against. Our ground truth images are made by projecting two sets of phase shifted patterns with 15 and 16 periods respectively, with 16 shifts of the first pattern and 8 shifts of the second, such that

$$n_p = \begin{cases} 15 & p \in [1, 2, \dots, 16] \\ 16 & p \in [17, 18, \dots, 24] \end{cases} \tag{10}$$

and

$$\phi_p = \begin{cases} 2\pi p \frac{1}{16} & p \in [1, 2, \dots, 16] \\ 2\pi (p-16) \frac{1}{8} & p \in [17, 18, \dots, 24]. \end{cases} \tag{11}$$

4.1.1 Rendering

The ground truth images are rendered using ray-tracing with 100 samples per pixel for anti-aliasing, and we use the Lambertian reflectance model to describe the surface of the mesh.

4.1.2 Noise

As these ground truth images are noise-free, we add noise to make the images more realistic. For this, we model the noise of a pixel with intensity x by a Gaussian distribution with mean x and variance

$$\sigma^2 = \sigma_r^2 + x\sigma_p^2, \tag{12}$$

where the first term σ_r describes the signal-independent sensor read-out noise and the second term $x\sigma_p$ describes the signal-dependent shot noise. We choose σ_r and σ_p by using the noise levels from a baseline camera (Adobe Systems Incorporated 2019). Our noise levels are then defined as multiples of this baseline noise level, controlled by k as follows

$$\sigma^2(k) = k(4.5 \cdot 10^{-7} + x \cdot 2 \cdot 10^{-5}), \tag{13}$$

such that $k = 1$ gives the noise levels of a baseline camera for $x \in [0; 1]$. If a pixel gets a value outside the $[0; 1]$ range after adding noise, we clip it to be inside the range. Examples of images for different values of k can be seen in Figure 3.

4.2 Quantitative evaluation

We quantitatively evaluate the performance of our method by measuring the symmetric volume difference between the ground truth and a reconstruction as a percentage of the volume of the ground truth. We refer to this as Δ_V and compute it as

$$\Delta_V = \frac{\left| \left(S \setminus \tilde{S}\right) \cup \left(\tilde{S} \setminus S\right) \right|}{|S|}, \tag{14}$$

where S and $\tilde{S}$ are the ground truth and reconstruction considered as solids, and $|\cdot|$ is the volume of a solid.

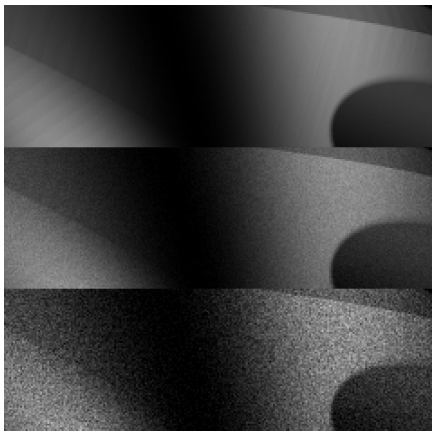


Fig. 3: Crop of image shown with varying levels of noise. From top to bottom: $k = 1$, $k = 10^2$, $k = 10^3$.

	Bunny	Box	Vase
Reconstruction	3 350	6 000	2 300
Ground truth	34 817	4 619	44 852

Table 1: Approximate number of vertices of each of the objects in Figure 5.

4.3 Experiment details

We evaluate our method on three different shapes. The Stanford Bunny (Turk 2000), a combination of a cylinder and a box with various truncated corners, and a dandelion vase (Steve 2013). These shapes are on the bottom row of Figure 3. When starting from a simple shape as described in subsubsection 3.3.2, we have used the same sphere across all of our experiments. When optimizing the simpler problem in Equation 9, we have done remeshing every 25^{th} iteration. At each remeshing step we decrease the target edge length which yields meshes with finer and finer resolution. For the i^{th} remeshing step we set the target to be

$$\ell_i = 0.99^i \cdot 0.025 \cdot d_{BB}, \tag{15}$$

where d_{BB} is the largest diagonal of the bounding box of the current mesh. To be able to solve the problem in Equation 9 we have estimated $\mathbf{X}_c$ using the heterodyne principle (Reich et al. 1997). In all of our experiments, we have used 60 camera-projector positions organized in 3 circles to mimic a structured light scanner with the object placed on a turntable, with the object being rotated three times. One of these circles are visualized in Figure 4. The camera resolution for all renderings are 1920×1080 pixels.

