# A simulation framework for the design and evaluation of computational cameras

Thomas Nürnberg[1,2], Maximilian Schambach[2], David Uhlig[2], Michael Heizmann[2], and Fernando Puente León[2]

[1]Robert Bosch GmbH, Corporate Research, Robert-Bosch-Campus 1, 71272 Renningen, Germany
[2]Karlsruhe Institute of Technology, Institute of Industrial Information Technology, Hertzstr. 16, 76187 Karlsruhe, Germany

## ABSTRACT

In the emerging field of computational imaging, rapid prototyping of new camera concepts becomes increasingly difficult since the signal processing is intertwined with the physical design of a camera. As novel computational cameras capture information other than the traditional two-dimensional information, ground truth data, which can be used to thoroughly benchmark a new system design, is also hard to acquire. We propose to bridge this gap by using simulation. In this article, we present a raytracing framework tailored for the design and evaluation of computational imaging systems. We show that, depending on the application, the image formation on a sensor and phenomena like image noise have to be simulated accurately in order to achieve meaningful results while other aspects, such as photorealistic scene modeling, can be omitted. Therefore, we focus on accurately simulating the mandatory components of computational cameras, namely apertures, lenses, spectral filters and sensors. Besides the simulation of the imaging process, the framework is capable of generating various ground truth data which can be used to evaluate and optimize the performance of a particular imaging system. Due to its modularity, it is easy to further extend the framework to the needs of other fields of application. We make the source code of our simulation framework publicly available and encourage other researchers to use it to design and evaluate their own camera designs.[1]

**Keywords:** Computational cameras, raytracing, simulation, evaluation

## 1. INTRODUCTION

Since conventional imaging sensors are intrinsically two-dimensional, in order to obtain high-dimensional data from the light of a scene, such as depth or spectral information, optical coding of the light before image acquisition is necessary. Therefore, the newly emerging *computational cameras* offer combined optical and digital signal processing techniques. This enables the design of novel optical imaging systems that allow the acquisition of different properties of a scene than from a traditional image. For example, hyperspectral snapshot imagers record a spatially resolved spectrum of a scene, whereas light field cameras capture the scene's structural information.

Since the design process of a computational camera has to consider both the physical and the digital domain, the development of new camera systems is more complex than that of traditional ones. Furthermore, due to the huge amount of possibilities to optically modulate a signal (e. g. by lenses, mirrors, apertures, spectral filters, etc.) there are numerous possible designs for new computational cameras for a given measurement task.[2] A simulation framework can support this process substantially and allows to further assess the system performance.[3]

In this article, we present a raytracing simulation framework tailored to the design and evaluation of cameras for computational imaging applications. Using easily accessible reference data from the simulation, the performance of the overall system can be assessed and optimized. Through the physically correct implementation of the contributing camera components, including a physical model of the camera sensor, it is ensured that the

Further author information:
1: thomas.nuernberg@de.bosch.com
2: {firstname.lastname}@kit.edu

system performance is comparable to a real prototype.[4] Furthermore, the simulation framework allows for the implementation of reference systems that do not posses a physical counterpart, for example by directly rendering intermediate simulation data like the image depth, surface normal or spectrum, or by simulating technically impossible aperture stops.

The proposed raytracer is written in C++ and we make the source code publicly available.[1] Due to its object oriented implementation, the raytracer is easily extendable in all of its components. Hence, other researchers can implement (and contribute) their own camera designs and use the simulated images to evaluate their system. Furthermore, by exchanging the abstract scene description, researchers can benchmark their newly implemented computational camera in comparison to existing traditional and computational imaging methods.

The remainder of this paper is organized as follows: We will first introduce related works and the principles of raytracing in Sections 2 and 3 respectively. We will then give a thorough overview of the proposed simulation framework, including a detailed presentation of the camera and sensor implementations as well as an overview of some component and reference implementations, in Section 4. We will summarize and give a brief conclusion in Section 5.

## 2. RELATED WORK

A simulation makes it possible to perform a multitude of experiments with a system without the need of an actual prototype and with parameterizations beyond the intended design or technically feasible implementations. As all system parameters are known exactly, a simulation can help to thoroughly assess a system, especially when ground truth data is hard to obtain. Both of these aspects apply to the field of image processing and computational imaging.

Early works on camera simulation focus on effectively generating synthetic images that can support users in their tasks, e. g. in the creation of an action plan for an industrial robot[5] or the design of a vision system.[6] Here, the process of projection and rasterization was used to generate images that can be immediately interpreted by a human user. More recent works use simulated images of a conventional camera as input for different image processing algorithms. Butler et al. use the publicly available raw data from an animated short film for testing and evaluating algorithms for optical flow estimation.[7] By varying the simulation complexity, the authors can analyze the influence of different phenomena of geometrical optics like depth of field on the image processing algorithms. Irgenfried et al. use raytracing to compare the results of different image segmentation algorithms using both simulated and real images.[8] The authors outline the importance of using a physically correct simulation method in order to get a meaningful comparison of the results generated from synthetic and real images. However, they use an empiric noise model to increase the variance of the simulated images. Retzlaff et al. also use raytracing to generate data for the training of a classifier of colored glass shards for an industrial sorting system.[9] They achieve similar classification rates on real glass shards when using a classifier trained with synthetic data compared to using a classifier trained with real data. In his dissertation, Meister analyzes the meaningfulness of results of image processing systems when using synthetic images as input.[10] One of his results is that it is of high importance to accurately simulate the physical phenomena that cause the features in the image, on which the used image processing algorithms highly depend on. For example, when evaluating an algorithm for optical flow estimation, phenomena like motion blur have to be accurately simulated. On the other hand, phenomena that do not affect these features but increase the overall realism of the synthetic image can be omitted.

