Simulation of Structured Light 3D Scanning using Blender

A. Puljčan[1], D. Zoraja[1], T. Petković[1],
[1] University of Zagreb, Faculty of Electric Engineering and Computing, Zagreb, Croatia
ana.puljcan@fer.hr, domagoj.zoraja@fer.hr, tomislav.petkovic.jr@fer.hr

***Abstract*—Many different optical setups and patterns are proposed for structured light 3D surface scanning. It is difficult and time consuming to experimentally assess all imaging options and to select the best one for a particular application. An efficient way of assessing imaging options is simulation of 3D scanning on a computer. We describe an efficient and low cost approach to simulating structured light scanning using Blender. Blender is an open source 3D computer graphics program primarily intended for 3D drawing and animation also supporting Phyton scripting to extend its capabilities. We describe how to simulate a simple scanning setup comprised of one camera, one projector, and one object. We also provide detailed description how to set both intrinsic and extrinsic parameters of projector and camera and discuss how to automate the simulation. To assess the correctness of the proposed simulation we simulate a real-world 3D surface scanner and we compare simulation results to the real-world results. We also perform experiments of scanning calibration objects. The proposed simulation method is correct and can be use to assess imaging geometry, to select structured light patterns, or to generate data for novel pattern decoders which use deep learning.**

***Keywords*—structured light (SL), surface scanning, simulation of surface scanning, Blender**

I. Introduction

In structured light (SL) scanning a projector is used to project images or patterns onto an object of interest and a camera then captures the projected patterns [1]. Captured patterns are then processed to recover the information which is necessary to reconstruct the surface of the observed object. This simple procedure offers both robust and precise 3D measurement and has found its use in many applications including reverse engineering, quality control, human body measurement, archaeology, etc. Three important problems to be considered when designing a SL scanner are: (a) the choice of patterns [2], (b) the spatial arrangement of projectors and cameras (the optical setup), and (c) scanning conditions including e.g. ambient illumination. Considering this prototyping and analyzing a particular SL scanner using real hardware is both difficult and costly which makes a computer simulation of a SL system an attractive alternative.

Structured light scanning is generally consider a computer vision (CV) problem [3], [4], while simulation of imaging the real world is mostly a computer graphics (CG) problem [5]. Therefore, simulating a SL system is best done using existing solutions from the field of computer graphics, and that is evident from the literature. In general, simulations of SL systems described in the literature either use (1) existing and well established software and solutions for computer graphics [6–15], [15] or use (2) custom solutions [16–19].

Works using existing computer graphics solutions utilize various software: RenderMan [20] is used in [6]; 3ds Max [21] is used in [7], [8]; and Blender [22] in used in [9–15]. Curiously, some of published works [6], [7], [14], [15] omit many or almost all the details about how a SL scanner is actually simulated stating only that a particular software is used for simulation. When only some details are provided then it is often briefly mentioned that a projector is simulated using a spotlight [8], [9], however particularities about how the spotlight illumination is modified to simulate the projected patters is missing. The best description about how to simulate SL scanner is available for Blender in [10–13] where the projector is simulated as a textured spotlight and where details about how the spotlight illumination is modulated/textured are provided. Next, some control parameters which describe camera and projector are not expressed identically in CV and CG. Extrinsic parameters which define the position and the orientation in world coordinates are almost the same (if one is careful about left and right coordinate systems), however intrinsic parameters which define internal properties of camera and projector are expressed differently, and that is most evident for the projector. In CV the projector is best modelled as an inverse camera [23], [24] which is defined by a 3×3 camera matrix, while in CG the projector is simulated using a textured spotlight which is defined using a completely different parameters which define texture mapping and/or light modulation and the cone of the spotlight [12], [13]. Next, considering possible choice of rendering engines most works again do not discuss the engine at all, while some specifically mention what engine is used. For Blender the standard rendering engine which is used is Cycles [25], which is a physically based renderer [26] included with Blender. Using a physically based renderer is a must and used renderer should support as many features as possible [27]; for SL scanning the most important features are ones concerning light transport, specifically reflections and refractions, and global illumination. Finally, the problem of how to verify simulated SL scanner and compare it to a real world scanner is an open problem.