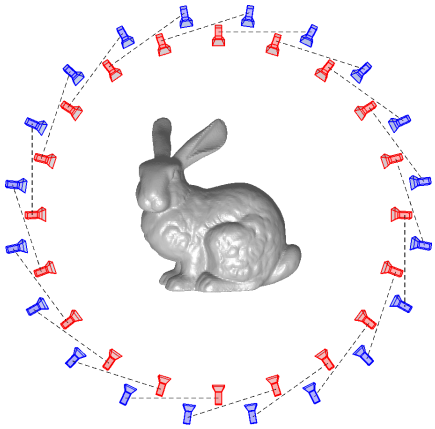


Fig. 4: Visualization of one of the circles from our camera-projector setup. In all experiments we use 3 circles with 20 cameras each. Red objects indicate a camera and blue indicates a projector. The dotted lines show which cameras and projectors belong together.

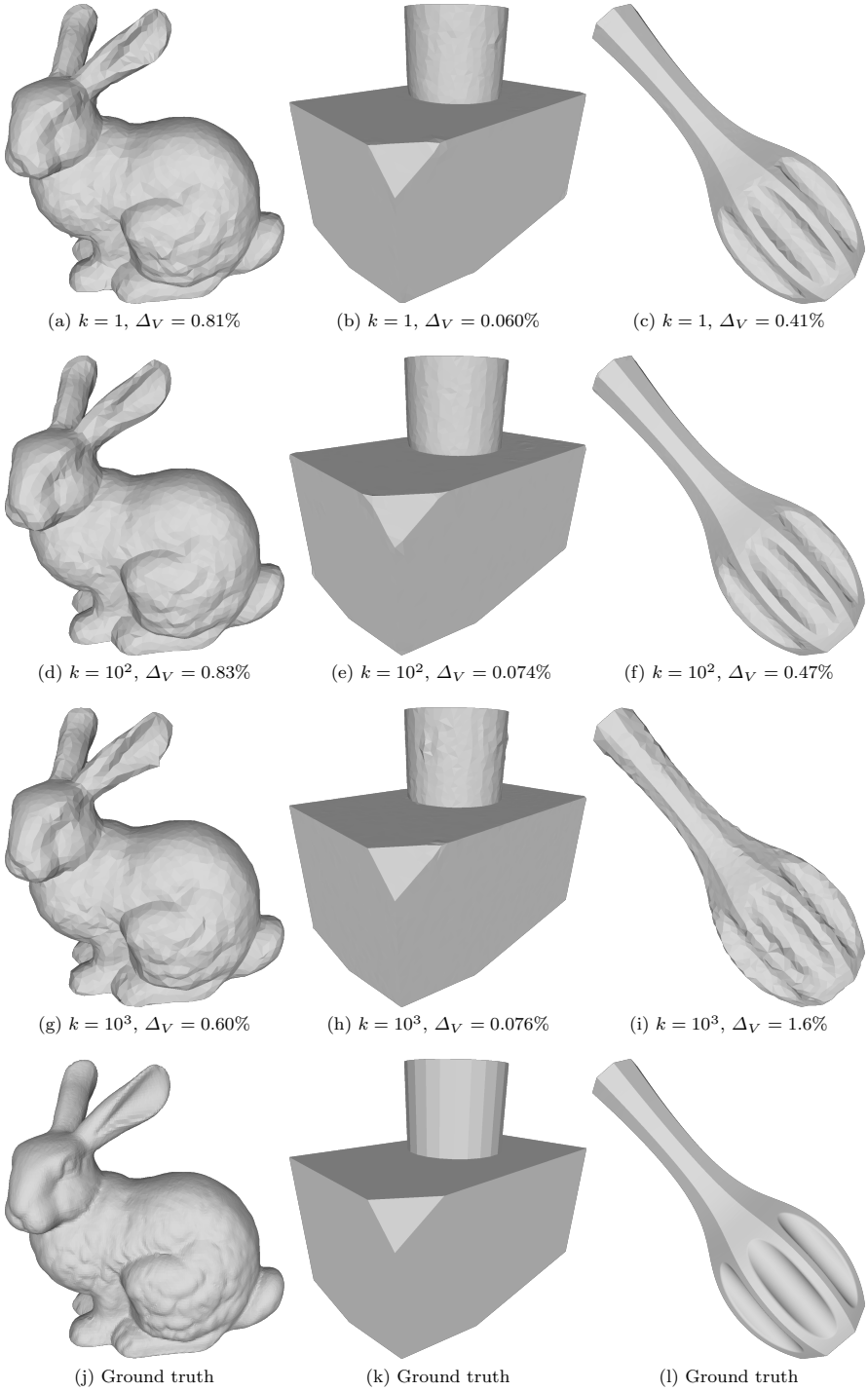
In Figure 1, we show how our method is able to reconstruct the Stanford Bunny starting from a sphere. After finishing the optimization based on Equation 9, we remesh the result by halving the target edge length ℓ from the last remeshing step, to get a mesh with even finer resolution. The two in-progress images shown, are during the initial optimization directly on $\mathbf{X}_c$. The final reconstruction contains many of the fine details of the true bunny (Figure 5j).

