Hardware Design and Accurate Simulation of Structured-Light Scanning for Benchmarking of 3D Reconstruction Algorithms: Technical Design Document

SEBASTIAN KOCH*, Technische Universität Berlin, Germany
YURII PIADYK*, New York University, USA
MARKUS WORCHEL, Technische Universität Berlin, Germany
MARC ALEXA, Technische Universität Berlin, Germany
CLAUDIO SILVA, New York University, USA
DENIS ZORIN, New York University, USA
DANIELE PANOZZO, New York University, USA

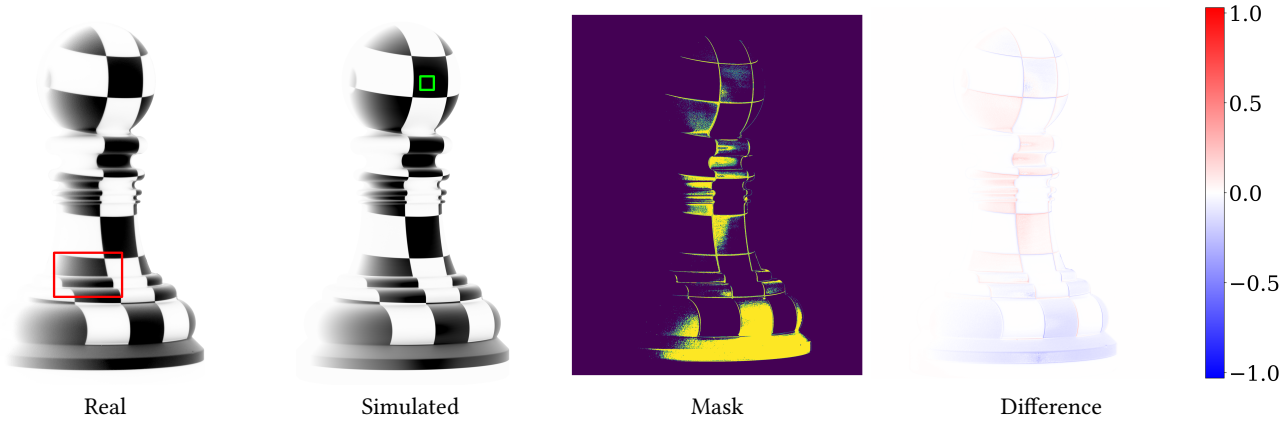


Fig. 1. From left to right: the image of the *Pawn* (inverted colors) acquired by a camera is faithfully reproduced by our scanning simulator after hardware calibration. The mask shows that pixel-wise difference rarely exceeds 5% of the average brightness in the middle of the white checker highlighted by the green rectangle which is used to normalize the images (i.e. 100%, or 1.0). A close-up view of the highlighted red region shown in Figure 10. The height of this model is 152.5 mm.

We co-developed a 3D structured light scanning hardware setup together with a corresponding light transport simulation with the objective of minimizing the difference of the images on a per-pixel level (Figure 1). Such scanner simulator is an ideal test-bed for developing data-driven reconstruction algorithms, as it allows generating synthetic datasets that match the result of scanning, enabling the algorithms trained on synthetic data to generalize onto real scanner.

In this document, we provide detailed description of our scanning setup and the corresponding simulator. We start with a Structured-Light Scanning (SLS) overview and discuss different factors driving our decisions and design choices, both for physical setup and the simulator.

General discussion in the main section is accompanied by four appendices on hardware design, parameterization and calibration, reconstruction, and dataset generation, with in-depth explanation of implementation details and code references. All the data can be found at https://archive.nyu.edu/handle/2451/62251 and the software repository is hosted at https://github.com/geometryprocessing/scanner-sim.

*Both authors contributed equally to this work.

1 HARDWARE SETUP AND SIMULATOR

A structured-light scanning setup is composed of 3 main components: a camera C, a projector P, and the object being scanned O, which can be optionally placed on a moving stage S (Figure 2).

We refer to Lanman and Taubin [Lanman and Taubin 2009] for a more detailed introduction to structured-light scanning. In the following we provide a brief reminder of the general approach and provide the details of our design decisions in view of the acquisition-simulation system as a whole. Our aim is to build the system so that its simulation can be performed as accurately as possible. Also, to support reproducibility and adoption, we restrict the design to readily available components.

SLS Primer. Assuming that the position of all the objects in the scene is known to sufficient accuracy, a 3D object can be reconstructed by illuminating a single pixel of the projector at a time, detecting the location of the illuminated point on the camera sensor and then triangulating. This procedure is impractically slow. By projecting a set of coded patterns (Figure 3) it is possible to establish correspondences between camera and projector pixels from a

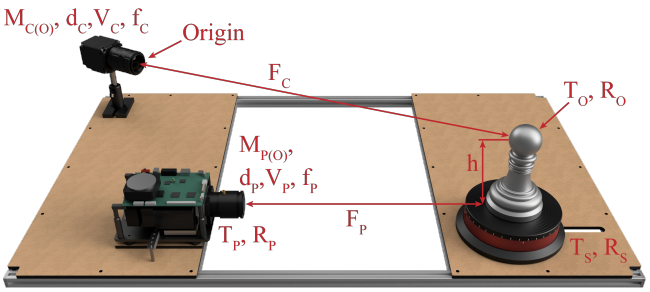


Fig. 2. A diagram illustrating our structured-light scanner setup and the parameters associated with each component (see Table 1 in Appendix A for a detailed explanation).

small set of images [Couture et al. 2011; Gupta and Nayar 2012]. However, this adds additional algorithmic complexity for decoding the patterns to compute the correspondences.

The accuracy of the reconstruction depends on many factors, including the resolution of the camera and projector, the lenses used, and the accuracy of the estimation of the relative position of camera and projector. Finding this set of parameters is called calibration of the scanner, which is usually performed by 'scanning' objects of known geometry.

In this work, we carefully analyze each component of the scanning system, selecting hardware components to minimize the noise that we cannot simulate, and we propose a corresponding calibration procedure with the goal of minimizing reconstruction errors. Different from existing approaches, we do not strive to make the calibration procedure simple and/or efficient, our goal is solely on minimizing effects that cannot be recovered by optimization methods in the computational part of the system.

Overview. We systematically consider each component of our scanning setup explaining: (1) why is it important and how does it affect reconstruction error, (2) the specific hardware we selected, (3) how the component is parametrized in the synthetic setting, (4) the protocol we use to calibrate these parameters using a combination of physical measurements and numerical optimization, and (5) how the model and parameters are used in the physically based renderer Mitsuba [Jakob 2010]. We then perform a series of experiments (see main paper) to validate our choices, quantifying the errors by pixel-wise difference of real-images with synthetic renderings, both on calibration objects (i.e. with geometry know up to machining tolerances) as well as unknown scenes.

For brevity, we focus on (1) and (5) in this section and provide a high level summary of (2-4) where necessary as well as a flow chart of the calibration pipeline (Figure 4). The details on (2-4) are available in the appendices.

1.1 Calibration Objects

To build a highly accurate structured-light scanner we need the following objects for calibration:

- A *CharuCo board* [Garrido-Jurado et al. 2014] for accurate calibration of the camera and projector geometry.

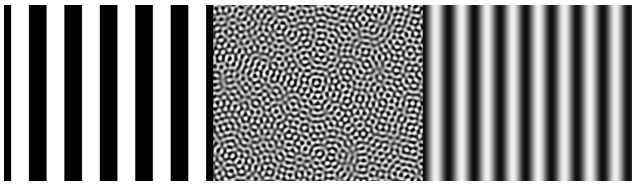


Fig. 3. The three different light coding schemes (from left to right, Gray codes [Sato and Inokuchi 1985], unstructured-light [Couture et al. 2011], and micro phase shifting [Gupta and Nayar 2012]) that we used in our structured-light scanning pipeline. Other schemes can be easily added.

- A *linear stage* capable of accurately reproducing positions for the CharuCo calibration board with known intervals to measure the focus distance and the aperture for both the camera and the projector.
- A *Spectralon®* for the radiometric calibration of camera and projector.
- A set of accurate calibration objects (Figure 12).

Figure 4 is showing the interdependencies of different scanner parameters (Table 1) with regard to how they are being acquired. At this stage, we are directly measuring the coating thickness d with a micrometer (before and after applying the coating).

Rendering. The CharuCo board is modeled as a properly sized slab (checkers with size S_C) with colored vertices placed in accordance with the markers of the physical board. The linear and rotation stage are defined by a translation T_S and rotation R_S of the calibration board. The calibration and validation objects are densely triangulated 3D models extracted from the CAD models using OpenCascade. In general, textured or colored OBJ and PLY objects can be loaded into the simulation. We disable the interpolation of surface normals to support sharp edges, corners and creases, which are common in machined objects.

1.2 Camera

The camera is a central component in the setup. All the other components are located/calibrated based on images obtained with the camera. We want to maximize its resolution (in pixels per mm) at a focus distance where the object of interest is located, while maintaining a sufficient field and depth of view. This value is a function of multiple parameters of the camera, the most relevant being the pixel size, focal length, sensor size, aperture, and actual focus distance. While the influence of each independent parameter on the final resolution is known, one cannot choose an arbitrary combination because they are interconnected. A compromise must thus be made. For example, increasing the focal length improves the desired resolution only up to reaching the diffraction limit of the lens and narrow depth of field due to the finite aperture size. We have chosen a 31.4 MP camera with 50 mm lens.

To calibrate the camera (Figure 4, 1) with subpixel accuracy (the average reprojection error amounted to 0.9 pixels) we used the CharuCo board and the OpenCV [Bradski 2000] implementation of the algorithm by Zhang et al. [Zhang 2000].

Another important factor is noise the dynamic range of the camera. Since we need to work with projected patterns, possibly bright

specular reflections and low ambient light levels, a high dynamic range (HDR) image capture is required [Debevec and Malik 1997]. We typically acquire 6 images with different exposure times and combine them into a single HDR image by calculating the total amount of light recorded by each pixel and dividing it by the total exposure time. This significantly reduces the amount of image noise (no need to simulate) and we also subtract corresponding dark frames and replace hot/stuck pixels.

Rendering. The camera is simulated with a thin lens perspective camera model, a common choice to simulate aperture and depth of field. In Mitsuba, this model is implemented by the *thinlens sensor* (Section 8.9.2 of the documentation). As in the physical setup, the camera frame is aligned with the world frame (z-axis pointing forward, x-axis pointing to the right and y-axis to the bottom). We calculate the vertical (or alternatively horizontal) field of view from the calibration matrix M_{CO} and the image width W_C and height H_C with the function `OpenCV:calibrationMatrixValues`. The aperture f_C and focus distance F_C were translated to scene units and plugged into the thin lens model. Similarly, the pixel ratio and the shift of the optical axis from the image center were derived from M_{CO} and applied as scaling and cropping to the virtual image sensor.