When custom solutions are used then technical details about simulation are almost always omitted. MATLAB seems to be commonly used, especially its function `peaks` which is a defacto standard surface for evaluation [16], [17]. Custom solutions almost always focus on a specific aspect of SL imaging

and are lacking compared to physically based rendering.

In this work we give a detailed description how Blender [22] can be used to simulated SL 3D scanning and we provide a detailed description how to convert standard intrinsic and extrinsic camera and projector parameters which are used in CV to ones used in Blender. Regarding verification we propose as set of values which are invariant to the actual position of camera and projector in space and which can therefore be used to verify that a simulated SL scanner has the same optical setup as the real SL scanner.

The rest of this paper is structured as follows: In Section II we present the main contribution. First we describe how camera and projector are modelled in CV and in CG and how to transform the parameters. Next, we describe how to geometrically calibrate the virtual SL scanner in Section II-D and how to perform virtual scanning in Section II-C. Results and discussion are presented in Section III-A and we conclude in Section IV.

II. Methodology

In this section we first describe how to simulate camera and projector in Blender and then we discuss how to calibrate the virtual scanner. Finally, we propose geometrical invariants to verify the simulation correctly reflects the optical configuration of a real world SL scanner.

Note that in CV a left-hand coordinate system is sometimes used. We present all material using the standard right-handed coordinate system and we assume the user will verify what coordinate system a particular software uses and will transform equations accordingly.

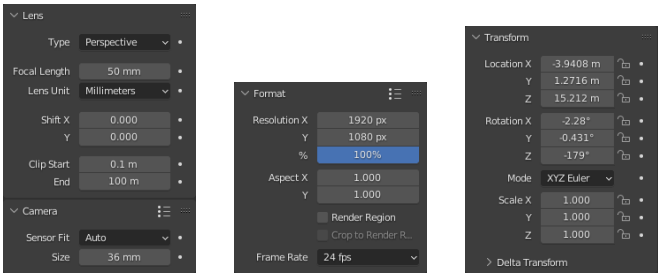
A. Camera

A camera in CV is most often described using an ideal pinhole model [4] with the lens distortion additionally modelled as a nonliner mapping of ideal image coordinates. Parameters describing such a pinhole model with distortion are usually divided into intrinsic and extrinsic parameters. Intrinsic parameters are focal length(s) f_x and f_y, skew s, the position of the principal point c, and lens distortion parameters which depend on the distortion type (radial, tangential, etc.). Focal length, skew, and principal point define the camera's calibration matrix

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

which describes an ideal pinhole camera placed at the origin. Extrinsic parameters then define the position and orientation of the camera w.r.t.the world coordinate system and are usually expressed using a rotation matrix $\mathbf{R}_c$ and a translation vector $\mathbf{t}_c$. The ideal pinhole camera is then described by a projective transformation

$$\mathbf{x}'_{2\text{D}} = \begin{bmatrix} x'_{2\text{D}} \\ y'_{2\text{D}} \\ z'_{2\text{D}} \end{bmatrix} = \underbrace{\mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}}_{\mathbf{P}} \mathbf{X}_{3\text{D}} = \mathbf{P} \begin{bmatrix} x_{3\text{D}} \\ y_{3\text{D}} \\ z_{3\text{D}} \\ 1 \end{bmatrix}, \tag{2}$$



(a) (b) (c)

Fig. 1. Camera's intrinsic (a-b) and extrinsic (c) parameters.

where $\mathbf{X}_{3\text{D}}$ is a 3D point in world frame and where $\mathbf{x}'_{2\text{D}}$ is a projective point in camera frame from which the undistorted image coordinates are computed using

$$x_{2\text{D}} = x'_{2\text{D}}/z'_{2\text{D}} \quad \text{and} \quad y_{2\text{D}} = y'_{2\text{D}}/z'_{2\text{D}}. \tag{3}$$

Lens distortion is then a non-linear mapping of ideal coordinates $(x_{2\text{D}}, y_{2\text{D}})$. Note that the standard description in CV often omits the sensor resolution which is given as width w and height h in pixels.

