Blazer: Laser Scanning Simulation using Physically Based Rendering

Sebastian Grans1 and Lars Tingelstad

1Department of Mechanical and Industrial Engineering
Norwegian University of Science and Technology

Abstract

Line laser scanners are a sub-type of structured light 3D scanners that are relatively common devices to find within the industrial setting, typically in the context of assembly, process control, and welding. Despite its extensive use, scanning of some materials remain a difficult or even impossible task without additional pre-processing. For instance, materials which are shiny, or transparent.

In this paper, we present a Blazer, a virtual line laser scanner that, combined with physically based rendering, produces synthetic data with a realistic light-matter interaction, and hence realistic appearance. This makes it eligible for the use as a tool in the development of novel algorithms, and in particular as a source of synthetic data for training of machine learning models. Similar systems exist for synthetic RGB-D data generation, but to our knowledge this the first publicly available implementation for synthetic line laser data. We release this implementation under an open-source license to aid further research on line laser scanners.

1. Introduction

A laser scanner is a type of structured light 3D scanner consisting of a camera in conjunction with a line laser. Through triangulation, the depth at the point where the laser intersects objects in the scene can be deduced. Laser scanners have many applications, especially in industry, but also in cultural heritage preservation. Within industry, applications range from part specification validation, assembly, general automation, and welding to name a few[18].

Within cultural heritage preservation, one of the more famous cases is The Digital Michelangelo Project [12], wherein they used laser 3D scanners to digitize large marble statues. The Smithsonian Institute is another case worthy of mention, as they currently have more than 1700 historic artifacts digitized [3], with laser scanning being one of their 3D reconstruction methods [23].

Despite the relatively long history of laser scanners being actively used, there are still some materials that are very difficult to correctly 3D reconstruct. These materials are typically reflective, transparent, or produce a lot of subsurface scattering [21, 7, 6]. Even though the laser line position might be completely obvious to a human observer, algorithms struggle to isolate it correctly, leading to biased reconstructions at best. In practice, difficult surfaces are often sprayed with anti-reflective coating [21], but this might not always be an option. For instance, when historic artifacts are being scanned, or when the spray might interfere with subsequent processes, such as welding. Hence, further research on improved laser line extraction methods is needed, and this system is designed to aid in that process.
Physically based rendering techniques combine ray-tracing methods, with physics based light-interaction models to produce photorealistic results. These techniques have previously been limited to large scale video productions due to the large computational cost. In recent years, however, graphics hardware has gotten better and cheaper, and now come with hardware-based ray tracing acceleration. This has made physically based rendering techniques (PBR) to be available even on consumer-level hardware. In addition to this, there are free and open-source 3D graphics software such as Blender® [2], which comes with paired with Cycles, a PBR engine. Hence, enabling anyone to produce photorealistic 3D artwork and simulations. In this paper, we leverage the capabilities of Blender to create a virtual laser scanner that can produce simulated scans with high realism. One application of these simulated scans is for training of neural networks.

Producing real datasets for machine learning applications consumes a lot of time and human resources. Since humans are doing the labeling, the accuracy is also limited. One of the benefits of synthetic data is that it requires less human intervention to produce, and comes with ground truth information by default, free of human bias.

Synthetic images from the system can also be very useful when developing traditional algorithms. In the context of software development, it is common to have unit testing as part of the development cycle, which ensures that new code does not break previous functionality. Using the type of system presented in this paper, it is very easy to make a base-line test set that the algorithm should always be able to process correctly. As development goes on, one can also easily add new cases with entirely new materials without having to acquire a real sample. Furthermore, synthetic data makes it possible to quantitatively measure the accuracy of an implementation which is typically very difficult to achieve on a physical system.

In both of these cases, it is key that the synthetic data is a good representation of the real data. This is in the field known as the reality gap.

The main contribution of this paper is the development of an open-source and physically accurate virtual laser scanner implemented in Blender. In Fig. 1 an example of how the virtual setup can be used is shown. The system has the potential to be used as a testing tool for new line laser algorithms, or for validation of existing ones. Another key application is synthetic data generation for training of machine learning models. Similar systems exist for synthetic RGB-D data generation, but to our knowledge, this is the first publicly released implementation designed for the development of line laser scanners.

A note on the notation is used in this paper: Bold upright symbols refer to matrices. Bold italic symbols refer to vectors, except for the zero vector, 0, which is upright.

When matrices are written with indices, such as $M_{ij}$, then the indices indicate between which coordinate frames it transforms. For instance, the product

$$p_i = M_{ij}p_j$$

describes the change of basis of a point $p$ from the coordinate frame $j$ to $i$. Common frames in this paper are $\{w\}$, $\{c\}$, and $\{i\}$, which represent the world, camera, and image frames respectively. Vectors and scalars are sometimes also explicitly written with such an index to indicate in which frame it is represented.

The structure of the paper is as follows: In Section 2 we give an overview of previous work that has been done in the field of synthetic data generation. We then proceed with Section 3 wherein we give a brief introduction to what physically based rendering is and the maths related to laser vision systems. A description of how the virtual laser scanner was constructed is presented in Section 4 followed by Section 5 where the details related to how various experiments were performed are contained. The results of these investigations are then presented and discussed in Section 6 followed by the conclusions in Section 7. Finally, in Section 8 we present some of the outlooks for the system presented in this paper.