To match the images of the physical camera, we render high dynamic range images with the same resolution $W_P \times H_P$ (covered by Mitsuba *hdrfilm*, see Section 8.12.1 in the documentation). Besides the color or luminance information in each pixel, we also read and store ground truth depth and distance values from the ray intersection records (Mitsuba *field extraction integrator*, Section 8.10.18 in the documentation) in the resulting OpenEXR images.

As we have recorded the distortion coefficients d_C and vignetting offset V_C in the hardware calibration, it would be possible to apply them to the simulated camera as well. However, we opted for removing them directly in the recorded images, as this is what is usually done for 3D scanning applications, because the distortion/vignetting might otherwise affect the reconstruction. Therefore, d_C and V_C are not used in the renderer.

1.3 Lights

We assume the scene is lit only by two sources of light: the projector and ambient light, which include parasitic light of the projector reflected back into the scene by the surrounding environment.

The *projector* is chosen with considerations similar to the camera: maximizing spatial resolution without sacrificing too much field of view or depth of field. An additional important consideration is the ability to focus at a close distance. DLP projectors (based on a Digital Micromirror Device) have an advantage over LCD because of their superior contrast and absence of color bleeding [Inoshita et al. 2013]. Colors in such projectors are obtained via temporal multiplexing of the RGB components instead of spatially separated color filters, which restricts the range of possible camera exposures to multiples of 1/refresh rate of the projector to avoid color artifacts (or temporal inconsistency for monochrome camera like ours).

Ambient light is used while acquiring the images of the CharuCo board during calibration of the projector and of the rotating stage. The brightness of the ambient lights should be lower than the projector so that they (together with the parasitic light of the projector)

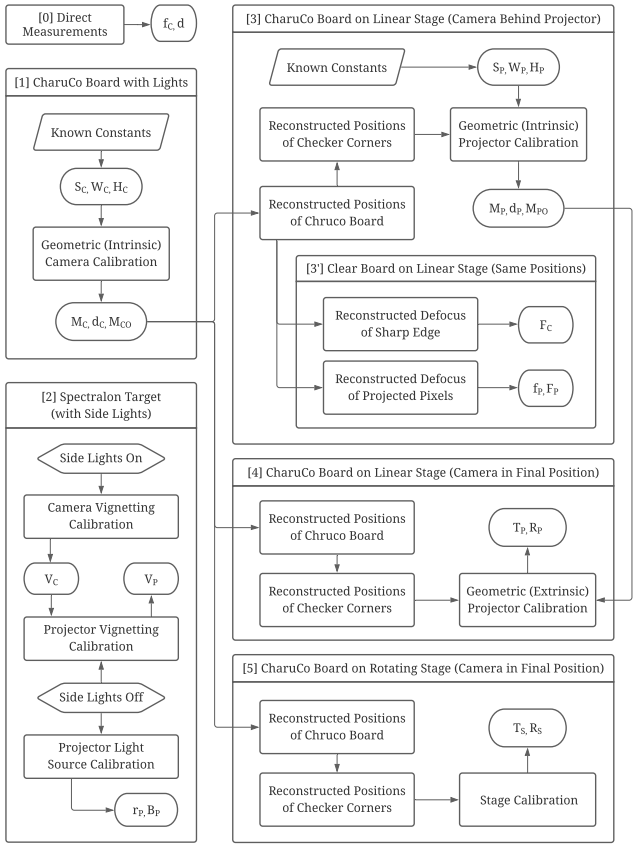


Fig. 4. Diagram of the dependencies of parameters during the calibration sequence. See Table 1 for a description of the symbols.

can be removed by subtraction to obtain a clean image of the object illuminated by projected pattern only.

The parametrization of the projector is similar to the one used for the camera, but with additional extrinsic parameters R_P and T_P. The calibration of the projector (Figure 4, [3]) is done with the help of the camera by projecting a checker pattern onto a CharuCo board (Figure 14). One can reconstruct the positions of checker corners of the projected pattern for different positions of the CharuCo board and combine them into a single "3D calibration object" to use for the estimation of the intrinsic projector parameters (see Appendix B.3 for more details). We use a linear stage to reproduce the same positions for the CharuCo board (Figure 4, [3]) but now with the clear side facing forward. This enables us to measure the projector focus distance F_P and aperture f_P (Figure 14, right). The same experiment is used to refine the value of the camera focus distance F_C, which was initially estimated from lens markings.

Rendering. To simulate the physical projector and its effects on the scanning procedure, we extend a spot light source (Mitsuba *spot emitter* see Section 8.8.3 of the documentation). The position and orientation of this light source are converted from T_P and R_P to scene units (meters instead of millimeters). For radial distortion and vignetting, we apply the measured distortion coefficients d_P and vignetting levels V_P to the projected patterns before projecting them

into the scene. This simulates the distortion and vignetting effect caused by the optical path in the physical projector. We also map the pre-distorted patterns using the measured projector response function r_P prior to simulation. The patterns are read in as images of size $W_P \times H_P$. To minimize interpolation artifacts occurring during the predistortion procedure, it is also possible to feed in larger pattern images as long as they have the same aspect ratio between width and height. We tested our procedure with the patterns shown in Figure 3, but arbitrary patterns are supported. The field of view is derived from the calibration matrix M_{PO} and the image width W_P and height H_P with the function `OpenCV::calibrationMatrixValues`.

Depth of field or defocusing of the projector light is a significant source of noise in structured-light scanning. We therefore extend the spot emitter to support depth of field effects. This is done similar to the thin lens camera model and was also already used by [Berger et al. 2013]. The modified spot emitter supports focus distance and aperture, which are plugged in from F_P and f_P after translation to scene units. In addition, we apply the measured pixel ratio and shift of optical axis read from M_{PO} by padding the pattern images and shifting the optical center. To match the intensity levels of the virtual projector and the physical projector, we use gradient descent as described in Section 1.5. The diffuse ambient light of the physical setup is also matched by a diffuse light source in simulation (Mitsuba *constant emitter* see Section 8.8.10 of the documentation).

1.4 Scanner Geometry

The geometry of the scanner is defined by the position of the projector (T_P, R_P) and of the rotating stage (T_S, R_S) relative to the camera. The angle between the optical axes of the camera and projector is a defining factor for the performance of the scanner. Small angles lead to decreased reconstruction accuracy. Large angles lead to shadows on large regions of the scanned object. As a compromise, we settle for a commonly used angle of 30 degrees.

To calibrate the extrinsic projector parameters, we repeat the same procedure as for intrinsic but with the camera in its final position and not behind the projector (see Figure 14, 4 vs 3). We enforce the already known intrinsic parameters and only keep the extrinsic ones because the intrinsic ones were found in a more favourable, and thus accurate, setup. The average re-projection error achieved is less than a pixel and more in-depth analysis of the overall scanner/simulator accuracy is shown in Figure 15 (right).

The rotating stage calibration (see Figure 14, 5) is following the same procedure with the only difference that the CharuCo board is now placed in the rotating stage instead of on the linear one and we are interested in reconstructing its axis of rotation based on the reconstructed positions of CharuCo markers. We can reconstruct the full frame of reference of the rotating stage by scanning a Pawn calibration object with know ball height h (knowing the axis of rotation is sufficient for the registration of scans from different angles). This information will be useful later on to localize other calibration objects.

Rendering. All parameters describing the scanner geometry are measured in the hardware calibration and can be directly translated to scene units and applied to the projector and object as rotations and translations.

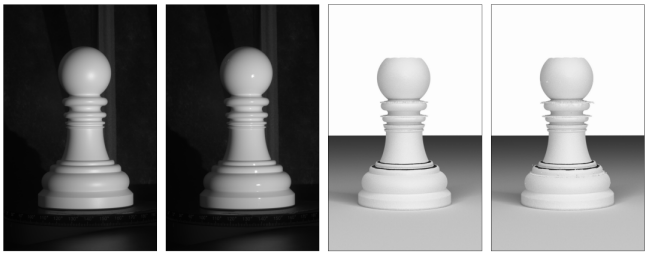


Fig. 5. From left to right: matte coated pawn, glossy coated pawn, matte reconstruction, glossy reconstruction. The difference between the scanned results is minor, the specularity of the material does not introduce artifacts when HDR images are used for scanning.

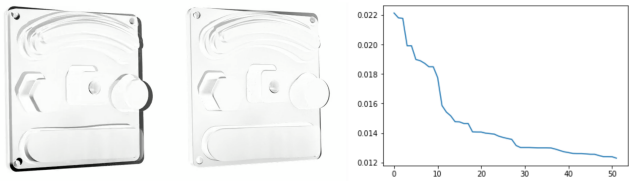


Fig. 6. Absolute pixelwise difference before (left) and after (middle) running the optimization for object position and orientation. White represents no difference, black the maximal difference. The mean squared error between the image from the physical setup and an initial guess decreases during the optimization procedure (right).

1.5 Materials

So far, we focused on the geometric aspects of the scanner. To accurately simulate images produced by the camera in a real-world setup, one also has to perform a radiometric calibration of both camera and the projector as well as match the material properties of the object being scanned/rendered. We are using a digital camera sensor with global shutter. We confirmed it has the expected linear radiometric response (e.g. gamma=1.0). This is not the case for the projector, so we acquired its response function r_P experimentally. We also calibrate the white balance of the projector and the vignetting for both camera and projector lenses as this is a significant cause of image brightness attenuation towards the borders.

We also evaluate the performance of the system for both matte and glossy materials. The reconstruction results of a scan with matte and glossy material coating are shown in Figure 5.

Getting material rendering right is heavily dependent on the radiometric calibration of the setup. We are using a Spectralon® for the calibration of the camera (V_C) and projector (V_P) vignetting. We measure projector response function r_P and white balance B_P using the same setup but with side lights off (Figure 4, 2). More details can be found in Appendix B.5.

Rendering. Matching the materials of the physical and virtual objects could be done by BRDF aquisition of the physical material. However, as the material of the calibration objects is relatively simple, we match it by choosing an equivalent material (Mitsuba *roughplastic material* [Cook and Torrance 1982], Section 8.2.8 of the documentation) and optimizing the relevant material parameters p (roughness, interior index of refraction and diffuse reflectance) using

gradient descent and luminance difference (MSE) as a loss function. For this, we acquired scans of a planar calibration object coated with the same material as the other calibration objects. In addition to the obtained material parameters, we measured the thickness d of the coating and added this coating $d = 0.075mm$ to the calibration object geometry before simulation. For the projector defocus calibration, we used the planar calibration object scanned at two different angles to the projector and ran the same optimization procedure to obtain the matching values for projector aperture f_P and focus distance F_P.