To examine how our method is affected by noise, we reconstruct all three objects starting from a sphere with varying levels of noise ($k \in [1, 10^2, 10^3]$), which we show in Figure 3. We see that our method largely unaffected by noise, but is still able to reconstruct the shape. The details lacking in our reconstructions are mostly due to us not having made the mesh fine enough to represent these details, which can also be seen in Table 1.

Finally, we compare against Screened Poisson reconstruction in Figure 3, for multiple depths of the reconstruction. It can be seen that our method is able to get a lower error than the Poisson reconstruction. The meshes from some of these data points are shown in Figure 2.

5 Discussion

Our method has implicit assumptions about the reflectance of the object that is necessary for structured light. These are that there are no phenomena such as subsurface scattering, or inter-reflections of light on the



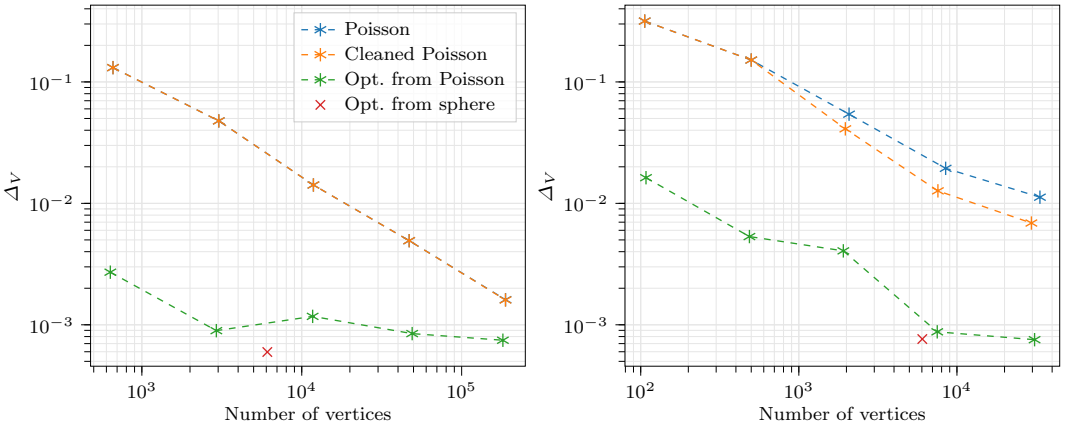
(a) $k = 1$, $\Delta_V = 0.81\%$ (b) $k = 1$, $\Delta_V = 0.060\%$ (c) $k = 1$, $\Delta_V = 0.41\%$