The key components of this work, such as relevant source code and the Blender file containing the virtual laser, are released under the MIT license and are available at https://github.com/SebastianGrans/Blazer.

## 2. Related work

To our knowledge, the closest related work is that of Abu-Nabah et al. [1], which implemented a similar system in the 3D graphics software Autodesk 3ds Max®. They evaluated the system, and the related algorithms, by comparing the virtual one with a real scanner setup. The main difference between our work and theirs is that we focus on realism through PBR to achieve as small of a domain gap as possible, for use in neural network training. We also highlight how our system can be used for developing novel algorithms for difficult materials, in particular reflective ones. In contrast, our system is implemented in the free and open-source 3D software Blender and made publicly available under the MIT license. In [17], they implemented a projector-based structured light system in a PBR engine. Using the ground truth data given by the simulation, they could perform quantitative evaluations on various encoding schemes.

### 2.1. Synthetic data

One of the main purposes of our system is synthetic data generation for machine learning applications. The use of
synthetic data is not a new concept in the field as it is a method to cheaply create and augment a dataset. The earliest approaches were based on rendering a 3D object on top of a random background image [26]. This simplistic approach did however not transfer well to real images, as the reality gap was too large. Later works have been done where entire scenes rendered in 3D were used, which have achieved better results [13, 29]. The concept of using simulations for training neural networks to then be applied to real data is known as ‘sim2real’. The concept and its validity for use in robotic applications was heavily discussed at the 2020 Robotics: Science and Systems conference [11].

There currently exist two Blender-based pipelines for synthetic data, namely BlenderProc [4] and BlendTorch [9]. The former is an offline renderer that focuses on realism through physically based rendering and was used for generating the datasets that were used in the 2020 edition of the BOP challenge [10]. BlendTorch on the other hand is based on real-time rendering and is designed to be used directly inside a PyTorch data loader.

3. Preliminaries

3.1. Physically based rendering

In this section, a very brief introduction to physically based rendering (PBR) is given. For a full description, please refer to [22].

Physically based rendering is the concept of creating an image by simulating how light interacts with the virtual 3D scene by using light-matter interaction models. As will be evident later in the text, this is a very time-consuming process since a large amount of light-rays needs to be simulated. This is contrasted with the rasterization-based rendering techniques in real-time render engines, which are used in video games. These types of render engines use approximations of light transportation and are typically not that good at replicating global illumination which is important for physically accurate renditions of a scene.

In path-tracing algorithms, such as those used by both Cycles [2] and LuxCoreRender [15], light rays are sent out in reverse, i.e., from the camera onto the scene. As the ray hits an object, it will scatter off in a new random direction with a probability described by the materials bidirectional scattering distribution function (BSDF). The ray might then continue to collide with another object, and so on. After a set number of interactions, or if the ray hits a light source, the tracing is terminated. The value of the originating pixel is then a weighted sum of all the materials and lights the ray interacted with. Since the scattering direction is random, each pixel must be sampled multiple times (> 10^2) to create a proper image, otherwise the resulting image will be very noisy and unrepresentative.

The key component to why this results in a physically accurate depiction is due to the BSDF. The BSDF is a general term distribution functions that describe scattering phenomena, such as reflection, transmission, and subsurface scattering. An illustration of which is shown in Fig. 2. These functions must have the following properties so as to be physically representative:

- Reciprocity: The probability of scattering into a certain angle is identical to scattering in the reverse direction.
- Energy conserving: Reflected light must have less or equal amount of energy as the incoming light.

The parameters of the BSDF are most commonly specified by eye until the virtual material looks similar to the real one. This requires a lot of knowledge and experience in order to achieve a realistic result. There are however methods to measure the BSDF of a material [19], and commercial tools and services that perform this also exist [27]. Hence, partially removing the human from the equation.

3.2. Camera model

The pinhole camera model is the most commonly used model and is a central projection model (c.f. parallel projection). It assumes that only light that travels from the observed scene and through the camera’s origin will reach the image plane. It is a mathematical representation of an ideal camera obscura. Mathematically it can be written as

\[ p_i = P p_{iw}, \]  

where

- \( P \) is the projection matrix,
- \( p_{iw} \) is the world coordinate of a pixel.

https://creativecommons.org/licenses/by-sa/3.0
where $P$ is a $3 \times 4$ projection matrix describing the mapping $P : \mathbb{P}^3 \rightarrow \mathbb{P}^2$.

The projection matrix $P$ can be further decomposed into three components

$$P = K[1 \ 0]T_{cw} \ ,$$

where $K$ is the intrinsic camera matrix, $T_{cw}$ the extrinsic camera matrix, and $[1 \ 0]$ is a $3 \times 4$ matrix known as the canonical projection matrix. The extrinsic camera matrix, $T_{cw}$ is an element of the Special Euclidean group $SE(3)$ and describes the change of basis from the world coordinate frame into the camera frame. The canonical projection matrix projects the point onto a normalized image plane lying at the unit focal length.

Finally, we have the intrinsic camera matrix, $K$, which describes the transformation of the normalized image plane into the image plane which is what we typically work with, namely, in pixels units with the origin in the top left corner. It consists of the following parameters,

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

The parameters $f_x$ and $f_y$ are typically referred to as focal lengths and are expressed in pixels per meter and can hence be thought of as a description of pixel density in the sensor’s $x$ and $y$ direction. For most cameras, these are equal, but due to various reasons they might differ. For instance, a sensor with non-square pixels, the use of an anamorphic lens, or if the image plane is not parallel with the focal plane. The latter is sometimes artistically intentional and is referred to as tilt photography.