Despite the extensive calibration, there might still be some differences between the hardware scanner and the simulation due to additional sources of noise that we did not account for, or limitations in the physical models we use. We use the same gradient descent optimization procedure to further refine other parameters of the setup too (e.g. object localization). Figure 6 shows such an optimization for finetuning of the position T_O and orientation R_O of a scanned object. For many parameters (such as aperture f_C, f_P and focus distance F_C, F_P) the optimization procedure finds a local minimum in less than 100 iterations. However, for optimizations (e.g. of the object position T_O and object rotation R_O) where the initial guess is too far from an optimum or when there are too many parameters optimized at once, it takes much longer or even runs into unfavorable local minima. It is therefore important to start with an adequate initial guess.

2 RELATED WORK

Our work uses Mitsuba [Jakob 2010], a physically based renderer, with additions to support the simulation of a projector. Our contribution is independent of particular mathematical approximations of light transport or their computational implementation. We refer an interested reader to [Pharr et al. 2016] for details on physically based rendering techniques. Likewise, since we focus on pixel-wise difference for measurement of such techniques, we note that [Ulbricht et al. 2006] offer a survey on other experimental validation approaches for physically-based rendering techniques. For a survey on reflectance (and geometry) acquisition we recommend [Weinmann and Klein 2015].

Scanning Hardware. The design of structured light 3D scanners has been a topic of research for several decades. Starting in the early 2000s, assembling good quality scanners out of commodity parts was a challenging but achievable task (see, e.g., Rocchini et al. [Rocchini et al. 2001]). Lanman and Taubin's SIGGRAPH course [Lanman and Taubin 2009] (which includes open-source code) is an excellent resource, and it has greatly simplified the process of building a working 3D scanner. The common objective had been to optimize the ratio of scanning quality and cost, and time investment required for calibration. In this work, we build a scanner selecting components with the unique goal of reducing the error relative to realistic image synthesis. The process of building such a scanner-simulator pair, detailed in Section 1, is challenging, and to the best of our knowledge, has not been done.

Berger et al. [Berger et al. 2013] introduced a benchmark for surface reconstruction algorithms. It included a synthetic 3D range scanner that could be used to simulate different aspects of the scanning process, such as varying the scanning noise and sampling rates. To validate their synthetic scanner, they 3D printed a a single object (the Gargoyle model) and scanned it with a physical scanner (a NextEngine 3D Laser Scanner). They do not *calibrate* their synthetic scanner to exactly match the NextEngine scanner, in part due to the lack of details on the internals of the NextEngine scanner.

Medeiros et al. [Medeiros et al. 2014] present a benchmark for structured light reconstruction algorithms with the goal of measuring the effects of illumination artifacts, including projector defocus, inter-reflections and subsurface scattering. They build a synthetic simulator using photo-realistic rendering (implemented with PBRT) that takes an object and its BRDF into account. They use their simulator to compare the Gray code pattern [Lanman and Taubin 2009], unstructured light scanning [Couture et al. 2011], micro phase shifting [Gupta and Nayar 2012], and ensemble codes [Gupta et al. 2013]. They are interested in qualitatively matching some of the artifacts typical of SLS, but their validation (described in the appendix) does not attempt to achieve pixel-perfect matching. It directly compares the synthetic scanner to one built with a Canon EOS Rebel T3i 18-55mm camera, a Mitsubishi HC4000 HD projector, and they use the software from Lanman and Taubin [Lanman and Taubin 2009] for binary classification (projector illuminated versus non illuminated areas).

Eiríksson et al. [Eiríksson et al. 2016] perform a set of experiments to determine precision and accuracy of parameters for structured light scanners. They are interested in determining the best calibration parameters and encoding strategies. They build a scanner, which they use for experiments of calibration parameters and encoding strategies. In their work, they use scenes with two different types of canonical objects. They also benchmark their scanner with a high-end commercial scanner (GOM ATOS III Triple Scan). Holroyd [Holroyd et al. 2010] propose a scanner to acquire geometry and surface reflectance at very high resolution. Zhang et al. [Zhang and Nayar 2006] analyze the effects of projector defocus in light scanning scenes and propose a method to increase the effective depth of field of scanning projectors.

In all these works, there is no attempt to create a digital twin in the form of a accurately matching simulator.

Light Coding Strategies. Since the development of structured light scanning, many light coding strategies have been proposed. Slightly outdated survey papers from Salvi et al. [Salvi et al. 2010, 2004] give an overview of the many different techniques. Recent light coding strategies [Couture et al. 2011; Gupta and Nakhate 2018; Moreno et al. 2015] aim for robustness towards the common sources of error in structured light scanning (interreflection, projector defocus, challenging materials). Another direction with similar goals is the automatic generation of coding patterns while optimizing the scanning accuracy. Mirdehghan et al [Mirdehghan et al. 2018] generate optimized patterns for a given scanner setup to minimize the correspondence errors. Chen et al. [Chen et al. 2020] optimize the coding-decoding procedure by modifying the projected patterns for arbitrary SLS setups in an automatic manner. The camera and projector setup is treated as a blackbox which is contrary to our approach of modeling the full setup. Instead of using high accuracy

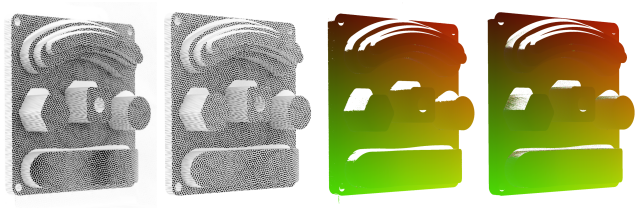


Fig. 7. From left to right: Shapes object illuminated with unstructured light patterns (40 patterns of frequency 0.1) from physical scanner, simulation, decoded lookup table from physical scan, decoded lookup table from simulated scans.

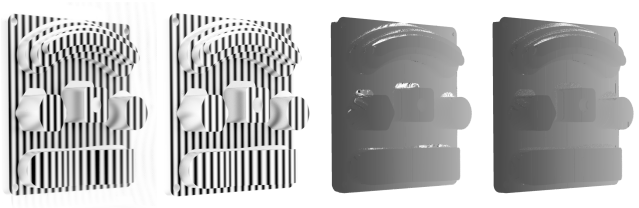


Fig. 8. From left to right: Shapes object illuminated with micro phase shifting patterns (32/8 frequency pattern) from physical scanner, simulation, decoded horizontal indices from physical scan, decoded horizontal indices from simulated scans.

objects for the purpose of objectively quantifying reconstruction errors (in SI units), their method relies on the same reconstruction technique (but with more patterns) for ground truth estimation. This does not permit the unbiased quantification of reconstruction errors. Our work is thus orthogonal to their direction and enables further research in light coding strategies.

Synthesis of Realistic Images. In terms of side-by-side comparisons between rendered and photographed images, Phong's [Phong 1975] seminal paper was first in using visual comparison of the rendered image of a sphere to a photograph to highlight the quality of his shading model. Meyer et al. [Meyer et al. 1986] performs two detailed studies: comparing radiometric measurements between physical and rendered models, and a perceptual study comparing rendered images shown on a color TV monitor to the physical model using the Cornell Box [Goral et al. 1984]. Pattanaik et al. [Pattanaik et al. 1997] calibrates a CCD camera to compare real and synthetic imagery of the Cornell Box, and attribute image differences to "mismatch between the numerical description of the scene geometry and the actual geometry". We are not aware of any existing work able to achieve a faithfulness comparable to our approach, especially on geometrically complex objects.

The problem of designing a perceptual model to compare real and synthetic images has been pioneered by Rushmeier et al. [Rushmeier et al. 1995]. In our setting, we opt for direct pixel-wise difference as our goal is to generate replicas of images to faithfully simulate a 3D reconstruction instead of producing perceptually similar images.

Hannemose et al. [Hannemose et al. 2020] propose a method for aligning photographs with rendered images, targeting settings where scanner and camera calibration are not available. Their goal is to support development of appearance models through quantitative validation. They let the user initialize the approximate orientation

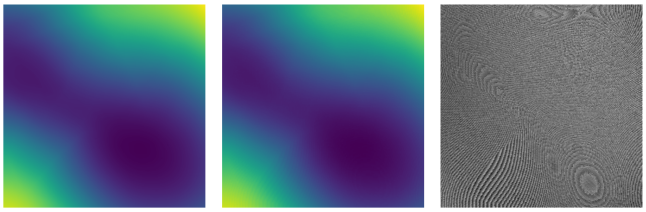


Fig. 9. Difference between the ground truth depth and the reconstructed depth for a test simulated scene with a smooth NURBS surface. From left to right: Ground truth depth map, reconstructed depth map, difference between the depth maps. The difference image shows the discretization artifacts that are produced by the Gray code patterns.

of the object relative to the planar surface, and then estimates the light source position and the camera and object poses. They validate their method using 3d printed models, showing good alignment between the acquired and rendered images. This technique is not required in our setting, where we have control over both camera and projector. We can accurately calibrate the scanning system and recover an object's position directly from the reconstructed point cloud.

3 ADDITIONAL ACCURACY EXPERIMENTS

We provide additional experiments to evaluate the scanner and reconstruction accuracy.

3.1 Correspondence Accuracy

We repeat the same experiment as proposed in the main paper with unstructured light (see Figure 7) and micro phase shifting (see Figure 8) coding patterns. Note that these patterns lead to less holes in the correspondence map, especially on inclined faces and regions with prominent subsurface scattering and interreflection. Similar to the Gray code results, the calculated correspondences from real and simulated images are almost identical, confirming that our simulation is accurate.

3.2 Reconstruction Accuracy

We want to evaluate the maximally achievable accuracy from the structured light scanning process given certain projection patterns. For this, we compare the ground truth depth map from simulation to the reconstructed depth map acquired from simulation renderings and subsequent decoding and reconstruction procedures. Figure 9 demonstrates the discretization artifacts that are introduced by using Gray code patterns which produce integer encoded pixel correspondences and therefore staircase artifacts after reconstruction. Other light scanning patterns such as unstructured light or microphase shifting produce continuous pixel correspondences and therefore no such artifacts. Besides the mitigation of artifacts, these optimized patterns have other advantages over standard Gray code patterns such as higher precision under defocusing and light interreflection effects. With the close pixel-wise matching of the simulated images to images from the scanner (as demonstrated for example in Figure 1) and the ground truth depth maps, our simulator serves as an ideal testbed for developing and evaluating optimal structured light scanning patterns.