(d) $k = 10^2$, $\Delta_V = 0.83\%$ (e) $k = 10^2$, $\Delta_V = 0.074\%$ (f) $k = 10^2$, $\Delta_V = 0.47\%$

(g) $k = 10^3$, $\Delta_V = 0.60\%$ (h) $k = 10^3$, $\Delta_V = 0.076\%$ (i) $k = 10^3$, $\Delta_V = 1.6\%$

(j) Ground truth (k) Ground truth (l) Ground truth

Fig. 5: Reconstructions made by our method on three different objects, for increasing levels of noise k.



(a) $k = 1$ (b) $k = 10^3$

Fig. 6: Comparison showing Δ_V as a function of the number of vertices in the mesh for reconstructions of the box with cylinder. Cleaned Poisson is the Poisson reconstruction, where only the connected component with the largest volume has been kept. Opt. from Poisson and sphere is our method using the initial guesses described in subsection 3.3. The Poisson reconstruction was done for spatial octree depths of 4 to 8.

object. This is because our method is also relying on the property that the projector light observed in a single pixel is only coming from a single x-coordinate of the projector, which is not true when these effects are present.

Although we have used the Lambertian reflectance model to render our synthetic data, we do not expect this to be a limitation of our method. It does not rely on the assumption of Lambertian reflectance due to the use of structured light, and therefore does not rely on estimating a texture map of the object, neither implicitly nor explicitly.

In our experiments, we have had the advantage of knowing the camera and projector positions exactly, which is not the case when working with real data. This can however be remedied relatively easily, by including the positions of these as parameters in the optimization problem. Additionally, we do not need to purely rely on our estimates of $\mathbf{A}_c$, $\mathbf{B}_c$, as these can also be optimized for. We do however expect both $\mathbf{A}_c$, $\mathbf{B}_c$, and the camera-projector positions to be known quite accurately, and would suggest letting them be part of the optimization only once the original optimization problem has converged.

In order to solve the optimization problem, we compute the derivative of our loss function, which involves the derivative of our rendering. This is potentially problematic as our rendering is not differentiable at points where non-neighbouring faces of the mesh are bordering each other in the image space. This could e.g. be the ear and body of the Stanford Bunny. However, in practice this turns out not to be a problem in our optimization. We suspect this is due to the fact that this is something that happens in a relatively small percentage of pixels each iteration.

The choice of using a mesh as the surface representation to optimize has some advantages. It is very efficient to compute the gradient of our loss function for a mesh, as each pixel only influences a constant number of elements in the gradient, which makes it suitable for parallel implementation on a GPU.

A disadvantage of our method is that it is not able to handle topology changes. In practice this means, that the initial mesh must have the same topology as the true object.

6 Conclusion

Our primary contribution is a novel model for computing a surface mesh directly from structured light images without the need for an intermediate point cloud. With this model, we have shown that the direct computation of a triangle mesh gives higher accuracy and is particularly good at reconstructing sharp features such as corners and edges, which are smoothed out using the two-step Screened Poisson reconstruction that we compare to. Further, we have obtained very high robustness to noise by optimizing the vertex positions directly from

the images and and not reconstructed from a computed point cloud, where intensity information is lost. Finally, our approach completely avoids partial scans that must subsequently be aligned, because the optimization is done for all images at once. This demonstrates the advantage of directly reconstructing a surface from structured light images using differentiable rendering.