The $s$ parameter refers to the skew of the image plane, in other words, if the image plane has a rhomboid shape. This is typically zero in modern cameras.

The parameters $c_x$ and $c_y$ describe the origin of the image plane, also known as the principal point. This point translates the image frame origin to the upper left corner, which is typical when working with digital images.

Additionally, real camera systems have lenses that introduce various distortion. Lens distortions are typically centered at the axis of projection and hence introduced before applying the extrinsic matrix. We express this as the application of the function $\Delta$ to the normalized image coordinate $\tilde{p}_i$ as

$$p_i = K\Delta(\tilde{p}_i) .$$

The distortion function is composed of a set of different types of distortions [30]. The most common ones that are considered by default in common camera calibration libraries are radial and tangential distortions.

Lenses work by refracting light, and the refraction index of a medium is wavelength-dependent, which can be realized by shining white light through a prism. This leads to wavelength-specific lens distortions such as chromatic aberration and chromatic focal shift. Lenses are typically designed with this in mind in order to minimize the effect. Lens distortion corrections used in this paper will be considered independent of wavelength.

3.3. Camera calibration

Camera calibration is the task of finding the parameters of the intrinsic and extrinsic matrix. There are various methods, but the gold standard is to use Zhang’s method which only requires a planar calibration target, and hence easy to produce to high accuracy [32].

The calibration target employed in Zhang’s method can be any planar shape with uniquely identifying features of known relative position such that point correspondences can be determined between the image plane and the world coordinate frame. The most common calibration target is a checkerboard type target, and automatic feature detection is implemented in various software packages and libraries.

The calibration method involves taking $(\geq 3)$ photos of the planar calibration target in various poses inside the field-of-view of the camera which gives enough constraints to determine all the intrinsic and extrinsic parameters.

3.4. Parametric representation of lines & planes

A line can be described by the set of points

$$L = \{ p = l_0 + \lambda v \mid \lambda \in \mathbb{R} \} ,$$

where $l_0$ is a point on the line, and $v$ a vector which describes the direction of the line. A plane can then be described as the span of two intersecting non-parallel lines, which gives us an analogously set

$$\pi = \{ p = p_0 + \alpha v_1 + \beta v_2 \mid \alpha, \beta \in \mathbb{R} \} .$$

However, it is more common to use the point-normal form:

$$\pi = \{ p \mid n \cdot (p - p_0) = 0 \} ,$$

where $n$ is the normal vector perpendicular to the plane and $p_0$ an arbitrary point on the plane. Given that $n = (a, b, c)$, $p_0 = (x_0, y_0, z_0)$, and $p = (x, y, z)$, (4) can be written as

$$(a, b, c) \cdot (x - x_0, y - y_0, z - z_0) = 0$$

$$ax + by + cz + d = 0 ,$$

where

$$d = -(ax_0 + by_0 + cz_0) .$$

Hence, a plane can also be described by the four-vector $\phi = (a, b, c, d)^T$.

3.5. Laser plane calibration
Similar to how camera calibration is the process of determining the parameters describing the camera, laser plane calibration is the operation of determining the point-normal vector, \( \mathbf{p} = (a, b, c, d)^T \).

The basic principle of laser plane calibration is to identify at least three non-collinear points lying on said plane. For increased precision and robustness to noise, we typically use many more points than this.

This laser calibration pipeline is similar to that of regular camera calibration. The camera and laser plane calibration can be combined into a single calibration step that utilizes the same images.

Calibration begins by acquiring several pictures of the laser line intersecting the checkerboard target. For each image, we say that the calibration target defines the world coordinate frame of that image and arbitrarily chooses it to lie at \( z = 0 \). The inner checkerboard corners are then found in the image, for which we then know the corresponding point \( p_i \) in world coordinates.

The mapping of the inner checkerboard corners from the calibration plane to the image plane is defined by a planar homography, \( \mathbf{H}_{iw} \). Such that

\[
p_i = \mathbf{H}_{iw} \mathbf{p}_w ,
\]

where \( \mathbf{p}_w = (x_w, y_w, 1) \) and \( p_i = (x_i, y_i, 1) \) are the world and image coordinates respectively represented in homogeneous form. The homography can be estimated using a direct linear transform (DLT) algorithm \[8\], which is then further refined by performing the following nonlinear minimization

\[
\min_{\mathbf{H}_{iw}} \sum_i \left\| p_i - \mathbf{H}_{iw} \mathbf{p}_w \right\|^2_2
\]

using the Levenberg-Marquardt algorithm \[32\].

The homography is related to the projection formula \[2\] by:

\[
\mathbf{H}_{iw} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \mathbf{K} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} .
\]

Where the right most matrix is a reduced form of the transformation matrix in \[2\], where the third column, \( r_3 \), has been discarded. This is valid since we have defined our calibration target to lie at \( z = 0 \), and any transformation leaves the \( z \)-component unchanged.