With the emergence of computational imaging systems, the focus of image simulation also shifts towards testing and optimization of a camera design. Brady and Gehm test their concept of a hyperspectral camera using compressive sensing techniques by simulating the camera system as idealized convolution of a hyperspectral input image with a spectrally coded aperture.[11] In order to simulate medical imaging techniques based on X-rays, Tabary et al. use raytracing amongst other simulation methods.[12] Birch et al. simulate a lensless imaging system with a random refracting element using a raytracing framework.[13] The random refracting element acts as analog compressive sensing method. Thus, a human interpretable image can be reconstructed after image acquisition.[14] By using a simulation, iterations to the camera concept can be easily tested and thus, the effort during the system design can be reduced significantly as there is no need to recalibrate a real prototype.

## 3. PRINCIPLES OF RAYTRACING

In the scope of geometrical optics, light is described as rays whose optical path lengths, according to Fermat's principle, attain an extremum. Since in conventional cameras, the dimensions of the optical elements are usually much larger than the wavelength of the incident light, geometrical optics is well suited to describe the imaging process of cameras, but as such is lacking the description of many essential properties of light. By means of an extension to geometrical optics (see Figure 1), properties such as color and intensity, which can only be described within wave optics (or higher order theories such as quantum electrodynamics), can be heuristically incorporated into geometrical optics: The light field or *plenoptic function* $L_{\lambda,t}(x, y, z, \phi, \theta)$ describes the optical radiance at point $(x, y, z)$ in direction $(\phi, \theta)$ of wavelength $\lambda$ at time $t$ in units $\mathrm{W\,m^{-2}\,sr^{-1}\,nm^{-1}}$. In homogeneous media that are free of occluders, the radiance along a ray is constant. The spatial dependency of the light field can hence be reduced by one dimension, resulting in the so-called *4D light field* $L_{\lambda,t}(u, v, a, b)$, where the coordinates $(u, v, a, b)$ correspond to a certain parametrization of the spatial dependency of the light field of which there are numerous.[2]

In the context of raytracing, the light field is used to simulate the image formation process of cameras.[15] Within the simulation, the scene (and the camera) are assumed to be static. Non-static cameras, for example cameras utilizing a digital micromirror device, or scenes with moving objects have to be synthesized in a serial fashion. Furthermore, due to the limitations of the underlying physical theory, simulation of phenomena associated with higher order theories, such as diffraction, interference or polarization, is not possible within a raytracing framework. However, just like color and intensity have been heuristically incorporated into geometrical optics, it is in fact possible to approach this in a similar fashion by assigning each ray a phase or polarization state[16,17,18,19] (this shall too not be part of the proposed framework). Therefore, the raytracing framework is most suited for raytracing that do not utilize these higher order effects, which in fact covers a vast portion of classical and computational cameras utilizing lenses, apertures, dispersive media etc.[2]

In this context, the physical process of image formation on the sensor can be described by

$$E_{\lambda,t}\left(\mathbf{p}\right) = \iint L_{\lambda,t}(x, y, z, \phi, \theta) \cos^4\left(\theta\right)\ \mathrm{d}\Omega_{\frac{1}{2}}\,, \tag{1}$$

where the hemisphere above a point $\mathbf{p} = (x, y, z)$ on a surface is denoted by $\Omega_{\frac{1}{2}}$. The angle between the incident ray and the surface normal of the sensor is denoted by $\theta$. The irradiance $E_{\lambda,t}\left(\mathbf{p}\right)$ is measured in units of

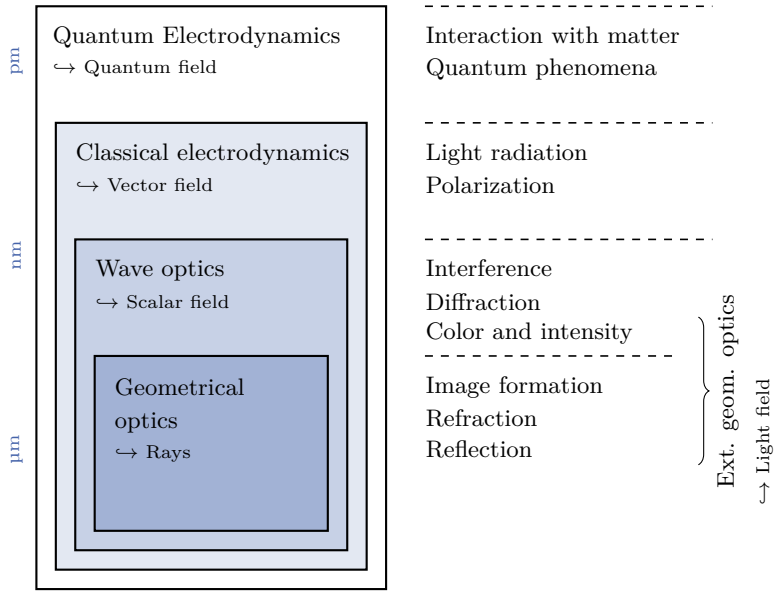



Figure 1: Overview of the physical theories of light and their associated phenomena.

$\mathrm{W\,m^{-2}\,nm^{-1}}$. Integrating the irradiance over the exposure time $T$ yields the spectral exposure

$$H_\lambda(\mathbf{p}) = \int_T E_{\lambda,t}(\mathbf{p})\ \mathrm{d}t\ . \tag{2}$$

The integration over the spectral responsivity $R(\lambda)$ of the sensor results in the radiant exposure

$$H(\mathbf{p}) = \int R(\lambda)\, H_\lambda(\mathbf{p})\ \mathrm{d}\lambda \tag{3}$$

of pixel $\mathbf{p}$ in units of $\mathrm{J\,m^{-2}}$. Finally, an integration over the active surface $A_\mathbf{p}$ of a pixel $\mathbf{p}$ yields the radiant energy

$$Q_\mathbf{p} = \iint H(\mathbf{p}')\ \mathrm{d}A_\mathbf{p} \tag{4}$$

which is digitized by the sensor.

The main idea of raytracing is to numerically approximate the physical process of image formation by representing the continuous light field by a limited amount of discrete samples, the rays. This approach corresponds to a Monte Carlo integration of eqs. (1)–(3). Hence, it can be proven that the approximation of $H$ converges towards the true value with increasing number of samples $N$.[20] Additionally, by assuming a static scene and camera, eq. (2) further reduces to a multiplication with the exposure time $T$.