A camera in CG [5] is described similarly using an ideal pinhole model, however, the model is almost always additionally idealized in the sense that there is only one focal length so $f = f_x = f_y$, there is no skew so $s = 0$, and the principal point in the image plane is usually fixed so principal point is in the center of the image. Therefore, a standard CG camera has less parameters than the CV one. One important difference are the units in which the parameters are expressed, e.g. considering Eq. (2) in CV 3D coordinates are measured in physical units (meters) while 2D coordinates are always measured in pixels, while in CG all values are usually expressed in physical units. This means that f_x, f_y, c_x and c_y in $\mathbf{K}_c$ are in pixels in CV and in meters in Blender. Additionally, due to division of Eq. (3) there is an additional degree of freedom when converting the parameters, i.e.if sensor's resolution $w \times h$ in pixels and physical size are fixed then the value of focal length in Eq. (7) is unique, otherwise it depends on the sensor size. Let f_x (w.l.o.g. $f_y = f_x$) be the focal length, and let w and h be the image resolution, all expressed in pixels as is usual in CV, and let k be an arbitrary positive number. Then the focal length f_B and physical sensor size $h_B \times w_B$ in Blender expressed in meters can be set as

$$f_B = k \cdot f_x, \quad w_B = k \cdot w, \quad \text{and} \quad h_B = k \cdot h. \tag{4}$$

In Eq. (4) the parameter k is in [m/px] and can be interpreted as the physical distance between pixels on the sensor for square pixels, however, note that if k is not known then it may be arbitrarily set without affecting the simulation results [12]. Colloquially, FOV (field-of-view) remains unchanged regardless of the actual value of k.

Parameters c_x and c_y in CV are converted into the shift of the principal point σ_x and σ_y in CG using the same k of Eq. (4) as

$$\sigma_x = k \cdot (c_x - w/2) \quad \text{and} \quad \sigma_y = k \cdot (c_y - h/2). \tag{5}$$

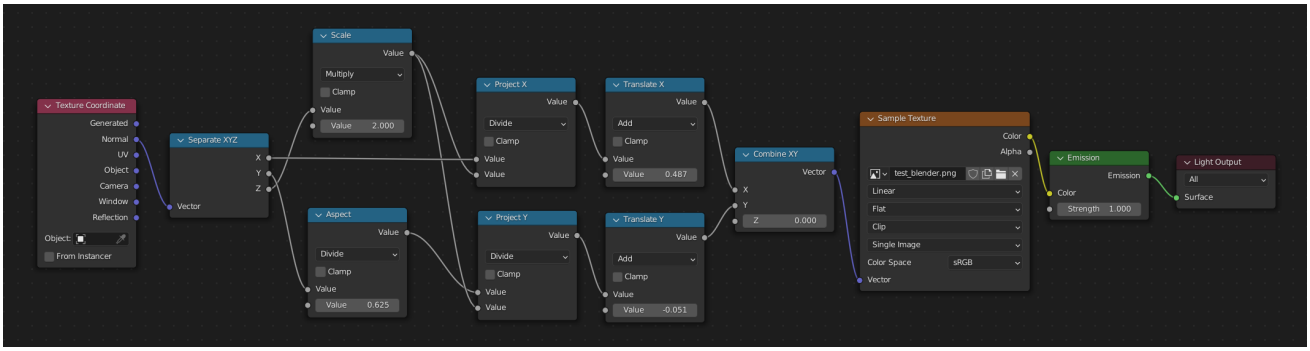


Fig. 2. Node tree for spotlight in Blender required to simulate a projector (compare to Fig. 4.12 in [11], to Fig. 3 in [10], and to Fig. 12 in [12]). Blocks in blue define arithmetic operations which compute coordinates for texture mapping.

Once intrinsic parameters are converted they can be directly entered in Blender as shown in Fig. 1(a).

Extrinsic parameters from CV correspond almost directly to camera position and orientation in CG, however in CV the focus is on coordinate transforms so $\mathbf{R}$ and $\mathbf{t}$ define the transformation from world frame to camera frame, i.e.

$$\mathbf{X}_{3\mathrm{D},c} = \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \mathbf{X}_{3\mathrm{D},w}. \tag{6}$$

In CG we are interested in position (location) $\mathbf{c}$ which is

$$\mathbf{c} = -\mathbf{R}\mathbf{t}. \tag{7}$$

Therefore, a simulated camera is positioned at location given by Eq. (7) and its orientation is set to one described by the rotation matrix $\mathbf{R}$ which is directly decomposed to Euler angles or quaternions, depending on the CG software used. Again, once extrinsic parameters are converted they can be directly entered in Blender as shown in Fig. 1(b).