Rearranging \((3.3)\) yields:

\[
\begin{align*}
r_1 &= \lambda \mathbf{K}^{-1} h_1 \\
r_2 &= \lambda \mathbf{K}^{-1} h_2 \\
r_3 &= r_1 \times r_2 \\
t &= \lambda \mathbf{K}^{-1} h_3
\end{align*}
\]

with

\[
\lambda = 1 / ||\mathbf{K}^{-1} h_1|| = 1 / ||\mathbf{K}^{-1} h_2|| .
\]

The last column, \( r_3 \), of the rotation matrix can be found as the cross product of the first two columns since elements from \( \text{SO}(3) \) are orthonormal \[16\].

Thus we can construct the rotation matrix \( \hat{\mathbf{R}}_{cw} \) as

\[
\hat{\mathbf{R}}_{cw} = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}
\]

As described by Zhang \[32\], due to noise, the resulting matrix rarely fulfill the requirements of being a rotation matrix, i.e. \( \text{det} \hat{\mathbf{R}}_{cw} \neq 1 \). By performing the following minimization,

\[
\min_{\mathbf{R}_{cw}} \left\| \mathbf{R}_{cw} - \hat{\mathbf{R}}_{cw} \right\|_F ,
\]

with the constraint that \( \mathbf{R}_{cw}^T \mathbf{R}_{cw} = \mathbf{I} \). Then we determine the closest proper rotation matrix to \( \hat{\mathbf{R}}_{cw} \), in terms of the squared Frobenius norm. The full transformation \( \mathbf{T}_{cw} \) is thus given as

\[
\mathbf{T}_{cw} = \begin{bmatrix} \mathbf{R}_{cw} & t \\ 0 & 1 \end{bmatrix} . \tag{5}
\]

For each image, the image coordinates of the laser line lying inside the calibration plane is then extracted and refined to sub-pixel accuracy. It can then be projected back onto the the world coordinate plane using the inverse of the previously described homography.

\[
\mathbf{p}_w = \mathbf{H}_{iw}^{-1} p_i \\
= \mathbf{H}_{w} \mathbf{p}_i
\]

By abuse of notation, \( \mathbf{p}_w \) is subsequently transformed into the camera frame using \( \mathbf{T}_{cw} \). This is possible since \( \mathbf{p}_w \) has an implicit \( z \) coordinate of zero, as was the

\[
\mathbf{p}_c = \mathbf{T}_{cw} \mathbf{p}_w .
\]

This is beneficial since the camera frame is a constant frame of reference shared by all views of the calibration plane.
Once the coordinates of the laser line in each view has been determined and transformed into the camera coordinate frame, we can relatively easily determine the parameters of the plane by setting up the following over-determined homogeneous system of linear equations:

\[
\begin{bmatrix}
    x_{11} & y_{11} & z_{11} & 1 \\
    \vdots & \vdots & \vdots & \vdots \\
    x_{ij} & y_{ij} & z_{ij} & 1
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d
\end{bmatrix}
= 0 ,
\]

where \( \phi \) refers to the \( i \)th coordinate in the \( j \)th view. Similar to before, and initial guess of the solution can be found using singular value decomposition and then refined further with Levenberg-Marquardt nonlinear optimization.

We optimize using the following point wise error function \( \epsilon_i \) on the laser line,

\[
\epsilon_i = \frac{|\hat{p}_{ij} \cdot \hat{n}|}{||\hat{n}||} + (||\hat{n}|| - 1)^2 .
\]

Where \( \hat{p}_{ij} = (\hat{a}, \hat{b}, \hat{c}, \hat{d})^T \) is the estimated plane parameter vector, and \( \hat{n} = (\hat{a}, \hat{b}, \hat{c}) \) is the estimated normal vector.

The first term in the error function is simply the shortest distance from the point to the plane, and the second term is a penalty term that ensures a unit length normal vector to avoid the trivial solution.

### 3.6. 3D reconstruction

Given that we have a calibrated the camera and been able to identify the four parameters that describe the laser plane, then we can fully reconstruct the corresponding 3D position of a point on the plane which represents the laser line.

This is possible since any point \( p_i = (x_i, y_i, 1) \in \mathbb{P}^2 \) in the image plane describes a line that intersects the camera origin and the image plane. If \( p_i \) also corresponds to a point in the image plane where the laser line is visible, then the line it describes must also intersect with the laser plane at some point, \( p_c \). This point is what we want to recover since it lies on the 3D geometry we wish to know the position of.

In order to express the point \( p_i \) as a line in space, we first need to transform it into the normalized image plane. This is achieved by multiplying it with the inverse camera matrix, \( K^{-1} \).

\[
\hat{p}_i = \begin{bmatrix}
    \hat{x}_i \\
    \hat{y}_i \\
    1
\end{bmatrix} = K^{-1} \begin{bmatrix}
    x_i \\
    y_i \\
    1
\end{bmatrix}
\]

where

\[
K^{-1} = \begin{bmatrix}
    \frac{1}{f_x} & -\frac{s_x}{f_x f_y} & \frac{c_x}{f_x} & \frac{c_x}{f_x} \\
    0 & \frac{1}{f_y} & \frac{s_y}{f_y f_x} & \frac{c_y}{f_y} \\
    0 & 0 & 1 & 1
\end{bmatrix}
\]

The normalized image coordinate, \( \hat{p}_i \), represents the direction vector, \( v \), in the line equation \( (1) \). And since this line intersects the origin, we select that as our arbitrary point \( l_0 \).