The representation of the continuous light field by discrete rays allows to effectively trace single rays inside the simulated camera and the simulated world, hence the name raytracing. Furthermore, by tracing the rays in the reverse direction of their actual propagation, that is beginning at the sensor and ending at light sources, only the fraction of the light field that actually contributes to the image formation is considered. The process of tracing the rays in the scene is described by the famous rendering equation, which is solved in a similar fashion as eqs. (1)–(3). As the main focus of this article is the camera simulation, we refer to the literature for more information about accurate scene simulation.[15,21]

## 4. SIMULATION FRAMEWORK

We propose an object oriented raytracing framework written entirely in C++ as shown in Figure 2. The main layout of the framework follows well known raytracing implementations.[15,21] In particular, the framework consists of the following:

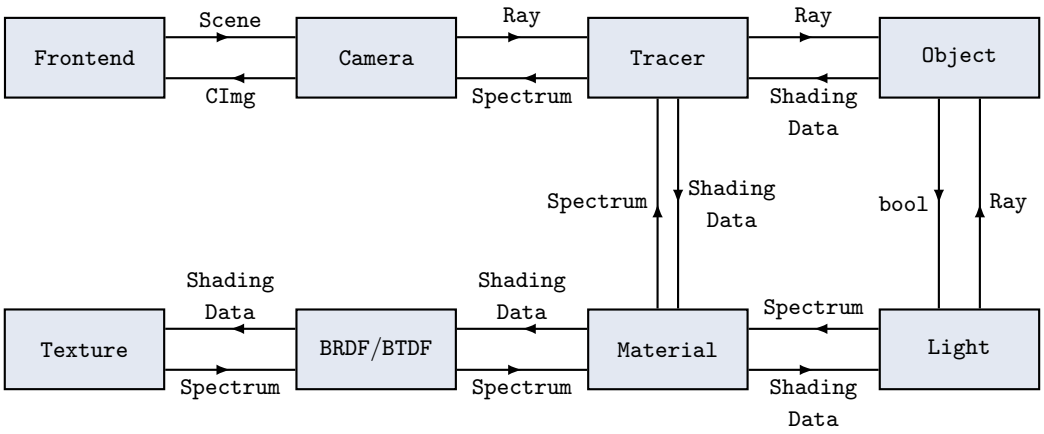



Figure 2: Simplified overview of the proposed simulation framework.

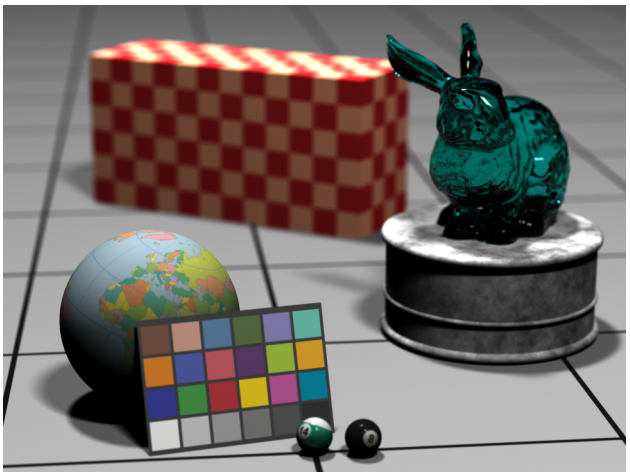


Figure 3: Example scene which is used throughout the paper.

The `Frontend` provides an abstract interface for the user to control the raytracing routine. The user provides an abstract scene description in form of a JSON file that is read and parsed by the `Frontend`. Furthermore, the user can control additional settings of the rendering process such as multithreading parameters or the output file name and image format. We provide a `GuiFrontend` with a graphical user interface as well as a command line based `BasicFrontend` (which does not depend on any external libraries) to control the raytracer.

The `Camera` initiates the sampling and tracing routine and is responsible for all image side ray calculations. Furthermore, the `Camera` provides an object side ray (in world coordinates) to be used for the actual rendering process. Finally, the `Camera` receives back a spectrum description, which can be monochromatic, RGB or true sampled spectra, of the traced ray which is saved to the (monochromatic, RGB or multispectral) image and passed back to the `Frontend`. Since our framework puts emphasis on the camera's capabilities, it will be discussed in more detail in Section 4.1.

The `Tracer` is responsible for all object side ray tracing and interactions of the rays with the scene. It receives the object side ray from the `Camera` and calculates possible intersections of the ray with the scene objects and initiates the shading process. Note, that the tracing is held abstract, i. e. the `Tracer` is not only capable of tracing the spectral properties of the objects (i. e. their color) but also abstract reference properties such as depth or surface normals (see also Section 4.2).

The remaining blocks are responsible for the object shading. That is, converting the abstract shading data of an `Object` into a spectrum, depending on the `Object`'s material properties such as color, texture and reflection, scattering and transmission properties (defined through the bidirectional reflectance/transmittance distribution function, BRDF/BTDF) which are provided by the `Material`, `BRDF/BTDF` and `Texture` blocks in combination with the spectral properties of the `Light` source(s). This ultimately results in approximating the rendering equation for the given scene. Many raytracing frameworks emphasize on the shading part of raytracing to render photo realistic images of complex textures such as hair, fur, skin, trees or grass (for example for animated cinematic movies or computer games). As our main focus lies on the implementation of computational cameras, we will not discuss the shading process in more detail and instead refer the interested reader to the literature.[15,21]

Due to the object oriented layout of the framework, all blocks depicted in Figure 2 can be extended to one's own needs. For example by adding a new `Camera` (which is the most likely case) but also by adding a new reference `Tracer`, a new `Texture` etc.

A simulated image of an example scene is shown in Figure 3 which we will use in the remainder of this paper to illustrate various aspects of our framework's capabilities.