B. Projector

A projector in CV is most often modelled as an inverse camera [23]. It is described using the same parameters as camera and is modelled by Eqs. (2) and (3). To distinguish between camera's and projector's parameters we will use subscript c for camera and subscript p for projector henceforth.

A projector is simulated by using a spotlight, which in CG is defined by the shape of its light cone and by illumination parameters, which are intrinsic parameters, and by its position and orientation, which are extrinsic parameters. Extrinsic parameters are easily converted in the same way as was done for camera, i.e. use Eq. (7) and decompose the rotation matrix into roll, pitch and yaw angles. Unfortunately, intrinsic parameters cannot be straightforwardly converted, which is not stresses in previous works such as [10], [15].

The angle of spotlight's illumination cone α corresponds to the largest FOV of the projector. Let projector's calibration matrix be $\mathbf{K}_p$ and let its resolution be $w_p \times h_p$. Projector's principal point defines where spotlight's optical axis intersects the image plane; its coordinates $(c_{x,p}, c_{y,p})$ in pixels are stored in the last column of $\mathbf{K}_p$. This geometry is shown in Fig. 3 where the circle of the illumination cone contains projector's

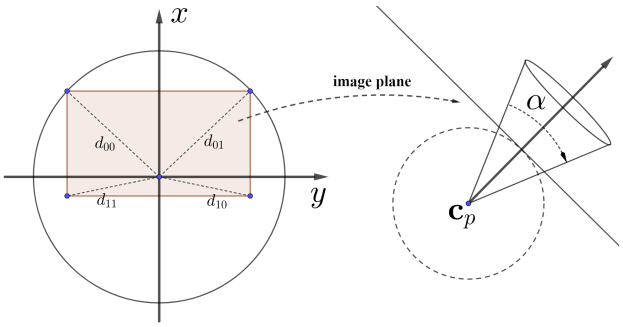


Fig. 3. Projector image formation

image. Let $\mathbf{x}_i = \{0, w_p\}$ and $\mathbf{y}_j = \{0, h_p\}$; then distances d_{ij} from the principal point to four image corners are

$$d_{ij} = \sqrt{(c_{x,p} - \mathbf{x}_i)^2 + (c_{y,p} - \mathbf{y}_j)^2}. \tag{8}$$

Denote with $D = \max_{i,j} d_{ij}$ the maximal distance. The angle of the illumination cone is then

$$\alpha = 2 \arctan\left(D / f_{x,p}\right), \tag{9}$$

where $f_{x,p}$ is projector's focal length.

Next, a texture must be mapped on the illumination cone to modulate the light. Coordinates for texture mapping are generated from spotlight's normals using Blender's node tree shown in Fig. 2. Note that texture coordinates are always normalized, i.e. a texture image is sampled using coordinates from a unit square with its lower left corner at the origin, hence normals within the image rectangle shown in Fig. 3 (left) have to be mapped to that unit square. The decomposition of the coordinate transformations we propose is comprised of arithmetic operations as shown in Fig. 2 which has the property that values of all nodes can be computed independently. The block *Aspect* controls the aspect ratio and its value is h_p/w_p. The block *Scale* defines the size in the image plane and its value is

$$4D \tan(\alpha/2)/w_p. \tag{10}$$

Blocks *Translate X* and *Translate Y* define the position and are set to

$$c_{x,p}/w_p \quad \text{and} \quad (h_p - c_{y,p})/h_p \tag{11}$$

respectively. Note that for Eq. (11) we assume the origin in CV is in the upper left corner and that x and y coordinate are scaled differently (due to the *Aspect* block).

C. Virtual Scanning

Once camera is configured as described in Section II-A and once spotlight is prepared as described in Section II-B, the virtual SL scanner is ready.

To simulate 3D scanning one must define a scene of interest by placing a 3D object in the scanners FOV. Then, for each image of selected SL pattern one image must be rendered. Described simulation procedure is best scripted in Blender using Python. A script we have found convenient is one which sequentially takes images comprising the SL pattern from user selected input directory, which then adjusts the appropriate element of the node tree shown in Fig. 2 to project the pattern, and which then outputs the rendered image(s) into user specified output directory.