This means that the intersection, \( p_c \), between this line and the laser plane must be a scalar multiple of

\[
p_c = \lambda v = \lambda \hat{p}_i
\]

Substituting this into \( (1) \) gives us

\[
n \cdot (l_0 + \lambda v - p_0) = 0 .
\]

And if we solve for \( \lambda \) we get:

\[
\lambda = \frac{p_0 \cdot n}{v \cdot n} \quad \text{(8)}
\]

(Note that if \( v \cdot n = 0 \), then the plane and the line is parallel to each other, and if \( l_0 = p_0 \), then the line is contained inside the plane. These cases can easily be avoided in a real-world system, by making sure that the laser plane is: not parallel with the camera’s z-axis, and not intersecting the camera origin.)

The point of line-plane intersection can then be calculated by inserting \( (8) \) into \( (7) \), which yields:

\[
p_c = \frac{p_0 \cdot n}{v \cdot n} \hat{p}_i \quad \text{(9)}
\]

If the plane is known only only in its four vector form, \( \phi = (a, b, c, d) \), then \( p_0 \) can be calculated as

\[
p_0 = -\frac{dn}{||n||^2}
\]

which represents the point closest to the plane.

### 4. Implementation

The laser scanner itself is implemented inside of Blender version 2.92, but older versions are also supported. We utilize both the built-in render engine Cycles, as well as the external render engine, LuxCoreRender, which is made available inside of Blender via the BlendLuxCore plug-in. Some features were also using the Python API which is accessible via the built-in scripting interface. Software related to calibration, 3D reconstruction, etc. were all implemented as Python scripts outside of Blender.

#### 4.1. Virtual Camera

The virtual camera is Blender’s default camera type with its parameters set to match a physical one. The emulated camera is an Omron Sentech STC-MCS500U3V equipped with a 12 mm lens set to an aperture of \( f/1.4 \) and focused at 1 meter. The camera outputs images with a resolution of 2448×2048, which with a physical pixel size of 3.45 μm corresponding to a sensor size of 8.4456×7.0656 mm.
4.2. Virtual laser

The virtual laser is implemented as a generic line laser, albeit inspired by a model Z20M18H3-F-450-lp45 line laser from Z-LASER GmbH, which has an output power of 20 mW at 450 nm. Neither Cycles [2] nor LuxCoreRender [15] is a spectral rendering engine, and hence the specific wavelength is not directly applicable. Instead, the color of the virtual laser is specified by eye. Due to differences in functionality between the two render engines, the line laser was implemented in two different fashions.

4.2.1 Cycles

In the Cycles render engine, the laser line is implemented by using a spotlight with an emission shader and a laser intensity mask. The mask is generated procedurally inside Blender by building up a node network, which is a form of visual programming. The procedural approach makes the virtual laser very flexible and can easily be tuned to match a real one.

A spotlight in Blender radiates light spherically from its origin, but only light inside a cone centered around the negative z-axis is emitted onto the scene. The direction of the light can be accessed via the normal vector, which we first normalize by dividing by the z-component. This essentially projects the vector onto the \( z = 1 \) plane.

The intensity profile of the cross-section of a line laser is typically Gaussian. By applying a Gaussian function to our normal vectors w.r.t. the \( x \)-component results in an intensity mask that represents our laser line. In this case, we use a unit amplitude Gaussian which has the form:

\[
g(x) = \exp \left( -\frac{x^2}{2\sigma^2} \right),
\]

where \( \sigma \) determines the perpendicular spread of the laser line.

The resulting mask, however, results in a physically inaccurate power value. To make the virtual laser perceptually similar to a real one, the power needs to be specified in watts, rather than milliwatts. In the spirit of physical correctness, a correction factor is calculated such that the value has a physical meaning.

The cause for this is due to how a spotlight is implemented in Blender. The power specified does not represent the visually emitted power, but rather the output power of a point light source. This might be artistically useful, as changing the cone angle does not change the light intensity, but not in this case. To compensate for this we must scale the intensity mask such that the intensity integral is equivalent to that of the unit sphere.

The point light source has an intensity of one in all directions, and its integrated intensity is the surface area of a unit sphere, i.e. 4\( \pi \).

For the laser line we find the integral as follows

\[
\int_{-\gamma}^{\gamma} \int_{-\infty}^{\infty} g(x) \, dx \, dy = 2\gamma\sigma\sqrt{2\pi}.
\]

Where \( \gamma = \tan(\theta_c/2) \) which is the maximum y-value of the projected normal vectors given the specified spotlight cone angle, \( \theta_c \).

In the \( x \)-direction we the improper integral as an approximation of the proper Gaussian integral which is valid since the length of the laser is much larger than its width and because the Gaussian quickly decays to zero. In the \( y \)-direction we simply integrate over the length of the laser line.

The intensity mask is then scaled by \( \lambda \) which is then the fraction

\[
\lambda = \frac{4\pi}{2\tan(\theta_c/2)\sigma\sqrt{2\pi}}.
\]

Hence, a physically correct intensity.