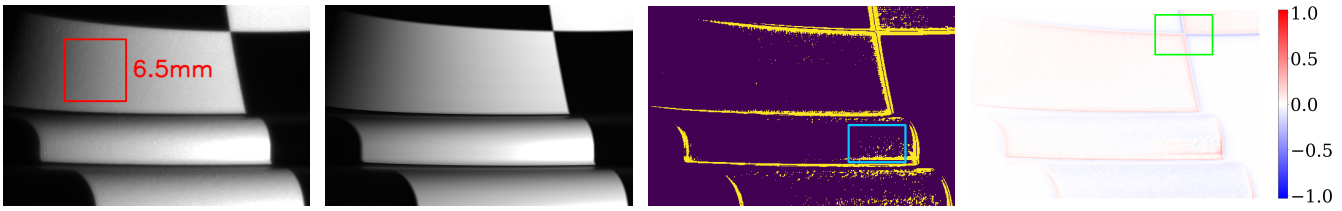


Fig. 10. From left to right: close-up view of the highlighted red region of Figure 1 with corresponding rendering/simulation in non-inverted colors. The error threshold for the mask image (5%) and the scale of difference images are the same as on Figure 1. The remaining error sources include gaps between projector pixels (≈ 2%, highlighted red), minor misalignment and difference in projector defocus around sharp transitions (up to 20%, highlighted green), and material surface microstructure in specular regions (about 5 − 10%, highlighted cyan). Relative sensor noise in the camera image is negligible (< 3% for dark regions) because of HDR image capture and, therefore, is not simulated.

3.3 Simulation Accuracy

The high accuracy of our system has been achieved by controlling the lighting setup and restricting the choice of materials. Despite its apparent simplicity, we believe this is an important setting, as it allows to establish a baseline performance for 3D reconstruction methods tailored to structured-light scanning.

Because all steps of our geometric calibration are performed with errors less than $100\mu m$ and radiometric calibration is accurate down to several percent (see supplementary material for more details), we are able to quantify different sources or errors in SLS. For instance, we were able to estimate, quantitatively, the effect of global illumination due to the secondary light scattering of the rotating stage supporting the scanned object (Figure 1). Moreover, the high resolution of our setup (one camera pixel corresponds to $\approx 65\mu m$ on objects surface) revealed that effects, such as gaps between individual micro-mirrors of the projector (one projector pixel corresponds to $\approx 200\mu m$ on objects surface), introduce noise greater than image sensor noise, and even more so for material surface microstructure in specular regions (Figure 10). This illustrates the hidden biases of many of the previous sim2real approached that do not dedicate enough attention to validation of their simulation techniques and simplifying assumptions. Our results also demonstrate the lower bound on what simulation accuracy is achievable, even in a very simple setup, without resorting to extreme measures like simulating internal structure of the light sources or accounting for thermal expansion.

A potential improvement to our simulator is inclusion of the varying blur kernels for projector defocus simulation. Projector focus and aperture calibration stage revealed that the defocus pattern for projector pixels changes across the projected image and with distance away from the focal plane. It is the largest remaining source of errors in our simulator (Figure 10). However, accurate simulation of this effect would complicate the calibration procedure significantly. Extending our work to more realistic lighting setups and to materials with non negligible subsurface scattering is an exciting avenue for future work too.

4 TECHNICAL DETAILS ON THE BENCHMARK BASELINES

4.1 Denoising

Inspired by other depth denoising [Yan et al. 2018] and image-to-image translation approaches [Isola et al. 2017], the denoising network follows the shape of a U-Net [Ronneberger et al. 2015]. As we aim to remove small-scale noise, we assume that the depth from the reconstruction process is already close to the ground truth depth. Thus, our network does not predict an absolute depth but an *offset* from the reconstruction, which spares it from having to learn the identity mapping. The encoder part consists of eight 4×4 convolution layers with stride 2 each followed by an instance normalization layer [Ulyanov et al. 2017] (except for the first one) and leaky ReLU activation. The first convolutional layer outputs 64 features and the number of output features is doubled with each consecutive layer. The decoder mirrors the encoder but uses plain ReLU activations. We follow the implementation of [Isola et al. 2017] but replace the transpose convolutions in the decoder with 3×3 resize convolutions to avoid checkerboard artifacts in the generated output [Odena et al. 2016]. After the final layer, a scaled tanh activation function produces values in the range of ±10 mm. This offset is then added to the input depth.

We do not train on the full resolution but use random crops of size 256×256 to stay within reasonable memory bounds. The training objective is the L1 loss between the predicted depth and the ground truth, calculated only over regions containing a valid reconstruction. We use the Adam optimizer [Kingma and Ba 2015] with a constant learning rate of 0.0001 and train for 20000 epochs. For inference on a full resolution depth map, we split the input into overlapping tiles that are processed individually and then merged back into the image grid.

While our simple model by no means is a flawless denoiser, it shows that our simulation of the scanning process can provide data for tasks that previously could not even be quantified accurately. In particular, the submillimeter errors and noise patterns of the scanning process seem to follow very interesting dynamics (Figure 11).

REFERENCES

Matthew Berger, Joshua A. Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T. Silva. 2013. A Benchmark for Surface Reconstruction. *ACM Trans. Graph.* 32, 2, Article 20 (April 2013), 17 pages. https://doi.org/10.1145/2451236.2451246

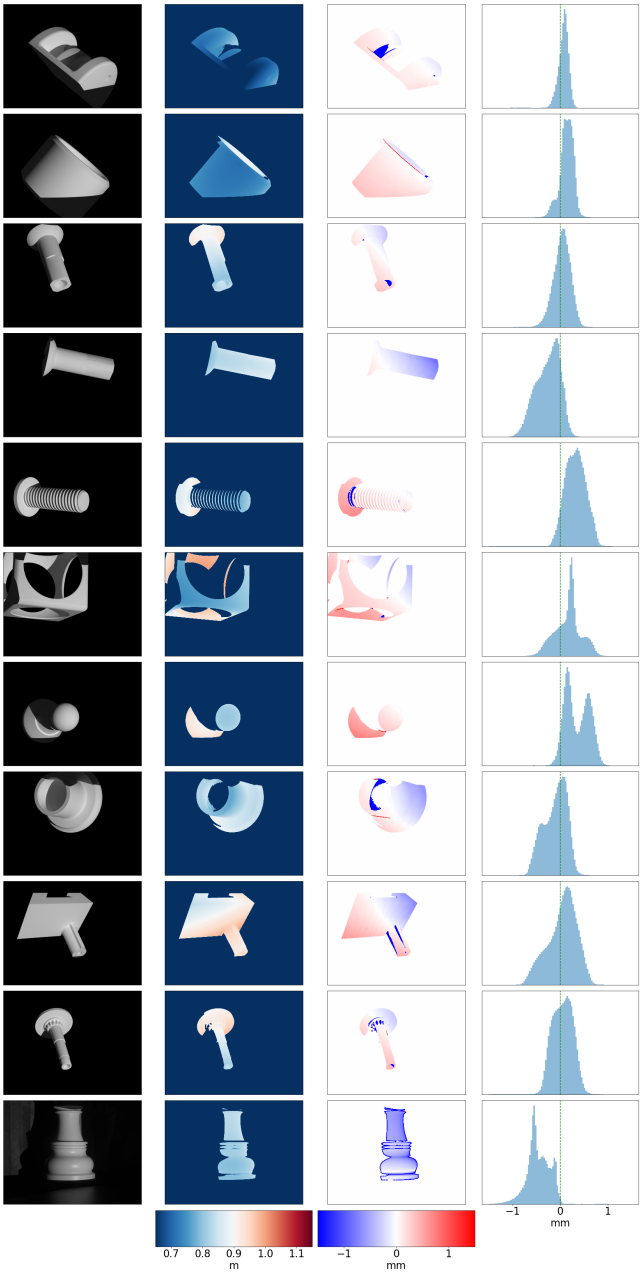


Fig. 11. Reconstruction errors for several virtual scans of LSD objects and a real scan of the rook (bottom row).

G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

Wenzheng Chen, Parsa Mirdehghan, Sanja Fidler, and Kiriakos N. Kutulakos. 2020. Auto-Tuning Structured Light by Optical Stochastic Gradient Descent. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

R. L. Cook and K. E. Torrance. 1982. A Reflectance Model for Computer Graphics. *ACM Trans. Graph.* 1, 1 (Jan. 1982), 7–24. https://doi.org/10.1145/357290.357293

V. Couture, N. Martin, and S. Roy. 2011. Unstructured light scanning to overcome interreflections. In *2011 International Conference on Computer Vision*. 1895–1902. https://doi.org/10.1109/ICCV.2011.6126458

Paul E. Debevec and Jitendra Malik. 1997. Recovering High Dynamic Range Radiance Maps from Photographs. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 369–378. https://doi.org/10.1145/258734.258884

E. R. Eiríksson, J. Wilm, D. B. Pedersen, and H. Aanæs. 2016. Precision and Accuracy Parameters in Structured Light 3-D Scanning. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL5 (April 2016), 7–15. https://doi.org/10.5194/isprs-archives-XL-5-W8-7-2016

S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* 47, 6 (2014), 2280 – 2292. https://doi.org/10.1016/j.patcog.2014.01.005

Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. 1984. Modeling the Interaction of Light between Diffuse Surfaces. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 213–222. https://doi.org/10.1145/964965.808601

Mohit Gupta, Amit Agrawal, Ashok Veeraraghavan, and Srinivasa G. Narasimhan. 2013. A Practical Approach to 3D Scanning in the Presence of Interreflections, Subsurface Scattering and Defocus. *Int. J. Comput. Vision* 102, 1–3 (March 2013), 33–55. https://doi.org/10.1007/s11263-012-0554-3

M. Gupta and N. Nakhate. 2018. A Geometric Perspective on Structured Light Coding. In *ECCV*.

Mohit Gupta and Shree K. Nayar. 2012. Micro Phase Shifting. In *CVPR*. 813–820. https://doi.org/10.1109/CVPR.2012.6247753

Morten Hannemose, Mads Emil Brix Doest, Andrea Luongo, Søren Kimmer Schou Gregersen, Jakob Wilm, and Jeppe Revall Frisvad. 2020. Alignment of rendered images with photographs for testing appearance models. *Appl. Opt.* 59, 31 (Nov 2020), 9786–9798. https://doi.org/10.1364/AO.398055