## 4.1 Camera

The camera block (see Figure 2) is the most crucial block for the investigation and simulation of computational cameras. The different camera implementations share a set of common features which are implemented using an abstract `Camera` base class that every particular camera, such as a `ThinLens` or `Pinhole` camera (see Section 4.1.2), is derived from. The very core of every camera is a `Sensor` instance (see Section 4.1.1) and one or multiple samplers defining the sampling scheme of the rays inside the camera. For example, one samples the starting points of rays on pixels and another samples the ray's directions, e. g. by sampling the intersection points with a lens. The other core functionality of each camera instance is a `Transformation` object which holds the camera's pose in world coordinates and takes care of the correct transformation between camera and world coordinates when passing a ray to the `Tracer` instance.

Other than that, every camera is responsible for correctly sampling and tracing the rays as they pass the different camera components. This process highly depends on the internal structure of the simulated camera. Therefore, it has to be implemented individually when adding a new camera to the framework. To make the implementation easier, we provide a set of core building blocks and reference implementations such as apertures, (real and thin) lenses etc., which we describe in more detail in Section 4.1.2.

### 4.1.1 Sensor simulation

Being part of the `Camera`, the `Sensor` instance holds all crucial parameters associated with the (physical) sensor of the camera. In particular, it holds the sensor size (in px) as well as the pixel pitch, to be able to convert back to the camera coordinate system. Furthermore, the sensor provides the sampler responsible for sampling the pixel area in a suitable fashion. Finally, to be able to simulate different sensor types, the sensor holds a spectral responsivity curve for which we provide numerous reference implementations of real existing sensors by Sony and ON Semiconductors (both CMOS and CCD). Apart from a physically correct monochromatic sensor implementation, we provide multiple reference sensors to be able to directly render RGB or multispectral images (see Section 4.2).

An image of a real camera always suffers from noise of different sources. Depending on the type and amount of noise, a specific computational or traditional imaging approach can become more favorable than another.[23,24] When using a simulation for early experiments of a new camera concept, noise may be omitted. But when a simulation is used to benchmark and optimize a camera design, noise must be considered.

The proposed framework implements the EMVA 1288 noise model[22] as shown in Figure 4. It models the noise variance of excited electrons $\sigma^2_{\mathrm{e}^-}$ on the sensor due to the inherent poisson noise characteristic of incident photons with mean $\mu_{\mathrm{photon}}$ and variance $\sigma^2_{\mathrm{photon}}$, where $\mu_{\mathrm{photon}}$ is the desired signal. In addition, there is measurable noise from the camera electronics even with no incident light. It consists of the read noise, which is assumed to be zero-mean and with variance $\sigma^2_{\mathrm{read}}$, and noise due to thermal excitation. Again, this is modeled by a poisson process, where the rate at which the electrons are excited is characterized by the dark current $\mu_I$. The combination of these noise terms is referred to as dark noise with mean $\mu_{\mathrm{dark}} = \mu_I T$ and variance $\sigma^2_{\mathrm{dark}} = \sigma^2_{\mathrm{read}} + \mu_I T$. After the amplification with the system gain $K$, the A/D conversion results in the addition of quantization noise $\sigma^2_{\mathrm{q}}$. Depending on the illumination of the captured scene, the exposure time and the camera parameters, either of these noise sources can be dominant over the others, with varying impacts on the overall system performance.

In favor of a more intuitive user interface, which allows new users to immediately render simple but meaningful scenes, intensities are defined as digital intensities rather than radiometric intensities. For example, material

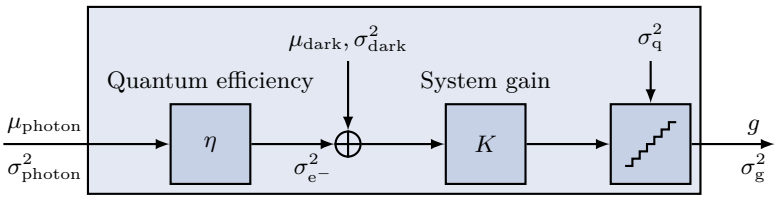
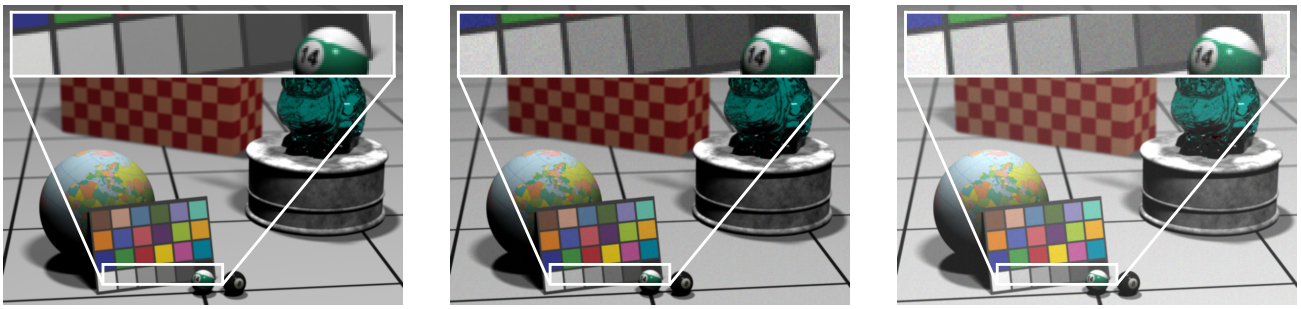



Figure 4: Noise model according to the EMVA 1288 standard.[22]



(a) No camera noise. (b) Exposure time $T = 1 \times 10^{-3}\,\mathrm{s}$. (c) Exposure time $T = 1\,\mathrm{s}$.

Figure 5: Simulated example scene with different camera noise at different exposure times.

colors can be defined directly as three RGB values, where the range of 0 to 255 directly corresponds to the representable dynamic range of common monitors. This also simplifies and speeds up the simulation process as all calculations can be performed in the domain of digital intensities. Radiometric quantities do not have to be considered. On the other hand, this implies that the physical domain, where the photon noise and the dark noise originate from, is not directly simulated.