The \( \sigma \) value is distance dependant, and instead lasers are commonly specified in terms of their divergence angle, \( \theta_l \). The relationship between this angle and the \( \sigma \) value is

\[
\sigma = \frac{\tan(\theta_l/2)}{\sqrt{-2 \ln(1/e^2)}},
\]

which can be used to set the virtual laser to a known value.
LuxCoreRender does not support the use of nodes on light sources, and the line laser cannot be procedurally generated. Instead, a picture representation of a line laser is produced in an image editing software coloring the middle column of pixels in the needed color. Gaussian blurring can be applied but does not seem to improve the rendered appearance of the laser line. The image can then be projected using the spotlight directly. This method of creating a virtual line laser is for obvious reasons not as flexible as that of the Cycles engine and might require tuning by eye to get a laser line that matches the physical one.

4.3. Rendering

Rendering of the scene is separated into two renderings. The first rendering produces the realistic RGB image that is expected but also contains other passes such as depth, and surface normals. The second rendering disables any lighting in the scene except for the laser and only renders direct reflections. The luminance value of this rendering corresponds to the ground truth location of where the laser intersects the scene. The renderings are then saved in raw form using the multilayer OpenEXR file format. In addition, the RGB image is also saved in PNG for easy viewing since most operating systems come with built-in viewers for this format. Examples of these output images are shown in Fig. 4. For each render, a Python Pickle file is also generated, which contains the camera matrix and the laser plane parameters. Other data can easily be extracted from Blender via its Python API and included if needed.
5.2. Laser plane calibration

Calibration of the laser plane was only performed in silico. Images of the same virtual calibration target were captured in a similar way as for the camera calibration, albeit ensuring that the laser line was intersecting the target. The plane parameters, \( \phi \), were then found by using a Python implementation of the process described in Section 5.3 with the aid of the libraries OpenCV, SciPy, and NumPy.

5.3. Laser line extraction

The method for extracting the image coordinates of the laser line is a two-step process. First, we find the discrete pixel location of the laser, followed by a sub-pixel refinement by locally fitting a Gaussian function. This is valid as lasers typically have a Gaussian intensity profile.

The discrete pixel location is found by calculating the channel difference image \([28]\), which for a green laser is

\[
I_d = I_g - \frac{I_r + I_g}{2}, \]

but can naturally be arranged to work for red and blue lasers. This amplifies pixels that are primarily green. Values below zero are subsequently discarded before convolving the one-channel image with a \(3\sigma\) Gaussian smoothing filter. From the smoothed image, we subtract by the mean intensity and clamp negative values to zero followed by a global normalization to unit intensity. Intensity values of less than 0.1 are subsequently discarded. The discrete laser location is then found by taking the maximum value of the row-wise normalized image. Excerpts from the pipeline can be seen in Fig. 5.

Sub-pixel accuracy was then achieved by row-wise fitting a Gaussian function to the smoothed channel difference image (before thresholding and normalizing) by using the discrete laser location as an initial location.

5.4. Modeling

For testing of weld profile measurements, a model of a single bevel tee weld joint was modeled inside of Blender. Freely available textures from [cc0textures.com](http://cc0textures.com) were used to give it a realistic appearance. A ground surface texture was applied in and in the proximity of the weld seam, while the rest was textured to appear slightly rusted.

6. Results and discussion

6.1. Camera calibration

The physical camera was calibrated using 30 images resulted in the following camera matrix with a RMS reprojection error of 0.986 px:

\[
\begin{bmatrix}
3479.8 & 0 & 1202.3 \\
0 & 3479.0 & 1010.2 \\
0 & 0 & 1
\end{bmatrix}
\]

And with the following lens distortion parameters:

\[
(k_1, k_2, k_3) = (-0.0785, -0.1658, 2.154) \\
(p_1, p_2) = (-0.0013, -0.0022)
\]

The virtual camera, which was calibrated with 38 images using the same procedure, albeit without lens distortion parameters, achieved a reprojection error of 0.058 px, with the camera matrix:

\[
\begin{bmatrix}
3478.4 & 0 & 1223.8 \\
0 & 3477.1 & 1021.4 \\
0 & 0 & 1
\end{bmatrix}
\]

The true camera matrix from Blender is the following:

\[
\begin{bmatrix}
3478.3 & 0 & 1224 \\
0 & 3478.3 & 1024 \\
0 & 0 & 1
\end{bmatrix}
\]

The resulting camera matrix of both the virtual and physical camera are close to the theoretical one. In the physical calibration, we can see a larger discrepancy in the principal point of around 22 and 14 pixels in \(x\)- and \(y\)-direction respectively. This indicates that the sensor is not perfectly aligned with the lens’ axis of projection. Given that the pixel size, this equates to about 76 and 48 \(\mu\)m, which are reasonable manufacturing tolerances. The similarity of the results signifies that the virtual camera is a close virtual representation of the physical one.