Michael Holroyd, Jason Lawrence, and Todd E. Zickler. 2010. A coaxial optical scanner for synchronous acquisition of 3D geometry and surface reflectance. *ACM Trans. Graph.* 29, 4 (2010), 99:1–99:12. https://doi.org/10.1145/1778765.1778836

Chika Inoshita, Seiichi Tagawa, Md Mannan, Yasuhiro Mukaigawa, and Yasushi Yagi. 2013. Full-dimensional sampling and analysis of BSSRDF. *IPSJ Transactions on Computer Vision and Applications* 5 (07 2013), 119–123. https://doi.org/10.2197/ipsjtcva.5.119

P. Isola, J. Zhu, T. Zhou, and A. A. Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5967–5976. https://doi.org/10.1109/CVPR.2017.632

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Douglas Lanman and Gabriel Taubin. 2009. Build your own 3D scanner: optical triangulation for beginners. In *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH ASIA 2009, Yokohama, Japan, December 16-19, 2009, Courses Proceedings*. ACM. https://doi.org/10.1145/1665817.1665819

Esdras Medeiros, Harish Doraiswamy, Matthew Berger, and Claudio T. Silva. 2014. Using physically Based Rendering to Benchmark Structured Light Scanners. *Computer Graphics Forum* 33, 7 (2014), 71–80. https://doi.org/10.1111/cgf.12475 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12475

Gary W. Meyer, Holly E. Rushmeier, Michael F. Cohen, Donald P. Greenberg, and Kenneth E. Torrance. 1986. An Experimental Evaluation of Computer Graphics Imagery. *ACM Trans. Graph.* 5, 1 (1986), 30–50. https://doi.org/10.1145/7529.7920

Parsa Mirdehghan, Wenzheng Chen, and Kiriakos N. Kutulakos. 2018. Optimal Structured Light a la Carte. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6248–6257. https://doi.org/10.1109/CVPR.2018.00654

Daniel Moreno, Kilho Son, and Gabriel Taubin. 2015. Embedded phase shifting: Robust phase shifting with embedded signals. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2301–2309. https://doi.org/10.1109/CVPR.2015.7298843

Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and Checkerboard Artifacts. *Distill* (2016). https://doi.org/10.23915/distill.00003

Sumanta N. Pattanaik, James A. Ferwerda, Kenneth E. Torrance, and Donald P. Greenberg. 1997. Validation of Global Illumination Simulations through CCD Camera Measurements. In *5th Color and Imaging Conference, CIC 1997, Scottsdale, Arizona, USA, November 17-20, 1997*. IS&T - The Society for Imaging Science and Technology, 250–253. http://www.ingentaconnect.com/content/ist/cic/1997/00001997/00000001/art00049

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Bui Tuong Phong. 1975. Illumination for Computer Generated Pictures. *Commun. ACM* 18, 6 (1975), 311–317. https://doi.org/10.1145/360825.360839

Claudio Rocchini, Paolo Cignoni, Claudio Montani, Paolo Pingi, and Roberto Scopigno. 2001. A low cost optical 3D scanner. *Comput. Graph. Forum* 20, 3 (2001).

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (Eds.). Springer International Publishing, Cham, 234–241.

Holly E. Rushmeier, Gregory J. Ward, Christine D. Piatko, Phil Sanders, and Bert W. Rust. 1995. Comparing Real and Synthetic Images: Some Ideas about Metrics. In *Rendering Techniques '95, Proceedings of the Eurographics Workshop in Dublin, Ireland, June 12-14, 1995 (Eurographics)*, Pat Hanrahan and Werner Purgathofer (Eds.). Springer, 82–91. https://doi.org/10.1007/978-3-7091-9430-0_9

Joaquim Salvi, Sergio Fernandez, Tomislav Pribanic, and Xavier Llado. 2010. A state of the art in structured light patterns for surface profilometry. *Pattern Recognition* 43, 8 (2010), 2666–2680. https://doi.org/10.1016/j.patcog.2010.03.004

J. Salvi, J. Pagès, and J. Batlle. 2004. Pattern codification strategies in structured light systems. *Pattern Recognit.* 37 (2004), 827–849.

K. Sato and S. Inokuchi. 1985. Three-dimensional surface measurement by space encoding range imaging. *Journal of Robotic Systems* 2 (Jan. 1985), 27–39.

Christiane Ulbricht, Alexander Wilkie, and Werner Purgathofer. 2006. Verification of Physically Based Rendering Algorithms. *Comput. Graph. Forum* 25, 2 (2006), 237–255. https://doi.org/10.1111/j.1467-8659.2006.00938.x

D. Ulyanov, A. Vedaldi, and V. Lempitsky. 2017. Improved Texture Networks: Maximizing Quality and Diversity in Feed-Forward Stylization and Texture Synthesis. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4105–4113. https://doi.org/10.1109/CVPR.2017.437

Michael Weinmann and Reinhard Klein. 2015. Advances in geometry and reflectance acquisition (course notes). In *SIGGRAPH Asia 2015 Courses, Kobe, Japan, November 2-6, 2015*. ACM, 1:1–1:71. https://doi.org/10.1145/2818143.2818165

Shi Yan, Chenglei Wu, Lizhen Wang, Feng Xu, Liang An, Kaiwen Guo, and Yebin Liu. 2018. DDRNet: Depth Map Denoising and Refinement for Consumer Depth Cameras Using Cascaded CNNs. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 155–171.

Li Zhang and Shree Nayar. 2006. Projection Defocus Analysis for Scene Capture and Image Display. In *ACM SIGGRAPH 2006 Papers* (Boston, Massachusetts) *(SIGGRAPH '06)*. Association for Computing Machinery, New York, NY, USA, 907–915. https://doi.org/10.1145/1179352.1141974

Z. Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334. https://doi.org/10.1109/34.888718

A HARDWARE GUIDE

In this Appendix, we describe in detail the specific hardware choices we made (A.1) and the software component necessary to operate the setup (A.2). The source code and design files are hosted at https://github.com/geometryprocessing/scanner-sim.

A.1 Physical Setup

A detailed bill of materials, the CAD models, the schematics of the calibration objects, and the schematics for the portable setup rig can be found in the scanner/hardware folder of the software repository. Here, we highlight the most important parameters of our hardware components.

Calibration Objects. We purchased a 400x300 mm CharuCo calibration board with 18x25 checkers $S_C = 15\ mm$ in size (fine pattern) from Calib.io. For the linear stage, we used a 400 mm long linear stage by Beauty Star from Amazon with STEPPERONLINE DM332T motor driver also from Amazon. We implement a motor controller using a Texas Instruments Tiva C Microcontroller. A 300x300 mm Spectralon®is purchased from Edmund Optics.

For the scanning objects, we develop a protocol to obtain high accuracy via CNC milling and a uniform diffuse coating. The objects (Figure 12) are fabricated with a geometric tolerance of $50\mu m$. We would like to emphasize that the effort to fabricate objects with such a low geometric tolerance is necessary to obtain pixel-wise image reproduction, as the objects will be used in a differentiable rendering pipeline to fine tune the simulator parameters (Section 1.5). To the best of our knowledge, the idea of using complex fabricated objects to calibrate the parameters of a 3D scanner has not been explored before.

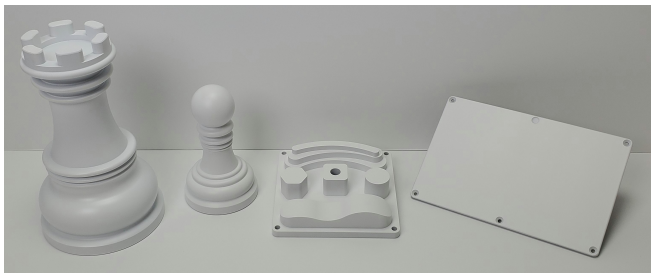


Fig. 12. The highly accurate objects that we use for calibration and validation of the scanner. From left to right: *Rook Pawn*, *Shapes*, and *Flat*.

Camera. We choose the following camera configuration:

- *Sensor*: Sony IMX342 CMOS Monochrome (C_C) with resolution (W_C x H_C) of 6464 x 4852 pixels and pixel size $3.45\mu m$ (Atlas 31.4 MP Model by Lucid Vision Labs).
- *Lens*: Edmund Optics TFL-Mount APS-C 50 mm lens with supported aperture in f/1.8–f/16 range (Edmund Optics).

The resulting resolution at $F_C = 810$ mm focus distance (this parameter is fine-tuned in Section 1.3) for the camera in our setup is $\approx$ 65 µm/pixel with contrast of 50% and aperture $f/12.5$ (see Figure 13, left). The f-stop of $f/12.5$, corresponding to an aperture diameter $f_C = 4$ mm, was chosen to maximize the depth of view for the camera without sacrificing the camera resolution (due to the lens diffraction at smaller f-numbers).

Lights. We settle on Texas Instruments DLP4710EVM-LC Light Control Evaluation Module as our light source. It is capable of projecting a 320 x 180 mm Full HD image at the focus distance $F_P = 490$ mm, resulting in a spatial resolution of roughly 165 µm/pixel with contrast greater than 150%. We did not use a "4k" projector as, during the development of this project, there were no commercially available ones with a DMD device at 4k resolution, despite many of them being advertised as 4k projectors.

Note that a DLP projector avoids color bleeding and the monochrome camera also reduces this potential problem. Color scans can be achieved by measuring R/G/B wavelengths independently. We did not observe any significant lens aberrations (in particular in the central region of the projected image used for scanning). We also use a x4 neutral filter for the projector to reduce its brightness as we use it at a close range and the camera has a lower bound on exposure time. Ambient lighting is modeled by surrounding the setup with semitransparent diffuse fabric from a standard soft-box light system.

Scanner Geometry. We build a custom frame to keep the camera and the projector locked in the desired position for consistent calibration parameters across multiple scans (see Figure 2). We use an off-the-shelf HP Turntable Pro for scanning the objects from multiple view angles. The entire setup is enclosed by a black or semitransparent and diffuse white covering to emulate black box or diffuse ambient lighting.

Materials. The objects are fabricated using 3D Hubs with their surface finish option of matte white powder coating for part of the

objects. The remaining objects were coated manually using Rust-Oleum Ultra Matte White paint (Amazon). In addition, we coat the pawn object with Winsor & Newton gloss varnish (Amazon) to evaluate the performance of the system for glossy materials.

A.2 Software Package

Software package for operation of the physical scanner setup can be found in scanner/capture folder of the software repository.