The EMVA 1288 noise model provides the connection of these two domains. By specifying the camera gain $K$, the dark current $\mu_I$ and the read noise variance $\sigma^2_{\mathrm{read}}$, which are usually listed in the data sheet of a camera or a sensor, and by choosing an exposure time $T$, the radiant exposure that causes a specific digital intensity is implicitly defined by the model. Furthermore, the model also allows to translate the parameters of the photon and dark noise to the domain of digital intensities. That is, in the simulation, the noise can be applied directly to the final digital image. The value of the photon noise is randomly drawn from the normal distribution

$$n_{\mathrm{photon}} \sim \mathcal{N}\left(0, K^2 g\right) \,, \tag{5}$$

where $g$ is the gray scale intensity of the current pixel. Drawing from a normal distribution is justified as the Poisson distribution converges towards a normal distribution for large numbers of event occurrences. Similarly, the value of the dark noise is randomly drawn from

$$n_{\mathrm{dark}} \sim \mathcal{N}\left(K\mu_I T, K^2\left(\sigma^2_{\mathrm{read}} + \mu_I T\right)\right) \,, \tag{6}$$

where the Poisson distribution of thermally excited electrons is again represented sufficiently by a normal distribution. If, due to negative noise values, the resulting grey scale value is negative, clipping is applied. Finally, the quantization noise is not simulated separately as intensities are stored internally as floating point values. Therefore, the quantization noise is added implicitly by converting the final image to the appropriate bit resolution (usually between 8 and 16 bit).

The workflow in order to simulate an image with the same degree of noise compared to a real image is as follows:

1. Look up the system gain $K$, read noise variance $\sigma^2_{\mathrm{read}}$ in units of $\mathrm{e}^-$ and dark current $\mu_I$ in units of $\mathrm{e}^-/\mathrm{s}$ from the data sheet of a camera or sensor.
2. Capture a real image of the desired scene and note the exposure time $T$.
3. Simulate the image with the determined noise parameters and tune the intensities of the light sources and the material colors until the resulting image intensities match those of the captured real image.

It is worth mentioning that this workflow is only valid when the real camera possesses a linear response curve. Otherwise, the captured image has to be linearized first.

Figure 5 shows simulated images without simulated camera noise and with camera noise at different exposure times $T$. In the proposed simulation framework, an increasing exposure time effectively decreases the radiant

exposure of the sensor. Therefore, the noise characteristic shifts from mainly multiplicative photon noise in Figure 5(b) to mainly additive dark noise in Figure 5(c).

In addition to the desired image noise from the physical and technical sources mentioned above, undesired noise or artifacts caused by insufficient approximation of eqs. (1)–(3) may degrade simulation results. The ultimate cause for these errors lies in the violation of the well-know Nyquist-Shannon sampling theorem when approximating a wide-band light field by a limited number of samples. The general strategy to minimize the effect of these approximation errors is to increase the number of discrete samples used for the simulation until the resulting energy of the undesired errors is negligible compared to the physically motivated image noise. Furthermore, sampling patterns that incorporate randomness should be preferred over deterministic patterns. The former cause the approximation errors to become apparent as additional noise in the whole image, which is more controllable than locally concentrated Moiré-like aliasing artifacts caused by regular sampling. As the overall bandwidth of the light field depends on the scene as well as the camera configuration, the appropriate amount of samples can only be decided on a case-by-case basis. Using a (partially) homogeneous scene might help in order to estimate the appropriate amount of samples in a specific case.

#### 4.1.2 Component kit

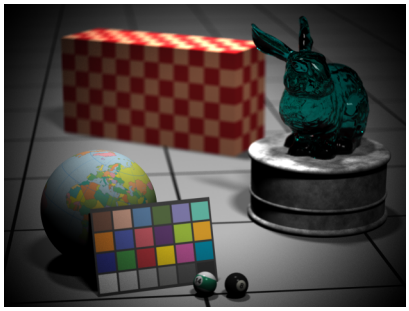
Apart from the sensor, a real camera consists of optical elements such as lenses, apertures, free space propagation, etc., which aim to project the rays from the environment onto the sensor surface. As mentioned before, the tracing is performed reversed starting from the sensor into the environment. The main task of the remaining camera module is, starting from each pixel, to determine the course of the rays until they exit the camera.

In order to simplify the design phase, we provide reference implementations of a few basic camera systems and various basic construction elements as a component kit which makes it easy to add new cameras to our framework. All components treat rays in the sense of (extended) geometric optics. Nevertheless, additional wavelength dependencies and dispersive elements can be simulated. In our framework, any optical component, which in general represents transitions between homogeneous media, can be simulated. In the following we give a short overview of these optical components.

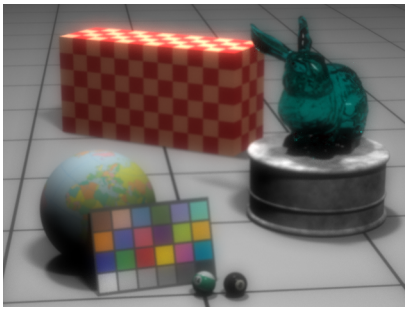
A major class of components are optical lenses, the most intuitive of which is the ideal thin lens. The refraction at the thin lens is calculated in paraxial approximation according to the ideal imaging law. A reference implementation which uses this component is the `ThinLens` camera. A more realistic implementation of an optical lens, using a single thick spherical lens without paraxial approximation, is exemplarily implemented in the `RealLensCamera`. Interactions of the rays with the lens are determined using the law of refraction. Thus, this camera shows geometric aberrations such as spherical aberration, coma and astigmatism. The strength of these effects can be adjusted by appropriate selection of the lens parameters. In Figure 6(b), the example scene is rendered using a `RealLensCamera`, showing blur effects introduced by geometric aberrations. As mentioned before, wavelength dependencies (dispersion) can be simulated. Therefore, an extended version of the thick lens has a wavelength dependent refraction index which can be found exemplary in the `ChromaticRealLensCamera` implementation. As a result, the lens shows chromatic aberrations in addition to the geometric aberrations of the `RealLensCamera`.