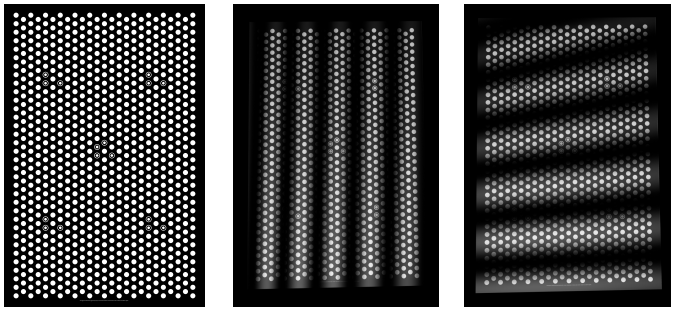
D. Calibration of Virtual Scanner

Virtual SL scanner can also be calibrated using any standard calibration procedure [4], [23]. One simple choice is to use a planar calibration board comprised of white circles as show in Fig. 4. Such a board enables simultaneous camera and projector calibration and is easily simulated in Blender by a flat textured surface where texture contains the calibration pattern. The size of the rectangle must be set to correspond to the actual physical size of the calibration board, and the rectangle is textured using a node tree shown in Fig. 5.

To get a calibration data SL scanning must be simulated for several positions of the calibration board which can again be scripted in Blender using Python. As simulation is perfect w.r.t. the imaging model we have found that recording three positions is almost always sufficient; more positions may be needed if advanced capabilities of Blender to simulate depth of field are used.

E. Geometrical Invariants

To quickly verify the correctness of the simulation we propose to use geometrical invariants. Proposed invariants can be computed quickly and avoid direct image to image or point cloud to point cloud comparisons as was done in [15]. We note



(a) (b) (c)

Fig. 4. Planar calibration board (a), see [23] for details. Projection of two SL patterns containing vertical (b) and horizontal (c) fringes.

Fig. 5. Node tree for a flat rectangle to simulate a planar calibration board.

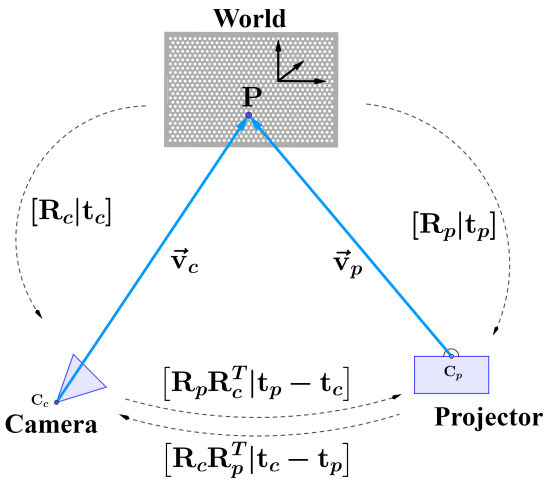


Fig. 6. Relative poses of camera and projector w.r.t. the world coordinate frame. Relative position between camera and projector is invariant to the change of the world frame.

here that direct comparison of simulated data is difficult as matching world frames of real and simulated scanner is hard, hence data registration is always used as an intermediate step.

Obtained extrinsic parameters are expressed in different coordinate frames, one of a hardware scanner (real frame, denoted R) and another of Blender (virtual frame, denoted B), hence derived values which are used for verification must be invariant with respect to the change of the coordinate frame. Invariants for verifying extrinsic parameters are one describing relative position of projector to camera, or vice versa, see Fig. 6. Specifically, these are distance between positions of camera and projector, and relative orientation change between camera and projector. Let d_R be the distance in the real frame and let d_B be the distance in the virtual frame. Then,

$$\begin{aligned} d_R &= \left|\left|\mathbf{R}_{c,R}\,\mathbf{t}_{c,R} - \mathbf{R}_{p,R}\,\mathbf{t}_{p,R}\right|\right|_2 \\ d_B &= \left|\left|\mathbf{R}_{c,B}\,\mathbf{t}_{c,B} - \mathbf{R}_{p,B}\,\mathbf{t}_{p,B}\right|\right|_2 \end{aligned}, \tag{12}$$