Configuration. The scanner software is dependent on camera and projector drivers. ArenaSDK with Python API needs to be installed in the system to operate the camera sensor and DLPDLC-GUI tool for projector configuration. Additionally, network drivers of the Thunderbolt to 5GBASE-T Ethernet adapter (e.g. Amazon) used to connect the camera sensor to the PC are required if the system does not natively support high-speed network connectivity. It is important to enable Jumbo packets support and increase transmission buffers size in the network driver settings to ensure stable image capture at full speed. Also, one should maintain the same color temperature settings in DLPDLC-GUI tool when adjusting the projector brightness.

Turntable and Linear Stage both have standard COM interface (see stage.py) and, thus, do not require any additional drivers besides Serial Python package for operation. Additionally, OpenGL and OpenEXR packages are required for pattern display(.py) and HDR(.py) image processing. The software was tested on the Windows operating system with Python versions 3.7 and 3.9.

Data Capture. The camera component (capture.py) is designed to capture single-exposure raw LDR images as well as multi-exposure HDR images. The latter is dependent on dark frames that need to be captured in advance (with the lens cap installed) using the capture_dark_frames function in capture.py and processed using the generate_dark_frames function in hdr.py. The final dark frames for typical exposure values and our sensor of choice can be found in the calibration/camera_instrinsics/dark_frames data folder.

All the scanner components (camera, projector, stage and user input) are being serviced independently in their own threads so that the main thread can be used to process user commands, including, among others, exposure or data output folder selection, loading/displaying of a pattern and ldr/hdr image capture. To accomplish a complete object scan with a set of decoding patterns (grayscale, microphase shifting etc) for different object positions (stage rotation) we introduce support for scripting of data acquisition, that is, a group of individual commands stored in *.script file can be loaded and execute one-by-one with a single "script" command. Examples of such scripts can be found in the "scripts" subfolder. Scripts executed form within another script should be loaded using "subscript" command to prevent the overload of "prefix" command that defines the output save folder. The "suffix" command can be used instead to define subfolders for different collections of patterns.

We use the gen_* functions from scan.py to generate commonly used script such as a scan for camera calibration (gen_calibration_script) or a script for complete scan of an object using multiple patterns (gen_multiscan_script). Other commands defined in scan.py include the "move" command for controlling the stage and various pattern display commands (e.g. "checker", "color", "load" etc). We refer the user to the source code for more details on the input parameters of these commands.

Table 1. Parameters of our hardware setup, notation used, and references to the sections where they are defined and calibrated. Positions T_i and orientations R_i are defined in the camera's frame of reference. N/A stands for values fixed by the hardware choice that do not require calibration or measurement.

Parameter	Not.	Def.	Cal.	Units
Calibration Objects				
CharuCo checker size	S_C	1.1	N/A	mm
Pawn ball height	h	1.1	N/A	mm
Coating thickness	d	1.1	B.1	μm
Camera				
Image width	W_C	1.2	N/A	pixels
Image height	H_C	1.2	N/A	pixels
Color range	C_C	1.2	N/A	components
Calibration matrix	M_C	1.2	B.2	3x3 matrix
Distortion coefficients	d_C	1.2	B.2	5-vector
Optimal calibration mat.	M_{CO}	1.2	B.2	3x3 matrix
Aperture (diameter)	f_C	1.2	B.2	mm
Focus distance	F_C	1.2	B.3	mm
Vignetting	V_C	1.2	B.5	W_C x H_C image
Projector				
Image width	W_P	1.3	N/A	pixels
Image height	H_P	1.3	N/A	pixels
Color range	C_P	1.3	N/A	components
Projected checker size	S_P	1.3	N/A	pixels
Calibration matrix	M_P	1.3	B.3	3x3 matrix
Distortion coefficients	d_P	1.3	B.3	5-vector
Optimal calibration mat.	M_{PO}	1.3	B.3	3x3 matrix
Aperture (diameter)	f_P	1.3	B.3	mm
Focus distance	F_P	1.3	B.3	mm
Vignetting	V_P	1.3	B.5	W_P x H_P image
Response function	r_P	1.3	B.5	scalar function
White balance	B_P	1.3	B.5	3-vector
Position	T_P	1.4	B.4	3-vector, mm
Orientation	R_P	1.4	B.4	3x3 matrix
Stage				
Position	T_S	1.4	B.4	3-vector, mm
Orientation	R_S	1.4	B.4	3x3 matrix
Objects				
Position	T_O	1.4	B.5	3-vector, mm
Orientation	R_O	1.4	B.5	3x3 matrix
Material parametrization	p	1.5	B.5	n-vector

B PARAMETERIZATION AND CALIBRATION

Here, we describe the parameterization and calibration of the hardware components in details. We first define each parameter for every component of the system. Then, we describe the setup and calibration procedure used to measure them. We also provide pointers to

the corresponding code in the supplementary material for each of these calibration techniques.

A comprehensive list of all the parameters used to define a virtual copy of the physical scanner is available in Table 1. The code for calibration is located in the *scanner/calibration* directory of the software repository. All the data can be found at https://archive.nyu.edu/handle/2451/62251.

B.1 Calibration Objects

Parametrization. The parameters of calibration objects used during the calibration of the other components of the setup as well as during simulation are: Camera calibration checker size S_C equal to 15 mm (CharuCo board checker size), the height h of the *Pawn*'s ball (center point) above its base, and coating thickness d for the spray painted objects.

Calibration. We measure the thickness of the spray painted coating using a micrometer for the *Flat* object before and after applying the coating, and it amounted to $d = 75\mu m$. The powder coated objects (i.e. Rook) have guaranteed tolerances of $\approx 50\mu m$ by the manufacturer which is ideal for validation purposes. The known geometry of the (spray painted) *Pawn* object (e.g. the height h of the top ball above the base) is also used to complete the calibration of the rotating stage (see Section B.4).

Code Reference. CAD models of calibration objects can be found in scanner/hardware folder of the software repository.

B.2 Camera

Parametrization. The camera is parametrized by: image resolution $W_C \times H_C$, number of color components C_C, aperture f_C, intrinsic parameters (calibration matrix M_C and distortion coefficients d_C), focus distance F_C and vignetting V_C. Some parameters follow from the choice of hardware (e.g. camera image resolution, number of color components, and aperture size). We will now discuss calibration of the intrinsic parameters, and postpone the remaining ones to later stages, as they require additional hardware.

We also use an additional camera calibration matrix M_{CO} obtained using the `OpenCV::getOptimalNewCameraMatrix` function in the OpenCV [Bradski 2000] library. M_{CO} is used in the simulator, while the combination of M_C and d_C are used to remove the distortion of the acquired images and for calibration.

Calibration. The geometric calibration of the intrinsic camera parameters (camera matrix M_C and distortion coefficients d_C) is done using the standard OpenCV [Bradski 2000] implementation of Zhang et al. [2000]. We use 65 images for the initial pass to find the initial reprojection errors for each image and then refine the calibration by running the algorithm again only for the images (31 total) with initial average reprojection error lower than 0.9 pixels. The resulting refined camera reprojection errors are shown in Figure 13 (right). The images were taken by fixing the camera one meter above the table with the lights on and moving the CharuCo calibration board slowly (to reduce the motion blur) while keeping close to the focus distance of the camera.

Code Reference. The intrinsic camera calibration is performed with the calibrate_intrinsic function from the camera.py script. It takes a data path with CharuCo images (camera_intrinsics/ data/charuco) and re-projection error threshold as an input. The CharuCo images have to be first preprocessed with the detect.py script to detect the markers and locate their corresponding corners. The intermediate results will be stored in the "detected" subfolder.

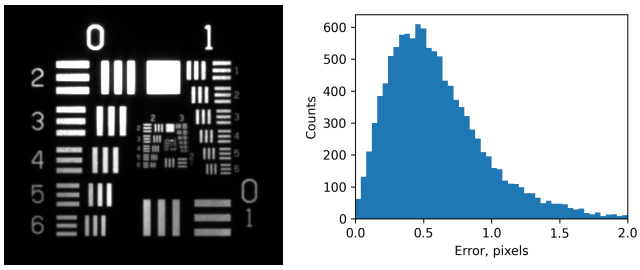
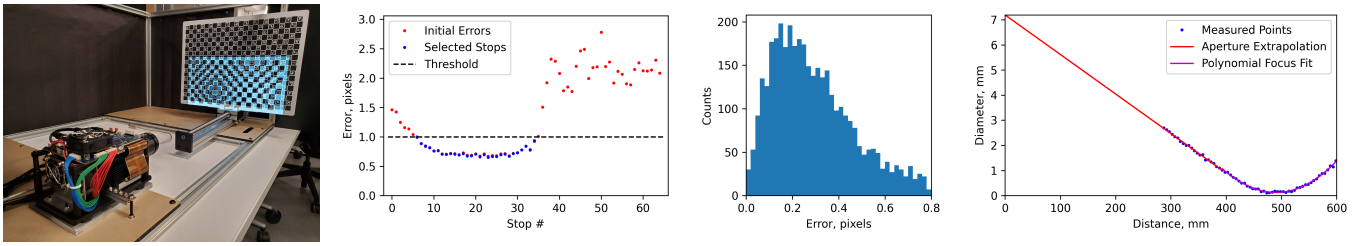


Fig. 13. Camera resolution at $F_C = 81$ cm focus distance and aperture $f/12.5$ (left) is ≈ 65 µm/pixel (lines 2-3) with contrast of 50%. The measurement was made using a USAF 1951 resolution test pattern. The reprojection errors during calibration of the intrinsic camera parameters (M_C and d_C) using the CharuCo board are shown on the right. Average reprojection error is 0.62 pixels.

B.3 Projector

Parametrization. The projector is parametrized by the same parameters as the camera (with the P subscript): projected pattern resolution resolution $W_P \times H_P$, number of color components C_P, aperture f_P, intrinsic parameters (calibration matrix M_P and distortion coefficients d_P), focus distance F_P, and vignetting V_P. Additionally, the projector has a non-linear response function r_P. In later stages, we project a regular checkerboard using the projector: the size of the checker size is S_P is fixed at 100 pixels.