Apart from placing one single lens in front of the sensor, it is possible to create a grid and place a microlens on every grid point to simulate a microlens array. The `LightFieldCamera` is a more exotic example of a camera, which among other things makes use of this microlens array element to create a lenslet based light field camera similar to the proposed design of Ng[25] and the formerly commercially available camera from Lytro. Figure 6(c) shows an example of a lenslet based `LightFieldCamera`. For the sake of clarity, the radius of the microlenses is chosen to be very large.

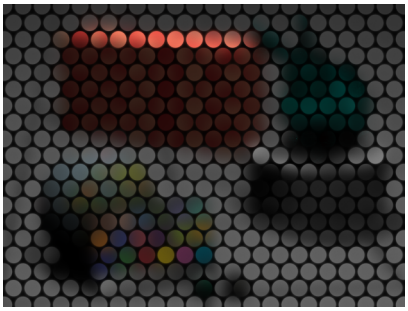
Another important class of components are apertures. In optics, they are devices that limit the diameter of ray bundles by intensity attenuation (or blocking). Their purpose is versatile, depending on the design and effect. Arguably the most prominent feature of an aperture is the limitation of the amount of light entering the lens. The smaller the aperture, the less light reaches the sensor. At the same time, the size of the opening determines the depth of field. Smaller apertures with larger f-number create a deep depth of field which allows objects at different distances to be simultaneously in focus. Another important application is the limitation of optical aberrations which tend to become more severe at the edge of the lens. Although very useful in some cases, apertures introduce



(a) Camera of type `ThinLens` with a `CircularAperture`.



(b) Camera of type `RealLensCamera`.



(c) A lenslet based `LightFieldCamera` with exaggerated lenslet size.

Figure 6: Simulated example scene using different combination of basic camera components.

new unintended effects: Vignetting occurs when rays are partially blocked by the aperture. This changes the shape of the entrance pupil as a function of the ray's incident angle, resulting in a gradual darkening towards the edges of the sensor. The smaller the aperture, the more visible becomes this effect. Figure 6(a) exemplarily shows the combination of an aperture with a `ThinLens` camera resulting in (mechanical) vignetting.

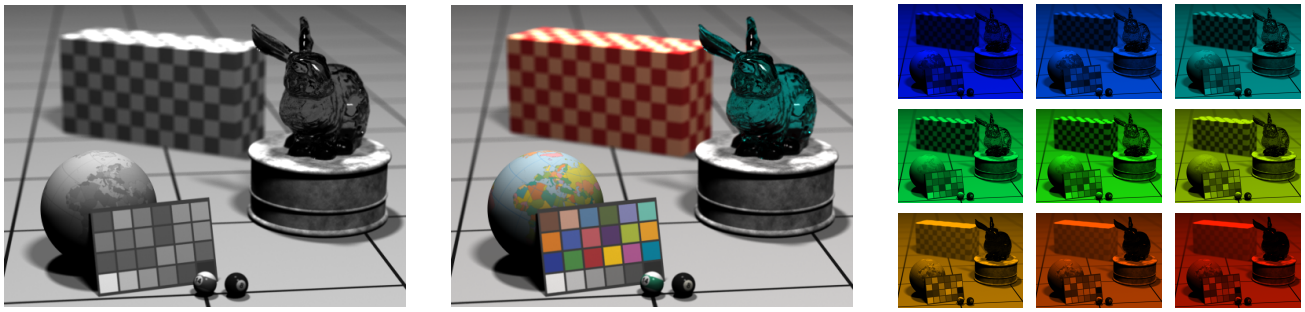
In our framework, any number of apertures can be placed at any position in the optical path, both before and behind the main lens. The classical aperture is circular and is implemented as the `CircularAperture` which is parameterized via its radius. Apart from such a classical aperture, more complex structures may be simulated. In general, the aperture can be defined by a binary or grayscale mask, using the `ImageAperture` implementation. This allows to simulate coded apertures which in real applications can be used to extract depth information[3,4] or to improve the depth of field to name two examples. The principle of the `ImageAperture` can be easily applied to arbitrary spectral components, allowing spectral filters to be modeled. In this way it becomes possible to evaluate hyperspectral camera systems. A simple example is to spatially code the sensor (as is the case in conventional digital color cameras) or to assign a spectral filter to individual lenses in a microlens array.

In summary, using the presented component kit, it can be said that the framework is easily extendable. New camera systems can be built from individual components and even new components can be added in a straightforward fashion.

### 4.2 Reference systems

Due to its design, the proposed framework allows for easy implementation of reference systems, i. e. virtual cameras that do not posses a real counterpart (usually due to physical limitations). In the context of computational cameras, this can be divided in three categories: Reference sensors, cameras and tracers.

Even though the sensor is in fact part of the camera block of the raytracer, it is worth discussing its reference implementations separately. A real sensor (such as a CMOS or CCD sensor) measures the number of photons that are collected at one pixel over the time interval given by the exposure time. As such, these sensors are intrinsically monochromatic (there have been approaches to true RGB sensors by Foveon but they can be considered exotic). In the raytracing process, every ray carries full spectral information. It is therefore straightforward to implement virtual sensors that measure the full (or RGB) spectrum at every pixel. In fact, measuring the RGB values at every sensor pixel is the default setting in most raytracers as it makes implementation of a Bayer pattern and according demosaicing unnecessary. Within the proposed framework, there are three types of (virtual) sensors: `Raw`, `Rgb` and `MultiSpectral`. The `Raw` sensor, in the above sense, is the only (physically) real sensor. It produces a monochromatic raw sensor image, either for an ideal sensor with constant quantum efficiency or by applying one of the predefined quantum efficiencies (see Section 4.1.1). Similarly, an `Rgb` sensor will save the full RGB value at every pixel, either with ideal or real quantum efficiency. This virtual full RGB sensor can for example be used to quantitatively evaluate different color coding and demosaicing schemes for digital color cameras. Finally, using the `MultiSpectral` sensor, we are able to render full multispectral images with an arbitrary number of



(a) Sensor of type `Raw`.