and translation error between real and simulated scanner is

$$e_{\text{translation}} = d_R - d_B. \tag{13}$$

Similarly, the orientation error can be defined as the rotation required to perfectly match relative orientations between camera and projector. Let the relative orientations of projector to camera be

$$\begin{aligned} \mathbf{R}_R &= \mathbf{R}_{c,R}\,\mathbf{R}_{p,R}^T \\ \mathbf{R}_B &= \mathbf{R}_{c,B}\,\mathbf{R}_{p,B}^T \end{aligned}, \tag{14}$$

then to perfectly align them we need additional rotation $\mathbf{S}$ such that

$$\mathbf{R}_R = \mathbf{S}\mathbf{R}_B \quad \text{so} \quad \mathbf{S} = \mathbf{R}_R\mathbf{R}_B^T. \tag{15}$$

Angular rotation error can then be defined as

$$e_{\text{rotation},\theta} = \arccos\left(\tfrac{1}{2}\text{Trace}(\mathbf{R}_R\mathbf{R}_B^T) - \tfrac{1}{2}\right). \tag{16}$$

Alternatively, a Frobenius norm may also be used so

$$e_{\text{rotation},F} = \left|\left|\mathbf{R}_R - \mathbf{R}_B\right|\right|_F, \tag{17}$$

although we note that it is more difficult to interpret.

Considering intrinsic parameters geometric invariants are horiznotal α_h, vertical α_v, and diagonal α_d angles of view which define FOV. Let f be the focal distance and let d be horizontal, vertical or diagonal size of the sensor. Then the corresponding angle of view is then computed as

$$\alpha = 2\arctan\frac{d}{2f}, \tag{18}$$

and the absolute angular error is simply the difference

$$e_{\text{FOV}} = \left|\alpha_B - \alpha_R\right|. \tag{19}$$

Proposed geometrical invariants can be used quickly verify if the geometrical setup of a real hardware scanner matches the setup of a virtual scanner.

III. Results and Discussion

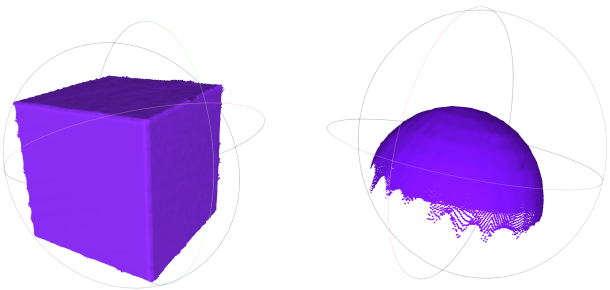
A. Results

We have simulated a SL scanner comprised of one camera and one projector which were defined as described in Section II. Camera parameters were set to simulate Flir's GS3-U3-23S6C-C camera and spotlight parameters were set to simulate Mitsubishi EW230U-ST projector.

This virtual SL scanner was first calibrated using three simulated positions of the calibration board. Reprojection error for pinhole model without lens distortion was 0.17 ± 0.27 px and with radial distortion was 0.16 ± 0.28 px, which is within expectations.

Considering the proposed invariants they were computed directly from Blender's data and from obtained calibration data: distances of Eq. (12) are $d_{\text{B}} = 763.2$ mm and $d_{\text{B}} = 764.7$ mm which results in translation error of 1.5 mm; the angular rotation error of Eq. (16) is $2.25°$ and Frobenius norm error is 0.056; considering FOVs the absolute angular error given by Eq. (19) for camera is $0.025°$ and for projector is $0.98°$. The proposed invariants are therefore within expectations with a somewhat higher rotational error which is most probably due to the minimal number of simulated positions of the calibration board.

In addition to calibration we have performed SL 3D scanning of two virtual objects, a cube with side length of 70 cm and a sphere with diameter 69.7 cm. Reconstructed point clouds for two objects are shown in Fig. 7. To assess the quality of the reconstruction we have fitted a plane to one of reconstructed cube's sides and a sphere to the reconstructed sphere. The average absolute distance of points in the cube's point cloud from the ideal plane is 0.91 mm, and the average absolute distance of points in the sphere's point cloud to the ideal sphere is 1.62 mm. Both errors are within acceptable range and are compatible with values we have obtained previously when using a real SL scanner for similar measurements.



(a) (b)

Fig. 7. Point clouds of reconstructed cube (a) and sphere (b).