Calibration. Calibration of the intrinsic parameters (M_P and d_P) for the projector is based on the camera calibration. We use a special setup (Figure 14, a) with the camera positioned behind the projector so that their focus plane matches. We then use the linear stage to move the calibration board with 5 mm steps and capture three images for each stop with the projector projecting (1) a black image, (2) a white image, and (3) a checker pattern image, all three with ambient (room) lights turned on. Then the black image is used to reconstruct the position of the CharuCo board and the rest are combined as $clean = (checker - black)/(white - black)$ to obtain a *clean* image containing the checker pattern projected onto the CharuCo board plane (with the CharuCo board texture removed basically). We reconstruct the position of the CharuCo board by detecting visible markers and using `OpenCV:solvePnP` function to ray-trace the detected corners of the projected checker pattern and obtain the corresponding 3D positions. With this "ground truth" positions of the checker corners, we execute the calibration algorithm by Zhang et al. [2000] as if the projector was a camera. This yields the reprojection error estimates for each stop and one can clearly see on Figure 14 (b) that the errors are the lowest if the CharuCo board is at (or close to) the focus distance of the projector. We select the positions with initial reprojection error lower than one pixel (29 stops out of 65) and rerun the algorithm again but combine this time all the points into a single large "3D calibration object" to obtain the



(a) (b) (c) (d)

Fig. 14. Projector calibration setup (a) and calibration metrics (b-d). The linear stage is used to repeat the measurements with both sides of the calibration board (first with checker projected onto the CharuCo side and then with single pixel dots onto the clear side). The positions where the projected checker image is most in focus (b) are used for intrinsic calibration (M_P and d_P). The mean projector reprojection error is 0.35 pixels (c). The recovered blur diameters of a single pixel dots is at different distances are used to find the focus distance F_P and projector aperture f_P (d).

final calibration matrix M_P as well as distortion coefficients d_P for the projector. The final reprojection errors are shown on Figure 14 (c).

To measure the focus distance F_P and aperture f_P we repeat the measurements at the same positions but with the clear side of the calibration board facing forward and projecting a single-pixel dot. We perform the experiment in a dark room so that we can capture the defocus pattern for a single projector pixel at different distances from the projector (the same used in the previous experiment, which is possible due to the reproducibility of the position of the linear stage). We then fit a sigmoid function in polar coordinates to find the degree of blur for the pixel across different distances from the projector origin. The results are shown in Figure 14, (d). We can then easily compute the focus distance $F_P = 49cm$ for the projector by finding a minimum of the polynomial function fitted to the data points. Additionally, by extrapolating the diameter of the blurred pixel all the way back to the projector origin, we can estimate the projector's aperture $f_P = 7.2mm$ (all pixels will blur out and converge into the projector aperture at this point).

This experiment is also used to refine the focus distance F_C of the camera (set at $F_C = 81cm$ in Section B.2). We attach a vertical strip of dark tape on top of board and measure how the sharpness of the right edge in the images recorded by the camera at different distances. We obtain the quantitative measure of sharpness by fitting a sigmoid function to the horizontal profile of the edge. The distance at which the image of edge is the sharpest is $F_C = 81cm$ in our setup. We also place an additional light on the side of the right edge used for this measurement to eliminate any shadows due to the thickness of the tape which could be misinterpreted as defocus of the camera image.

The optimal projector calibration matrix M_{PO} is obtained using `OpenCV::getOptimalNewCameraMatrix` function in OpenCV and is used this time to predistort the projected patterns so that they can be used together with M_{PO} to simulation distortion effects during rendering.

Code Reference. The projector calibration requires the same preprocessing of the CharuCo images with the addition of the detection of the corners for the projected checker pattern as described in the previous paragraph of this section. This preprocessing is performed

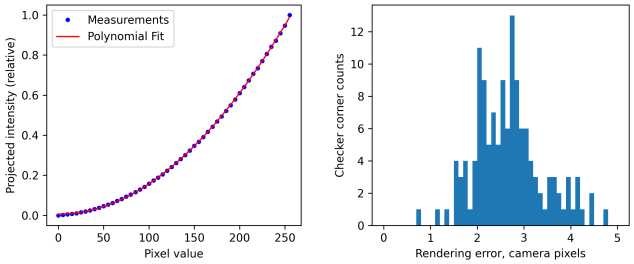


Fig. 15. The projector response function r_P for gray colors (left) and the geometric rendering accuracy (right). The checker corners detection contributes an average error of about half a camera pixel.

with the process_stage function in process.py. It takes a data directory as an input (e.g. projector_intrinsics/data/charuco_checker_5mm) and stores the detection results in corresponding subfolders. Then, the same data path and previously acquired camera calibration (stored as a json file) can be provided to the calibrate_geometry function in projector.py to estimate intrinsic ans extrinsic projector parameters.

The same setup but with clear side of the CharuCo board facing the camera and projector during the capture is being used for measuring the camera (F_C) and projector (F_P) focus distance as well as projector aperture f_P. The code for finding this values is divided into two functions calibrate_camera and calibrate_projector in focus.py script. Note that the positions of planes are being reused from the previous (with CharuCo side facing forward) scan because they are reproducible with high fidelity due to the high precision linear stage used.

B.4 Scanner Geometry

Parametrization. The geometry of the scanner is defined by the position of the projector (translation T_P and orientation R_P) and of the rotation state (translation T_S and orientation R_S). The scanned object is defined by a translation T_O and an orientation R_O.

Calibration. To calibrate the extrinsic projector parameters T_P and R_P, we repeat the same calibration procedure used for the intrinsic parameters but with the camera setup in its final position (at $\approx$ 30 deg angle). The only difference is that this time the camera cannot see the entire checker pattern, so we project only the visible

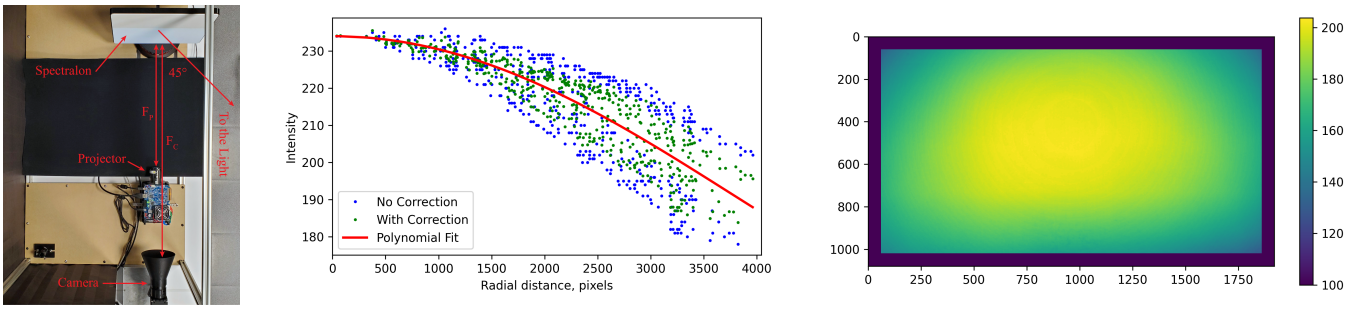


Fig. 16. Vignetting calibration setup (left) is used to measure radial profile of camera vignetting (middle) and asymmetric projector vignetting (right). The Spectralon® did not cover the entire projector image so we later extrapolate missing parts by fitting a surface polynomial for the measured region.

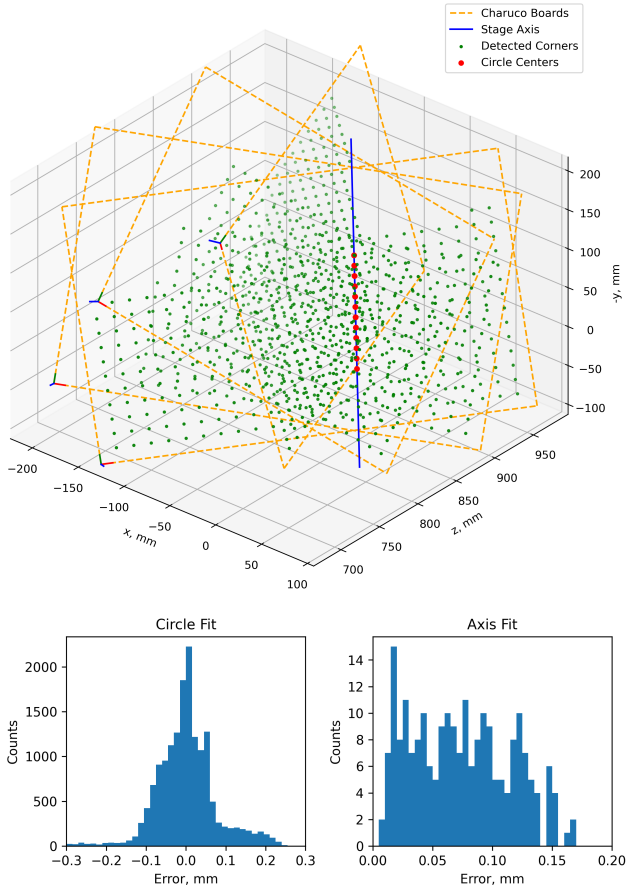


Fig. 17. The calibration of the stage axis (top) is sufficient for the registration of the scans obtained at different angles. The errors of reconstructing trajectories of the CharuCo corners and the axis are shown on the bottom. The mean stage axis fit error is $75\mu m$.

part as the OpenCV detection algorithm needs to see the entire pattern to detect it.

This reduces the total number of calibration points as well as their reconstruction accuracy. To compensate, we enforce the intrinsic calibration parameters obtained from the previous, more favourable, setup and only keep the extrinsic ones. This completes the geometric calibration of the camera-projector rig and we can proceed to locating the rotating stage.

To validate the correctness of the geometric calibration and to estimate the rendering accuracy, we conduct an experiment with the CharuCo board as a target object. We scan it as a regular object but also exploit its accurate markers to find out the ground truth position without relying on the reconstruction pipeline. This position is then used to simulate the identical setup (involving the complete set of calibration parameters) and the resulting images are compared with the ones captured by the camera when a checker pattern is being projected onto the CharuCo board as in Section B.3. The deviation of the checker corners detected for real world and simulated images can then be measured in terms of camera pixels (three camera pixels roughly correspond to one projector pixels). The resulting distribution is shown in Figure 15 (right). We observe that the average error does not exceed one projector pixel and the reprojection errors for the CharuCo markers during reconstruction of the ground truth position of the board were also on the order of the camera calibration accuracy of 1 camera pixel ($\approx 65\mu m$ on the object surface perpendicular to the camera optical axis).