(b) Sensor of type `Rgb`.

(c) Sensor of type `MultiSpectral` in case of 9 color channels (colored).

Figure 7: Simulated example scene using the three available (virtual) sensor types (with constant quantum efficiency).

color channels defined by the user at compilation. This sensor can be used to benchmark and evaluate multi- and hyperspectral imaging systems. A comparison of the different (virtual) sensor types is shown in Figure 7.

A reference camera implementation makes it possible to compare camera systems to their idealized perfect systems, sometimes even without the need of any actual optical components. Three of such (virtual) camera systems will are presented here: `Pinhole`, `Orthographic` and `LightFieldReference` cameras whose image simulations are shown in Figure 8.

The probably best known reference example is the lens-free pinhole camera. A realistic (approximate) implementation of this camera would mean to place only a single aperture with a very small opening in front of the sensor. However, this configuration would cause the majority of the sampled rays to be directly blocked by the aperture and thus, although being calculated, they would have no influence on the creation of the image. The `Pinhole` camera reference implementation by-passes this problem by calculating the direction of the rays in such a way that they always pass exactly through the pinhole point. As a result, images with infinite depth of field can be generated.

A direct definition of the ray direction allows to sample rays that are exactly orthogonal to the sensor. This way one obtains an orthogonal projection of the scene onto the sensor which in real applications can only be achieved with complicated setups of lenses and apertures (e. g. telecentric optics). The `Orthographic` camera implementation has no perspective distortion, meaning that there is no vanishing point. One characteristic of this configuration is that the magnification does not change with axial object displacement and the objects therefore always appear the same size regardless of the object distance.

Finally, the `LightFieldReference` provides an implementation of a reference light field camera that directly captures the light field inside a (thin lens) camera and saves it in a 2D representation as a collection of subaperture views. To be able to compare it with light fields captured by the microlens array based camera (see Section 4.1.2), the parameters to initialize the camera are closely related to those of the `LightFieldCamera`. As shown in Figure 8(c), the light field shows subaperture views of the scene from different angles comparable to a multi camera array with (exaggerated) baseline. In combination with the `MultiSpectral` reference sensor, we are able to sample a full hyperspectral light field $L_\lambda(u, v, a, b)$ which is, to our knowledge, a unique feature of the proposed framework.

The tracer is not directly part of the camera block. It manages the interaction of the simulated environment and the object side rays generated by the camera. Conventional tracers calculate the intersection of the ray with the scene and return the spectral information of the hit point to the camera. More complex tracers are able to evaluate interactions with reflective materials by further tracing the reflected ray(s). And even more sophisticated tracers are capable of using global illumination to render photorealistic scenes as it was used to render Figure 3. Apart from the spectral information of the scene, tracers can be used to encode other information into the image. In our framework, there are multiple abstract tracers that return non-spectral information: The `Depth` tracer returns the distance of the imaged object to the first intersection of the ray with the scene as gray value to the sensor pixels. This is helpful when ground truth data is needed in order to evaluate depth estimation

algorithms. The `Normal` tracer returns the normal of the hit surface point as RGB values to the sensor pixels. Again, this can efficiently be used to evaluate algorithms that operate on surface normals, e. g. for camera based 3D-reconstruction. A comparison of the different abstract tracers is shown in Figure 9.

## 5. CONCLUSION

We have proposed a new raytracing simulation framework for the design and evaluation of computational cameras. We have discussed those parts that are most important to consider when developing and benchmarking new computational cameras (namely the camera, sensor and tracer implementations) in detail and provided many practical simulation examples. Using the provided reference implementations and building blocks, the implementation process for new designs is supported as much as possible. With the proposed framework, not only new camera designs can be implemented, but also reference data can be generated to benchmark and optimize the new designs, for example using multispectral, hyperspectral, lightfield, depth or normal reference simulations.

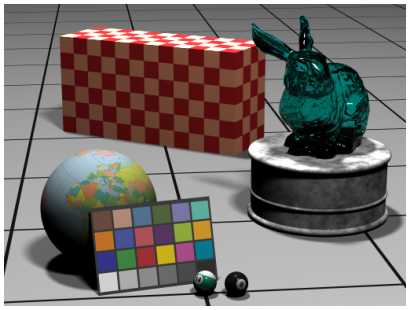
The proposed framework is modular in its design and we encourage the community to extend it with new designs using the publicly available source code.[1]

## REFERENCES

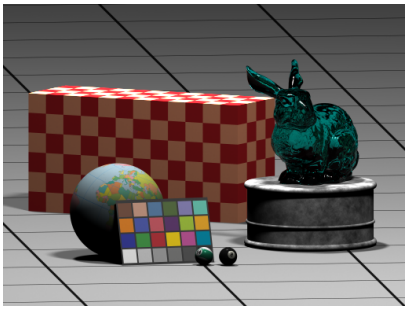

[1] Institute of Industrial Information Technology, Karlsruhe Institute of Technology, "Public GitLab repositories." https://gitlab.com/iiit-public (2019).

[2] Zhou, C. and Nayar, S. K., "Computational cameras: Convergence of optics and processing," *IEEE Transactions on Image Processing* **20**(12), 3322–3340 (2011).

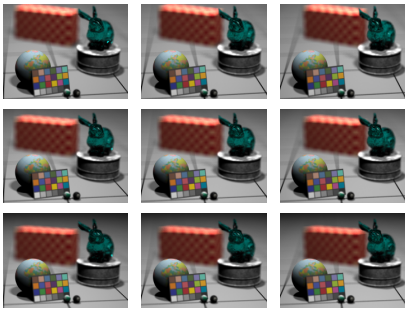
[3] Nürnberg, T., Zimmermann, C., and Puente León, F., "Simulationsgestützte Optimierung einer Computational-Kamera zur dichten Tiefenschätzung," *tm – Technisches Messen* **83**(9), 511–520 (2016).

(a) Camera of type `Pinhole`.

