Physics-Based Sensor Models for Virtual Simulation of Connected and Autonomous Vehicles

SAFETY RESEARCH USING SIMULATION

UNIVERSITY TRANSPORTATION CENTER

Dan Negrut, PhD
Professor
Department of Mechanical Engineering
University of Wisconsin-Madison

Radu Serban, PhD
Senior Scientist
Department of Mechanical Engineering
University of Wisconsin-Madison

Physics-Based Sensor Models for Virtual Simulation of Connected and Autonomous Vehicles

Dan Negrut, PhD
Professor
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0003-1565-2784

Asher Elmquist
Graduate Research Assistant
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0002-0142-1865

Radu Serban, PhD
Senior Scientist
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0002-4219-905X

A Report on Research Sponsored by

SAFER-SIM University Transportation Center
Federal Grant No: 69A3551747131

May 2020

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Table of Contents

List of Figures

Abstract

This document contains the final project report for the SAFER-SIM project titled "Physics-Based Sensor Models for Virtual Simulation of Connected and Autonomous Vehicles." The report includes discussion of sensors models for simulation autonomous vehicles, and overviews the simulation framework developed in accordance with the project. The framework, called Chrono::Sensor is developed as a module alongside Project Chrono to augment the open-source multi-physics engine with the capability to simulation sensor data from within its virtual environment. Chrono::Sensor provides support for the modeling and simulation of camera, lidar, GPS, and IMU. It also provides a framework to implement custom sensors that can leverage existing sensor generation functionality. Chrono::Sensor generates data using ray-tracing algorithms that mimic the data acquisition process of cameras and lidars. In addition to data collection, the sensors support further data augmentation including the addition of sensor-specific noise. Results from each sensor implementation are included as part of the corresponding sensor discussion, with the report concluding with two demonstrations showing the use of Chrono::Sensor, in combination with Chrono, to simulate autonomous vehicles.

1. Introduction

Simulation promises to provide a safe and cost-effective means to train and evaluate perception and navigation algorithms for operating connected autonomous vehicles. In order for these simulation platforms to provide a comprehensive platform to conduct meaningful scenarios, the simulation tool must be capable of providing realistic vehicle dynamics, providing vehicle-to-vehicle and vehicle-to-everything communication, allowing for multiple interacting agents (vehicle, pedestrians, etc.), and providing realism sensor data from within the virtual environment. To that end, this project has focused on developing, implementing, and expanding a platform and models for physics-based simulation of sensors for training and evaluating autonomous vehicles.

In order to comprehensively test the effectiveness and safety of perception and navigation algorithms, it is important to provide, from within the virtual environment, synthetic data that is equivalent to data that would be produced by a real sensor. The research conducted under this project sought to further these efforts by implementing a simulation framework that builds upon a high-fidelity multi-physics simulation tool [3, 4, 5], implementing simulation methods that closely mimic the data acquisition process of real sensors along with distortion and noise models that can augment ground truth data in an effort to generate realistic sensor data with artifacts, noise, and distortion as would be seen on real sensors.

The importance of sensor simulation in the software-in-the-loop testing of a vehicle control stack is illustrated in Figure 1.1 where the information of the world, known by the autonomous vehicle, is given by sensors.

To contextualize the type and number of sensors, Table 1.1 lists the sensors used on a selection of autonomous vehicles as of 2018. While additional sensors are part of continued work, the focus of this project and report was on two interoceptive sensors (GPS and IMU) and two exteroceptive sensors (camera and lidar). Interoceptive sensors give internal state information about the vehicle such as position, velocity, acceleration, angular velocity, etc. Exteroceptive sensors provide information about a vehicle's surroundings and are heavily used in perception, obstacle avoidance, and path planning. Furthermore, these sensors provide the ability to detect pedestrians in safety-critical scenarios.