B. Discussion

Using Blender [22] or any other CG software for simulating SL scanning is an attractive alternative to a hardware SL scanner as the geometrical setup of a SL scanner is easily and reliably replicated. However, there are several remaining issues one must carefully address and which we will discuss here: (a) projector as a spotlight and software incompatibilities; (b) physically based rendering; and (c) side effects which are unimportant in CG.

First, a projector must be simulated using a textured spotlight while in CV projector is nowadays most often modelled as an inverse camera [23]. To complicate the issue further, there are incompatibilities between spotlight implementations, e.g. a simulation prepared in [12] for Blender v2.79a cannot directly be used in more recent versions of Blender and it had to be adopted to Blender v2.90 in [13]. Interestingly, this issue of how exactly the projector is modelled is mostly ignored in [10], [15] which are two most recent works discussing how Blender is used; the most probable reason is that the intention was not replicating an existing 3D scanner but generating a dataset for machine learning.

Second, to correctly simulate SL scanning a physically based rendering engine [26] must be used. In Blender this means using a Cycles renderer [25] and setting all the relevant parameters correctly. One of more important parameters is the limit to light ray bounces which must be set sufficiently high so inter-reflections in the scene are correctly modelled. An alternative renderer to Cycles is LuxCoreRender [28] which supports light refractions and caustics; it should be the renderer of choice for simulating imaging through glass or underwater imaging [29]. Unfortunately, switching between renderers [25], [28], [30] may alter the specifics of how a spotlight is modelled, so one must always be careful when modelling a projector. Finally, incompatibilities between different versions are again present.

Thirdly, all CG software due to its construction cannot correctly simulate other important problems in SL scanning first of which is the synchronization between camera and projector [31]. In Blender spotlight and camera are perfectly synchronized and simulating unwanted effects of DLP projector's color wheel would require a substantial effort. Similarly, there is no gamma mismatch and color calibration is always perfect. Overall, using CG software to simulate SL scanning will always produce images which are significantly better than is obtainable using real world hardware.

Considering all discussed issues, especially ones concerning the spotlight, we strongly suggest that once the simulation is prepared that the virtual SL scanner is calibrated using any standard calibration procedure [4], [23] and that calibration of virtual SL scanner is then compared to the calibration of real SL scanner using the invariants proposed in Section II-E. We feel this verification is necessary as due to the complexity of involved software the possibility of unintentional errors is not negligible and most of them will be reflected in discrepancies in invariants. Next, regarding software incompatibilities a redeeming quality of Blender is that all previous versions are freely available so when preforming simulation of SL scanning in Blender we recommend the version of Blender and the used renderer be explicitly stated so the simulation can be more easily replicated.

IV. Conclusion

We have presented a detailed guide on how to simulate a structured light 3D scanner using Blender and have provided formulas for parameter conversion between standard parameters used in computer vision and ones used in computer graphics. We have also proposed a set of geometrical invariants which can be used to quickly verify that the simulation in Blender matches to a real structured light scanner. Overall, Blender is a convenient solution to simulate SL 3D scanning, however, as more subtle distortions from the imaging model are not simulated its use to assess the robustness of scanning solution must be further investigated. For future work we also plan to investigate how to use Blender to simulate SL 3D scanning for underwater imaging.

Acknowledgment

This work has been supported by the Croatian Science Foundation under the grant number HRZZ-IP-2019-04-9157 (3D-CODING).

References

[1] J. Geng, "Structured-light 3d surface imaging: a tutorial," *Adv. Opt. Photon.*, vol. 3, pp. 128–160, Jun 2011.

[2] J. Salvi, S. Fernandez, T. Pribanic, and X. Llado, "A state of the art in structured light patterns for surface profilometry," *Pattern Recognition*, vol. 43, no. 8, pp. 2666–2680, 2010.

[3] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[4] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[5] J. D. Foley, F. D. Van, A. Van Dam, S. K. Feiner, J. F. Hughes, and J. Hughes, *Computer graphics: principles and practice*, vol. 12110. Addison-Wesley Professional, 1996.

[6] A. M. Sá, P. C. P. Carvalho, and L. Velho, "Structured light color boundary coding for 3d photography.," in *VMV*, pp. 299–306, 2002.