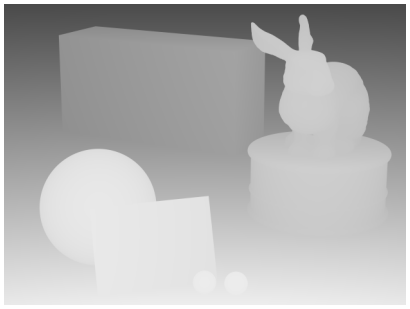


(b) Camera of type `Orthographic`.

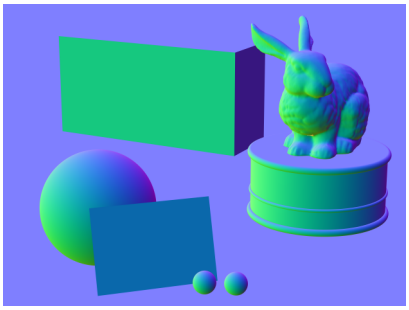


(c) A 3×3 `LightFieldReference` camera with exaggerated baseline.

Figure 8: Simulated example scene using different reference cameras.




(a) Tracer of type `Depth`.




(b) Tracer of type `Normal`.

Figure 9: Simulated example scene using different (virtual) tracer types.

[4] Nürnberg, T., Weinreuter, H., and Puente León, F., "Tiefenmessung mithilfe einer Kamera mit programmierbarer Apertur," *tm – Technisches Messen* **84**(S1), S52–S59 (2017).

[5] Raczkowsky, J. and Mittenbuehler, K. H., "Simulation of cameras in robot applications," *IEEE Computer Graphics and Applications* **9**(1), 16–25 (1989).

[6] Ikeuchi, K. and Robert, J.-C., "Modeling sensor detectability with the vantage geometric/sensor modeler," *IEEE Transactions on Robotics and Automation* **7**(6), 771–784 (1991).

[7] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J., "A naturalistic open source movie for optical flow evaluation," in [*European Conference on Computer Vision (ECCV)*], 611–625, Springer Berlin Heidelberg (2012).

[8] Irgenfried, S., Dittrich, F., and Wörn, H., "Realization and evaluation of image processing tasks based on synthetic sensor data: 2 use cases," in [*Forum Bildverarbeitung*], 35–46 (2014).

[9] Retzlaff, M.-G., Richter, M., Längle, T., Beyerer, J., and Dachsbacher, C., "Combining synthetic image acquisition and machine learning: accelerated design and deployment of sorting systems," in [*Forum Bildverarbeitung*], Puente, F. and Heizmann, M., eds., 49–61, KIT Scientific Publishing (2016).

[10] Meister, S. N. R., *On Creating Reference Data for Performance Analysis in Image Processing*, dissertation, IWR, Fakultät für Physik und Astronomie, Universität Heidelberg (2014).

[11] Brady, D. J. and Gehm, M. E., "Compressive imaging spectrometers using coded apertures," in [*Defense and Security Symposium*], **6264**, 62460A–1–62460A–9, SPIE (2006).

[12] Tabary, J., Marache-Francisco, S., Valette, S., Segars, W. P., and Carole, L., "Realistic X-ray CT simulation of the XCAT phantom with SINDBAD," in [*IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*], 3980–3983 (2009).

[13] Birch, G. C., LaCasse, C. F., Stubbs, J. J., Dagel, A. L., and Bradley, J., "Tolerance analysis through computational imaging simulations," in [*Optical Design and Fabrication (Freeform, IODC, OFT)*], *OSA Technical Digest (online)*, Optical Society of America (2017).

[14] Liutkus, A., Martina, D., Popoff, S., Chardon, G., Katz, O., Lerosey, G., Gigan, S., Daudet, L., and Carron, I., "Imaging with nature: Compressive imaging using a multiply scattering medium," *Scientific Reports* **4**, 5552–1–5552–7 (2014).

[15] Pharr, M. and Humphreys, G., [*Physically Based Rendering: From Theory to Implementation*], Morgan Kaufmann Publishers Inc., 2 ed. (2010).

[16] Agu, E. and Hill, F. S., "A simple method for ray tracing diffraction," in [*International Conference on Computational Science and Its Applications (ICCSA 2003)*], Kumar, V., Gavrilova, M. L., Tan, C. J. K., and L'Ecuyer, P., eds., 336–345, Springer Berlin Heidelberg (2003).

[17] Oh, S. B., Kashyap, S., Garg, R., Chandran, S., and Raskar, R., "Rendering wave effects with augmented light field," *Computer Graphics Forum* **29**(2), 507–516 (2010).

[18] Wolff, L. B. and Kurlander, D. J., "Ray tracing with polarization parameters," *IEEE Computer Graphics and Applications* **10**(6), 44–55 (1990).

[19] Chipman, R. A., "Polarization ray tracing," in [*Recent Trends in Optical Systems Design and Computer Lens Design Workshop*], **766**, 61–69, International Society for Optics and Photonics (1987).

[20] Leobacher, G. and Pillichshammer, F., [*Introduction to Quasi-Monte Carlo Integration and Applications*], Springer International Publishing (2014).

[21] Suffern, K. G., [*Ray Tracing from the Ground Up*], A K Peters, Ltd., Wellesley, Massachussetts (2007).

[22] European Machine Vision Association, "EMVA standard 1288 – Standard for characterization of image sensors and cameras." www.emva.org (2010).

[23] Cossairt, O., Gupta, M., and Nayar, S. K., "When does computational imaging improve performance?," *IEEE Transactions on Image Processing* **22**(2), 447–458 (2013).

[24] Levin, A., "Analyzing depth from coded aperture sets," in [*Proceedings of the 11th European conference on Computer vision: Part I*], *ECCV'10*, 214–227, Springer-Verlag (2010).

[25] Ng, R., Levoy, M., Brédif, M., Duval, G., Horowitz, M., and Hanrahan, P., "Light field photography with a hand-held plenoptic camera," *Stanford University Computer Science Tech Report CSTR 2005-02* **2**, 1–11 (2005).