6.2. Laser plane calibration

For calibration, the virtual laser was placed 20 cm in the \(+x\)-axis direction of the camera frame with a 13° rotation inwards around \(y\)-axis. This results in a ground truth point-normal vector with the following (rounded) values:

\[
\phi_{gt} = (9.744 \times 10^{-1}, -6.706 \times 10^{-8}, 2.249 \times 10^{-1}, -1.949 \times 10^{-1})^T
\]

Note that the \(y\)-component should be zero in theory, and the discrepancy most likely comes from rounding errors during floating-point arithmetic. The estimated (rounded) values after calibration were found as

\[
\phi_{est} = (9.743 \times 10^{-1}, 4.095 \times 10^{-4}, 2.254 \times 10^{-1}, -1.954 \times 10^{-1})^T
\]

This corresponds to an angle difference of 0.63 mrad (36 millidegrees) between the ground truth and the estimated normal vector. Such a discrepancy corresponds to about a 1 mm difference at one meter from the pivot point. Higher accuracy can most likely be achieved by using utilizing even more calibration images. The results nevertheless illustrate that this method is viable, even when relatively few images are used.
6.3. 3D reconstruction

As an example, a profile of a weld seam similar to that of Fig. 8b was scanned virtually. The depth is found in three different ways: By triangulating the depth using the coordinate extracted from the RGB image. Through triangulation by using the laser mask (e.g., Fig. 4d), or by using the ground truth depth image. For triangulation, the ground truth plane parameters were used.

The ground truth was calculated by using the sub-pixel accurate locations of the laser mask. The depth values at these coordinates were recovered by interpolating the depth output image. In this example, we used a distance-weighted average of a local 3 x 3 window as the interpolation function.

As can be seen in the leftmost plot in Fig. 6, the naive laser line extraction method being used is not adequate for reflective surfaces. From the second two plots, we can also see that there is a systematic error occurring since the distance error is always negative even without the presence of severe reflective distortion (e.g. top of the image). This bias could stem from the laser extraction method but is not evident upon inspection of the extracted coordinates. This indicates that there might be a bias in either the ground true depth image, or the plane parameters. The plane parameters could for instance be wrong if the virtual laser line projection is off-center from the z-axis. The average (normalized) difference vector between the points triangulated using the ground truth laser mask (blue dots in the plot), and the points reconstructed from the depth image was $(-0.19, 0.01, 0.98)$. This suggests that an off-center projection is indeed the culprit, since the $x$-component is significantly large. Further investigation is nevertheless required.

During the process of making these figures, it was also discovered that the output from the $z$-pass given LuxCoreRender does not refer to the distance in z direction from the camera, but rather the Euclidean distance from the camera. LuxCoreRender does however offer a ‘position’-pass that returns the 3D coordinate for each pixel. The correct depth can easily be recovered from that and is what is used as ground truth in the figure.

In this study, we have not compared the results of the virtual system with that of a real system since this would require highly accurate and calibrated targets. Such a method is described in VDI/VDE 2634(2), which is a standard for evaluating structured light 3D scanners [5]. The standard could be used to further validate this system.

6.4. Reflections

One of the potential applications of this system is for simulated laser scanning of weld seems. On physical weld seems, rust and other surface contaminants are often ground away before welding which yields a relatively reflective surface. As mentioned in the introduction, reflected laser light is difficult to filter out. For this system to be applicable for research in this area we need to ensure that these reflections are captured well. Fig. 8 shows a comparison between a real weld seem sample and renderings of a virtual weld seem, illustrating that the system does indeed handle these reflections properly when the right render engine is used.

In computer graphics, reflections of this type are referred to as ‘caustics’. Path-tracing rendering algorithms in general do not handle caustics that well due to the stochastic na-
6.5. Laser mask

The ground truth laser position is currently produced as a mask which must be further processed to give a sub-pixel accurate laser line location. This might be adequate, but further work should be done to evaluate this. A better option is to find a method for extracting the sub-pixel location directly from Blender which would remove any form of ambiguity.

6.6. Lens distortion

The virtual camera inside of Blender is an ideal pinhole camera and hence does not suffer from lens distortion. This is not a physically accurate representation, and since the goal is to produce data with the least amount of reality gap as possible, it might be needed to include this in the simulation. It is currently only possible to add rudimentary lens distortion as a post-processing step, which might not be a good solution since this would introduce interpolation artifacts. However, when working with a real system, you typically correct your images to remove distortion which also an interpolation step.

Pbrt [22] is another render engine that has the ability to model a lens as a stack of optical elements. The dimensions of the optical elements are typically not disclosed by the manufacturer, so it is unclear how useful this is.

Another option is to use differentiable rendering techniques, such as those described in [20], to generate a virtual lens that produces the same distortion as that of the real lens.

A final approach would be to disregard the traditional camera model and use a generalized camera model which uses a per-pixel parametrization of the camera [24]. This would completely remove the problem of having to consider the complexities of lens distortions. This would be possible to implement in Blender since it is open-source, but would nevertheless require substantial efforts.

6.7. User limitations

A significant limitation of this implementation lies at the user end. In order to produce a virtual object that matches a real one requires both knowledge about 3D modeling, as well as some artistic skill and expertise when choosing the correct texture and BSDF parameters.

This could possibly be mitigated in the future as differentiable techniques mature. For instance, in [25] and [14], they utilize such techniques to reconstruct geometry, texture, and material properties from a sample.

A specific example of where a naive approach might yield poor results is shown in Fig. 7. The outward appearance of the middle row is quite similar, despite that the underlying geometry is vastly different in the number of faces. In the second row, the shading is set to ‘smooth’ where the normal of a certain point on the geometry is an interpolation of the neighboring vertex normals which results in a smooth surface appearance. This has the benefit of achieving a visually appealing render while keeping the number of faces to a minimum and hence reducing the render time. However, as can be seen in the final row where the depth values are visualized, the choice of shading does not change the resulting depth and hence the produce ground truth depth would not be accurate.