[7] F. Li, B. Zhang, G. Shi, Y. Niu, R. Li, L. Yang, and X. Xie, "Single-shot dense depth sensing with color sequence coded fringe pattern," *Sensors*, vol. 17, no. 11, 2017.

[8] Z. Yan, L. Yu, Y. Yang, and Q. Liu, "Beyond the interference problem: hierarchical patterns for multiple-projector structured light system," *Appl. Opt.*, vol. 53, pp. 3621–3632, Jun 2014.

[9] G. Birch, A. Dagel, B. Kast, and C. Smith, "3d imaging with structured illumination for advanced security applications," *Sandia Report SAND2015-7936*, 2015.

[10] F. Wang, C. Wang, and Q. Guan, "Single-shot fringe projection profilometry based on deep learning and computer graphics," *Opt. Express*, vol. 29, pp. 8024–8040, Mar 2021.

[11] W. Eikrem, "6d synthetic data generation pipeline with digital representation of structured light sensor," Master's thesis, NTNU, 2021.

[12] A. Puljčan, "Generiranje umjetnih podataka za učenje dekodera strukturiranog svjetla," June 2018.

[13] M. Maglić, "Simulacija oslikavanja strukturiranim svjetlom u blenderu i kalibracija virtualnog projektora i kamere," Jan. 2021.

[14] Y. Wu, V. Boominathan, X. Zhao, J. T. Robinson, H. Kawasaki, A. Sankaranarayanan, and A. Veeraraghavan, "FreeCam3D: Snapshot structured light 3D with freely-moving cameras," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 309–325, Springer International Publishing, 2020.

[15] Y. Zheng, S. Wang, Q. Li, and B. Li, "Fringe projection profilometry by conducting deep learning from its digital twin," *Opt. Express*, vol. 28, pp. 36568–36583, Nov 2020.

[16] S. Fernandez and J. Salvi, "A novel structured light method for one-shot dense reconstruction," in *2012 19th IEEE International Conference on Image Processing*, pp. 9–12, 2012.

[17] Q. Zhang, X. Su, L. Xiang, and X. Sun, "3-d shape measurement based on complementary gray-code light," *Optics and Lasers in Engineering*, vol. 50, no. 4, pp. 574–579, 2012. Computational Optical Measurement.

[18] H. Kawasaki, R. Furukawa, R. Sagawa, and Y. Yagi, "Dynamic scene shape reconstruction using a single structured light pattern," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[19] Budianto, *Robust Fringe Projection Profilometry*. PhD thesis, The Hong Kong Polytechnic University, 2016.

[20] RenderMan. https://renderman.pixar.com/. [online; accessed: May 2022].

[21] 3ds Max. https://www.autodesk.com/products/3ds-max/overview. [online; accessed: May 2022].

[22] Blender. https://www.blender.org/. [online; accessed: May 2022].

[23] T. Petković, S. Gasparini, and T. Pribanic, "A note on geometric calibration of multiple cameras and projectors," pp. 1157–1162, 09 2020.

[24] S. Zhang and P. S. Huang, "Novel method for structured light system calibration," *Optical Engineering*, vol. 45, no. 8, pp. 1–8, 2006.

[25] Cycles. https://www.cycles-renderer.org/. [online; accessed: May 2022].

[26] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[27] E. Medeiros, H. Doraiswamy, M. Berger, and C. T. Silva, "Using physically based rendering to benchmark structured light scanners," *Computer Graphics Forum*, vol. 33, no. 7, pp. 71–80, 2014.

[28] LuxCoreRender. https://luxcorerender.org/. [online; accessed: May 2022].

[29] D. Zoraja, T. Petković, T. Pribanić, and J. Collad, "Projector calibration in a two-layer flat refractive geometry for underwater imaging," in *MIPRO 2022 45th Jubilee International Convention*, pp. 1069–1074, 05 2022.

[30] Intel® oneAPI Rendering Toolkit. https://www.intel.com/content/www/us/en/developer/tools/oneapi/rendering-toolkit.html. [online; accessed: May 2022].

[31] T. Petković, T. Pribanić, M. Đonlić, and N. D'Apuzzo, "Software synchronization of projector and camera for structured light 3D body scanning," in *Proceedings of the 7th International Conference on 3D Body Scanning Technologies*, 2016.