References

Adobe Systems Incorporated. 2019. *Ditial Negative (DNG) Specification.* `www.adobe.com/content/dam/acom/en/products/photoshop/pdfs/dng_spec_1.5.0.0.pdf`

Ha, H., Oh, T.-H., and Kweon, I. S. 2015. A multi-view structured-light system for highly accurate 3d modeling. In *International Conference on 3D Vision (3DV 2015)*. IEEE, 118–126.

Horn, B. K. P. 1970. *Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object From One View.* Ph.D. Dissertation. Massachusetts Institute of Technology. MAC TR-79.

Horn, B. K. P. 1975. Obtaining shape from shading information. In *The Psychology of Computer Vision*, edited by P. H. Winston. McGraw-Hill, Chapter 4, pp. 115–155.

Inagaki, A., Tagawa, N., Minagawa, A., and Moriya, T. 2001. Computation of shape and reflectance of 3D object using moire/spl acute/phase and reflection model. In *International Conference on Image Processing (ICIP 2001)*, Vol. 2. IEEE, 177–180.

Isidoro, J. and Sclaroff, S. 2002. Stochastic mesh-based multiview reconstruction. In *First International Symposium on 3D Data Processing Visualization and Transmission.* IEEE, 568–577.

Isidoro, J. and Sclaroff, S. 2003. Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints.. In *International Conference on Computer Vision (ICCV 2003)*, Vol. 2. IEEE, 1335–1342.

Kazhdan, M. and Hoppe, H. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics* **32**(3): 1–13.

Liu, S., Li, T., Chen, W., and Li, H. 2019. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *International Conference on Computer Vision (ICCV 2019)*. 7708–7717.

Loubet, G., Holzschuch, N., and Jakob, W. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics* **38**(6): 228:1–228:14.

Naganuma, S., Tagawa, N., and Minagawa, A. 2003. Estimation of 3D shape and reflectance using multiple moiré images and shading model. In *ACM Symposium on Applied Computing (SAC 2003)*. 943–950.

Naganuma, S., Tagawa, N., Minagawa, A., and Moriya, T. 2004. Simultaneous determination of object shape and color by moire analysis using a reflection model. In *International Conference on Pattern Recognition (ICPR 2004)*, Vol. 3. IEEE, 202–205.

Reich, C., Ritter, R., and Thesing, J. 1997. White light heterodyne principle for 3D-measurement. In *Sensors, Sensor Systems, and Sensor Data Processing*, edited by O. Loffeld, Vol. 3100. SPIE, 236 – 244. `https://doi.org/10.1117/12.287750`

Rockwood, A. P. and Winget, J. 1997. Three-dimensional object reconstruction from two-dimensional images. *Computer-Aided Design* **29**(4): 279–285.

Steve. 2013. Dandelion Vase. `www.thingiverse.com/thing:157102`

Turk, G. 2000. The Stanford Bunny. `https://www.cc.gatech.edu/~turk/bunny/bunny.html`.

Wu, C., Wilburn, B., Matsushita, Y., and Theobalt, C. 2011. High-quality shape from multi-view stereo and shading under general illumination. In *Conference on Computer Vision and Pattern Recognition (CVPR 2011)*. IEEE, 969–976.

Yu, T., Xu, N., and Ahuja, N. 2004. Shape and view independent reflectance map from multiple views. In *European Conference on Computer Vision (ECCV 2004)*. Springer, 602–615.

Yu, T., Xu, N., and Ahuja, N. 2007. Shape and view independent reflectance map from multiple views. *International journal of computer vision* **73**(2): 123–138.

Zhang, L. and Seitz, S. M. 2000. Image-based multiresolution shape recovery by surface deformation. In *Videometrics and Optical Methods for 3D Shape Measurement (Proceedings of SPIE)*, Vol. 4309. 51–61.