Stage calibration uses a similar procedure, but with the CharuCo board placed on top of the rotating stage and making a scan with small degree increments (2 deg in our case). The reconstructed positions of the CharuCo board are then combined based on the "charuco IDs" of each detected corner into arc segments in the plane perpendicular to the stage rotation axis (see Figure 17, left). One can then recover these planes with PCA on the 3D positions of the detected corners on a per arc basis. For the arcs with sufficient number of point ($\geq$ 70 in our case, because not all points are always visible in the camera image) a 2D circle is fitted in the recovered plane. The centers of the circles are then combined for a final line fit of the rotating stage axis (see Figure 17, right). Note that it is not possible to recover a full set of T_S and R_S parameters for the stage from this experiment alone, as you recovered them up to the translation along the rotation axis. To recover the missing translation along the axis we scan the *Pawn* object, since it has a sphere on top with known offset h with respect to the base of the stage (Figure 2).

Code Reference. As described in Section 1.3, we use one setup (with camera directly behind the projector) to calibrate the intrinsic projector parameters. We then use the final scanner setup to

calibrate the extrinsic projector parameters, with previously determined intrinsic projector parameters provided as an auxiliary input to calibrate_geometry function projector.py script. The calibrate_geometry function can estimate both intrinsic and extrinsic projector parameters from the same data folder but the proposed procedure yields higher accuracy calibration. The code for validation of the camera and projector calibrations is in the accuracy.py script (test_accuracy function) and the scan of the CharuCo board is in the accuracy_test data folder.

A preprocessing code (see Section B.3) is being used during calibration of the rotating stage.The localization of the stage axis is done based on CharuCo markers only but the data acquired for different angle rotations needs to be brought into the same format as for linear stage with merge_positions function (the results are stored in the /merged subfolder) prior to calling process_stage in process.py. At this point, we can provide the merged data path and camera calibration to calibrate_axis in stage.py to recover the rotating stage axis. The minimal number of positions required for the CharuCo marker to be considered during calibration can be passed via the min_circle_points parameter.

Note that the rotating stage axis reconstruction is sufficient for registration of point clouds acquired from different scanning angles but one can recover a complete local frame of reference for the rotating stage from the reconstructed point cloud of the Pawn calibration object. Please refer to the locate_pawn function in reconstruction/locate.py script that employs knowledge of the Pawn ball height h to recover the stage base. This information is required by the locate_rook function to locate the Rook calibration object which does not have enough distinct geometric features like the Pawn to be located independently.

B.5 Materials

Parametrization. We parameterize camera (V_C) and projector (V_P) vignetting as an image of corresponding size (W_C x H_C and W_P x H_P correspondingly) denoting relative brightness drop across the camera/projector image. The projector response function r_P is a scalar function mapping pixel value to the actual light intensity of the said projected pixel. For the material, we use a realistic microfacet scattering model with 3 adjustable parameters p and while balance B_P is used for colored objects.

Vignetting Calibration. Camera vignetting V_C is calibrated by capturing a single image of a Spectralon with known illumination (Figure 16, left). However, because our light source is not at infinity, we need to correct for the brightness drop due to the increasing distance from the light source across the Spectralon. This is possible because we know the geometry of the setup and that the profile is radially symmetric. The measured profile for the camera vignetting is shown in Figure 16 (middle). There is still some discrepancy (it is not perfectly symmetric) due to the reillumination by secondary light scattering in the room. We correct for the light attenuation, and then fit a radial polynomial $I(r) = p_0 + p_1 r^2 + p_2 r^3 + p_3 r^4$ (where r is a distance, in pixels, from calibrated image center) to obtain a final, symmetric result. Note that we skip linear component to enforce zero derivative (flat surface) at the center.

For the projector vignetting V_P, we cannot assume radial symmetry because of the non-axial design of the optics (the optical axis is off the projected image center). Therefore, we fit a surface polynomial $I(x, y) = p_0 + p_1 x^2 + p_2 xy + p_3 y^2$ (where $x = c - C_c$, $y = r - C_r$, and (C_c, C_r) is the calibrated image center in $[row, column]$ coordinates) to the image obtained for the projector vignetting calibration (after removing the camera vignetting using V_C). We automatically find a corresponding cropping of the camera image by capturing an additional image with a checker pattern projected (Figure 16, right).

Light Source Calibration. We are using the same calibration setup (Figure 16) to measure the projector response function r_P (Figure 15, left) and white balance B_P. We found that there is no valid gamma value for the projector response function, so we use a second degree polynomial to parametrize it. Such parameterization is easy to invert during the decoding stage if non-binary patterns are used. For the camera, we validated that the sensor response is linear as expected for a sensor with digital shutter.

Code Reference. The camera (V_C) and projector (V_P) vignetting calibration is computed with the calibrate_vignetting functions in the scripts camera.py and projector.py. The input data is in Calibration/camera_vignetting and calibration/projector_vignetting folders. The same data is being used for the calibration of the projector white balance B_P with the calibrate_white_balance function in projector.py. Additionally, the calibration/projector_response data folder has to be supplied to calibrate_response in projector.py to find out the polynomial fitting for the projector response function r_P. The implementation of the optimization procedure (see Section 1.5) used to find out material parameters p is included in the provided code.

C RECONSTRUCTION

In this section, we first describe the point cloud reconstruction pipeline and then the calibration object localisation technique which is being used during scanner geometry calibration (B.4). The source code for both can be found in scanner/reconstruction folder of the software repository.

C.1 Point Cloud

The point cloud reconstruction pipeline consists of three stages: (1) decoding of the projector rays corresponding to different camera pixels, (2) triangulation of the corresponding camera and projector rays to obtain a point cloud of one side of the scanned object, and (3) registration of the partial scans of the object from different angles into a single frame of reference using stage calibration information.

1. Decoding. We use a gray code patterns by default which requires independent decoding of horizontal and vertical projector ray indices. We also project the inverse gray code patterns for greater robustness of the decoding algorithm in presence of specular materials. The corresponding pairs of HDR images can be processed using the decode_single function from decode.py. It takes the data path with HDR images as an input and the results will be saved in the *decoded* subfolder, which can later be loaded with the load_decoded function. Providing camera calibration via "undistor" keyword parameter will remove lens distortions prior to decoding. The decode_many

function can be used to decode multiple scans at once if using the multiscan script during data capture.

Because our camera typically has three time greater spatial resolution on the surface of the scanned object than the projector, there will be multiple camera pixels with identical decoded projector ray indices. We provide an additional processing option (group=True) that will combine those camera pixels into a single camera ray passing through their middle point. These reduces the amount of points in the reconstructed point cloud by one order of magnitude as well as minimizing the discretization errors. The results of this processing step will be saved alongside regular decoding results in the *decoded* subfolder.

2. Triangulation. We are using a standard triangulation method (http://mesh.brown.edu/byo3d/) that finds the point minimizing the distance to both camera and projector rays for each pair of such rays (see the triangulate function in reconstruct.py). The only difference is that we project the said point onto the camera ray so that we can have regular grid of depth values from camera's point of view. This enables us to work with depth map images in certain applications (e.g. hole filling) instead of raw point clouds. The reconstruct_single function in reconstruct.py performs this step and, obviously, depends on camera and projector calibrations. The path with original HDR images is expected to be provided to this function on input as it automatically locates the "decoded" subfolder from previous step and identifies whether lens distortions have been removed prior to decoding as well as group flag. The reconstruction results (depth maps and point clouds) will be saved to the new "reconstructed" subfolder. The reconstruct_many function in reconstruct.py is equivalent to the decode_many function in decode.py for batch processing multiple scans (e.g. for different scanning angles).

3. Registration. The registration of scans acquired at different turntable positions is rather simple when knowing the stage calibration (e.g. axis of rotation). We use the merge_single_30_deg function from merge.py to combine point clouds from difference scans into a single frame of reference of the first scan (position_0). It takes the parent multi-scan directory, stage calibration and filename template (*.ply point cloud files saved to "reconstructed" subfolder for each scan by reconstruct_many function during triangulation) as input and saves merged result into "merged" subfolder of the parent directory. merge_both_30_deg function is a convenience function for merging / registering both full resolution and grouped point clouds and names them according to object_name parameter value with appropriate suffixes (_all for full resolution and _group for grouped point clouds).

C.2 Object

The calibration objects are an integral part of the system. They (e.g. a Pawn) are being used to complete the stage calibration (see Section B.4) and to evaluate the faithfulness of the light transport simulation in our virtual setup.

Localization. To localize the calibration object, we use MeshLab to select the points in the reconstructed point cloud (with grouping feature enabled during decoding) corresponding to sections of the object with known geometric dimensions (e.g the ball on top of the Pawn or cylindrical base of the Rook). We then use the locate_<object_name> function from locate.py to recover the local frame of reference for that object. The ball on top of the Pawn is enough to fully locate the object as described in Section B.4. Complete stage calibration is then used to localize the Rook based on its cylindrical base. The Shapes object can always be localized based on top surfaces of its cube, cylinder and hexagon shape features. The Plane is easy to localize using PCA.

Texture. We are using white colored patterns when performing the geometric scan of the object. In addition, we also project uniform red, green, and blue patterns to capture the color information and the texture of the scanned object (useful for applications such as hole filling). These HDR images, corresponding to independent RGB colors, can be combined into a single colored HDR image using the color.py script which depends on projector calibration only (e.g. projector white balance). undistor.py can further be used to remove the camera lens distortion for these images (with obvious dependency on camera calibration) so that they can be directly compared to the simulated/rendered ones (extra image translation might be needed to match the optical axes of both images). By using the projector calibration instead, one can pre-distort the projector patterns for simulation of the projector lens distortion with the pinhole model but we have conveniently incorporated this step already into our simulator.

D DATASET GENERATION

Our scanning simulation software matches the calibrated setup from the physical scanner and can be used to generate large amounts of realistic data. The first stage of the simulation is the generation of images by physically based rendering. This process is the most time-consuming of the processing pipeline as there are multiple images that need to be rendered for a full scan. For example, there are 46 images that need to be rendered for each scan when using Gray code patterns. However, our implementation supports parallel processing and is therefore ideally suited for data generation on multi-core clusters. Multiple parameters and inputs can be adjusted and chosen at this step, ranging from the input geometry, the amount of samples for the rendering to the camera image resolution. Given all the settings, the renderer produces HDR (openexr) and LDR (png) images together with the ground truth depth values (npy).

After the image generation process, we use the same processing steps as described in Appendix C with minor modifications like different thresholds, cropping, etc. Next to the source version of our framework, we supply a Docker image with the preinstalled framework. This image runs a Jupyter notebook which contains a full tutorial on how to use our data generation framework. The tutorial demonstrates all the different options of how to generate data with our simulator. It is divided into 2 parts: the first part covers the initialization and rendering stage, and the second part covers the decoding, triangulation and registration stages. The tutorial can be followed and adapted step by step to generate data for most use-cases. More advanced adaptations are possible but require to modify or extend our source code.