Regardless of these user limitations, the papers presented in Section 2 clearly show that even simple synthetic data can be used for machine learning applications. Instead of thinking of synthesized data as being the sole source for training, it might be better to think of it as a form of augmentation to be used in conjunction with real data.

7. Conclusion

In this paper, we have presented Blazer, a virtual 3D laser scanner implemented in Blender®. It leverages both built-in and external physically based rendering engines to create labeled data for training a neural network or validating traditional methods. The implementation is made available at...
The system consists of a virtual camera and line laser, which are implemented to match a real-world system closely. In one of the line laser implementations, care was taken to ensure that the specified parameters had a physical meaning.

Through experiments, we show that a virtual camera can be simulated to match a real camera’s parameters to a high degree. Lens distortion was not included, but we presented suggestions on how this can be implemented and accounted for. The laser plane was also calibrated using traditional methods resulting in parameters with a 36 millidegree plane normal inaccuracy to the ground truth, which indicates that the rendered images are an accurate representation of the virtual system parameters.

The limitations of the system and their potential impact on its suggested applications were discussed, such as how the choice of render engine can result in non-physical results. Ideas on how differentiable rendering techniques could mitigate some of these limitations were also presented and a method for further validation of the system is proposed.

8. Future work

The current implementation of the pipeline requires much knowledge about the intricacies of Blender and the implementation itself. To increase the usability, the authors aim to make this implementation a part of the BlenderProc framework, which would help remove these intricacies via abstraction layers. This will make this work more accessible to researchers working in the sim2real field who might already be familiar with the BlenderProc API.

Future work also entails using synthetic data to train neural networks targeted towards laser scanning tasks. Early tests on laser line extraction trained only with synthetic data show very promising results at transferring the knowledge to the real domain.

Acknowledgments

The first author wants to thank Ola Alstad for all the valuable discussions. Andreas Theiler for his helpful comments on the manuscript. And finally, CG Matter for his inspiring content that made this paper possible.

References

[1] B. A. Abu-Nabah, A. O. ElSoussi, and A. E. K. Al Alami. Virtual laser vision sensor environment assessment for surface profiling applications. Measurement, 113:148–160, 2018.

[2] Blender - a 3D modelling and rendering package. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org

[3] N. Daher. You can now download 1,700 free 3-D cultural heritage models, Mar 2020. URL https://www.smit
T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, Z. Li and N. Snavely. CGintrinsics: Better intrinsic image
S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, E. R. Eiríksson, J. Wilm, D. B. Pedersen, and H. Aanæs. Pre-
R. Hartley and A. Zisserman. Multiple View Geometry in
M. Gupta, A. Agrawal, A. Veeraraghavan, and S. G. Narasimhan. Structured light 3D scanning in the presence of
global illumination. In Proceedings of the European Conference on Computer Vision (ECCV), pages 131–144, 2000.

[18] Micro-epsilon. Applications for 2D/3D laser scanners, 2021. URL https://www.micro-epsilon.com/2D_3D/laser-scanner/applications/ Site accessed April 13, 2021.

[19] G. Müller, J. Meseth, M. Sattler, R. Sarlette, and R. Klein. Acquisition, synthesis, and rendering of bidirectional texture
functions. In Computer Graphics Forum, volume 24, pages 83–109. Wiley Online Library, 2005.

[20] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A retargetable forward and inverse renderer. ACM Transactions on Graphics (TOG), 38(6):1–17, 2019.

[21] D. Palousek, M. Omasta, D. Koutny, J. Bednar, T. Koutecky, and F. Dokoupil. Effect of matte coating on 3D optical measurement accuracy. Optical Materials, 40:1–9, 2015.

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

[23] V. Rossi. 3D digitization with The Smithsonian Institution, 2020. URL https://www.faro.com/en/Resources/Language/Library/Article/3d-digitization-with-the-smithsonian-institution Site accessed April 13, 2021.

[24] T. Schops, V. Larsson, M. Pollefeys, and T. Sattler. Why having 10,000 parameters in your camera model is better than twelve. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2535–2544, 2020.

[25] L. Shi, B. Li, M. Hašan, K. Sunkavalli, T. Boubekeur, R. Mech, and W. Matusik. MAtch: differentiable material graphs for procedural material capture. ACM Transactions on Graphics (TOG), 39(6):1–15, 2020.

[26] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In Proceedings of the IEEE International Conference on Computer Vision, pages 2686–2694, 2015.

[27] Scattering Measurements. Synopsys, Inc., 2020. URL https://www.lighttec.fr/scattering-measurements/ Accessed on April 13, 2021.

[28] G. Taubin, D. Moreno, and D. Lanman. 3D scanning for personal 3D printing: build your own desktop 3D scanner. In SIGGRAPH ’14, 2014.
[29] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 23–30. IEEE, 2017.

[30] J. Wang, F. Shi, J. Zhang, and Y. Liu. A new calibration model of camera lens distortion. Pattern recognition, 41(2): 607–615, 2008.

[31] Y. Wang and H.-Y. Feng. Modeling outlier formation in scanning reflective surfaces using a laser stripe scanner. Measurement, 57:108–121, 2014.

[32] Z. Zhang. A flexible new technique for camera calibration. IEEE Transactions on pattern analysis and machine intelligence, 22(11):1330–1334, 2000.