Zhou, Q. 2020. Pymesh—geometry processing library for python. *Software available for download at https://github. com/PyMesh/PyMesh*

A Derivation of Equation 6

In this appendix, we derive the gradients of the elements of $\widetilde{\mathbf{X}}_c(\mathbf{v})$ wrt. the vertices $\mathbf{v}$. Let $\widetilde{X}_c^{i,j}$ be the i,j^{th} element of $\widetilde{\mathbf{X}}_c(\mathbf{v})$. This is found by tracing a ray through the center of the i,j^{th} pixel in the camera until it intersects the surface at a point $\mathbf{r}$, that is

$$\mathbf{r} = \mathbf{o} + t\mathbf{d}, \tag{16}$$

for some t, where $\mathbf{o}$ is the position of the camera and $\mathbf{d}$ is the direction of the ray. Modelling the projector as a pinhole camera, this point is projected back into the projector by

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & P_{1,4} \\ P_{2,1} & P_{2,2} & P_{2,3} & P_{2,4} \\ P_{3,1} & P_{3,2} & P_{3,3} & P_{3,4} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ 1 \end{bmatrix}. \tag{17}$$

If we define $\mathbf{p}_1 = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} \end{bmatrix}$ and $\mathbf{p}_3 = \begin{bmatrix} P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$ then

$$\widetilde{X}_c^{i,j} = \frac{q_1}{q_3} = \frac{\mathbf{p}_1 \cdot \mathbf{r} + P_{1,4}}{\mathbf{p}_3 \cdot \mathbf{r} + P_{3,4}}. \tag{18}$$

The point $\mathbf{r}$ intersect a triangle with vertices $\mathbf{p}_a, \mathbf{p}_b$, and $\mathbf{p}_c$. The normal of this triangle can be computed by

$$\mathbf{n} = (\mathbf{p}_c - \mathbf{p}_b) \times (\mathbf{p}_a - \mathbf{p}_c). \tag{19}$$

Using the normal we can write t as

$$t = \frac{(\mathbf{p}_b - \mathbf{o}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}. \tag{20}$$

Using the chain-rule we have that

$$\frac{\partial \widetilde{X}_c^{i,j}}{\partial \mathbf{p}_a} = \frac{\partial \widetilde{X}_c^{i,j}}{\partial t} \frac{\partial t}{\partial \mathbf{p}_a} \tag{21}$$

$$= \left(\sum_{i=1}^{3} \frac{\partial \widetilde{X}_c^{i,j}}{\partial p_i} \frac{\partial p_i}{\partial t} \right) \frac{\partial t}{\partial \mathbf{p}_a} \tag{22}$$

$$= \left(\frac{\partial \widetilde{X}_c^{i,j}}{\partial \mathbf{r}} \cdot \mathbf{d} \right) \frac{\partial t}{\partial \mathbf{p}_a}. \tag{23}$$

Here

$$\frac{\partial \widetilde{X}_c^{i,j}}{\partial \mathbf{r}} = \frac{\frac{\partial q_1}{\partial \mathbf{r}} q_3 - q_1 \frac{\partial q_3}{\partial \mathbf{r}}}{{q_3}^2} \tag{24}$$

$$= \frac{\frac{\partial q_1}{\partial \mathbf{r}} - \widetilde{X}_c^{i,j} \frac{\partial q_3}{\partial \mathbf{r}}}{q_3} \tag{25}$$

$$= \frac{\mathbf{p}_1 - \widetilde{X}_c^{i,j} \mathbf{p}_3}{q_3}, \tag{26}$$

The last missing term in Equation 23 is

$$\frac{\partial t}{\partial \mathbf{p}_a} = \frac{\partial}{\partial \mathbf{p}_a} \frac{(\mathbf{p}_b - \mathbf{o}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}} \tag{27}$$

$$= \frac{\frac{\partial}{\partial \mathbf{p}_a} \left[(\mathbf{p}_b - \mathbf{o}) \cdot \mathbf{n} \right] (\mathbf{d} \cdot \mathbf{n}) - \left[(\mathbf{p}_b - \mathbf{o}) \cdot \mathbf{n} \right] \frac{\partial \mathbf{d} \cdot \mathbf{n}}{\partial \mathbf{p}_a}}{(\mathbf{d} \cdot \mathbf{n})^2} \tag{28}$$

$$= \frac{\frac{\partial \mathbf{n}}{\partial \mathbf{p}_a}^\top (\mathbf{p}_b - \mathbf{o}) - t \frac{\partial \mathbf{n}}{\partial \mathbf{p}_a}^\top \mathbf{d}}{\mathbf{d} \cdot \mathbf{n}} \tag{29}$$

$$= \frac{\frac{\partial \mathbf{n}}{\partial \mathbf{p}_a}^\top (\mathbf{p}_b - \mathbf{o} - t\mathbf{d})}{\mathbf{d} \cdot \mathbf{n}} \tag{30}$$

$$= \frac{\frac{\partial \mathbf{n}}{\partial \mathbf{p}_a}^\top (\mathbf{p}_b - \mathbf{r})}{\mathbf{d} \cdot \mathbf{n}}, \tag{31}$$

with

$$\frac{\partial \mathbf{n}}{\partial \mathbf{p}_a} = \frac{\partial}{\partial \mathbf{p}_a} (\mathbf{p}_c - \mathbf{p}_b) \times (\mathbf{p}_a - \mathbf{p}_c) \tag{32}$$

$$= \frac{\partial}{\partial \mathbf{p}_a} (\mathbf{p}_c - \mathbf{p}_b) \times \mathbf{p}_a \tag{33}$$

$$= \frac{\partial}{\partial \mathbf{p}_a} \left[\mathbf{p}_c - \mathbf{p}_b \right]_\times \mathbf{p}_a \tag{34}$$

$$= \left[\mathbf{p}_c - \mathbf{p}_b \right]_\times, \tag{35}$$

where we utilise that a cross-product $\mathbf{a} \times \mathbf{b}$ can be written as a matrix-vector product $\left[\mathbf{a}\right]_\times \mathbf{b}$ using a skew-symmetric matrix. Inserting into Equation 31 we get

$$\frac{\partial t}{\partial \mathbf{p}_a} = \frac{(\mathbf{p}_b - \mathbf{r}) \times (\mathbf{p}_c - \mathbf{p}_b)}{\mathbf{d} \cdot \mathbf{n}}. \tag{36}$$

If we write $\mathbf{r}$ using barycentric coordinates, $\mathbf{r} = \lambda_a \mathbf{p}_a + \lambda_b \mathbf{p}_b + \lambda_c \mathbf{p}_c$ with $\lambda_a + \lambda_b + \lambda_c = 1$, then Equation 36 reduces to

$$\frac{\partial t}{\partial \mathbf{p}_a} = \frac{(\mathbf{p}_b - (\lambda_a \mathbf{p}_a + \lambda_b \mathbf{p}_b + \lambda_c \mathbf{p}_c)) \times (\mathbf{p}_c - \mathbf{p}_b)}{\mathbf{d} \cdot \mathbf{n}}$$

$$= \frac{(\lambda_a (\mathbf{p}_c - \mathbf{p}_a) + (1 - \lambda_b)(\mathbf{p}_b - \mathbf{p}_c)) \times (\mathbf{p}_c - \mathbf{p}_b)}{\mathbf{d} \cdot \mathbf{n}}$$

$$= \frac{\lambda_a (\mathbf{p}_c - \mathbf{p}_a) \times (\mathbf{p}_c - \mathbf{p}_b)}{\mathbf{d} \cdot \mathbf{n}}$$

$$= \frac{\lambda_a \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}. \tag{37}$$

Putting it all together we get that

$$\frac{\partial \widetilde{X}_c^{i,j}}{\partial \mathbf{p}_a} = \left(\frac{\mathbf{p}_1 - \widetilde{X}_c^{i,j} \mathbf{p}_3}{q_3} \cdot \mathbf{d} \right) \frac{\lambda_a \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}. \tag{38}$$

The derivatives wrt. $\mathbf{p}_b$, and $\mathbf{p}_c$ can be found with similar derivations.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