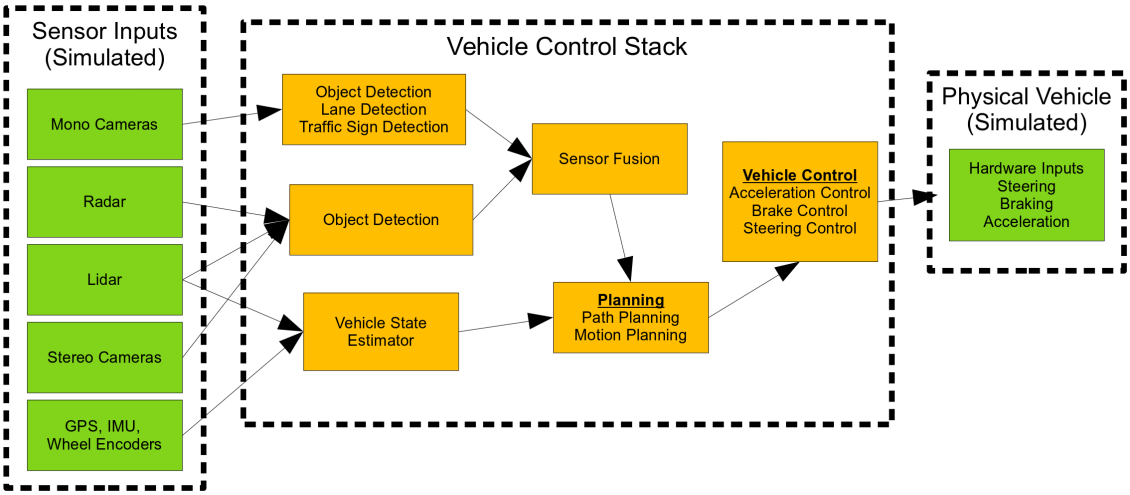


Figure 1.1: The flow of data into and out of an autonomous control program is shown here. The control stack is independent of whether the setup is virtual or physical, so sensing and dynamic response can be simulated to understand autonomous behavior in highly configurable environments (adapted from Girardin [1]).

Table 1.1: Sensor configuration from a select set of AV companies shows the reliance on vision-based sensing for autonomous behavior. Long-range (LR), medium-range (MR), and short-range (SR) lidar along with cameras and radars are used heavily for object detection (adapted from Girardin [1]).

Company	**Lidars**	**Radars**	**Cameras**	**IMUs**
Waymo	1-LR, 1-MR, 4-SR	4	8	1
Uber	1-LR	4	7	1
Toyota	4-MR, 4-SR	4	9	1
GM Cruise	5-SR	8	16	1
Renault-Nissan	4	5	8	1
Baidu	1-LR, 3-SR	4	2	1
Navya	3-LR, 7-SR	4	6	1

2. Chrono::Sensor

For autonomous vehicle simulation, sensor models must generate data that corresponds to the vehicle dynamics and scene in which that vehicle is being simulated. To this end, Chrono::Sensor was developed as a framework for implementing physics-based sensor models alongside simulation of vehicle dynamics and dynamic scene evolution. Through a tight coupling, Chrono::Sensor can leverage the internal dynamic state of an object from Chrono and use that, in part, to generate interoceptive data for GPS and IMU. For exteroceptive sensors, Chrono::Sensor uses the dynamic simulation in Chrono to drive the scene evolution to allow vehicles and other moving objects to be perceived as the vehicle interacts with the virtual environment.

The tight integration between Chrono and Chrono::Sensor is illustrated in Figure 2.1. Here, the virtual world from Chrono initializes the information for Chrono::Sensor. That data serves as ground truth information and is augmented to incorporate effects from lag, finite data collection time, noise, distortion, and other sensor characteristics.

For sensors that rely solely on internal state information (IMU and reduced GPS model), Chrono::Sensor stays in lock-step with the Chrono simulation. Since interoceptive sensors rely on data from many or most time steps but have little data augmentation overhead, this lock-step simulation prevents unnecessary overhead. For exteroceptive sensors such as camera and lidar, this is not the case. Since these sensors perceive the virtual scene, a different approach is implemented.

Excluding implementation details, two main factors impact the sensor models themselves. First, the sensors need highly detailed information about the scene, which aligns with the type of visual assets used in computer graphics. Second, a process that mimics the data acquisition of camera and lidar can improve the model accuracy and simplicity. For these two reasons, a ray-tracing approach is used for simulating camera and lidar. However, rather than implement a computer graphics pipeline based on ray-tracing, the methods implemented as part of this SAFER-SIM project assume the primary purpose is simulation, which allows the framework to be designed to focus on efficiency and data-realism rather than interaction and photo-realism.

For ray-tracing capabilities, Chrono::Sensor leverages the OptiX framework [6] which allows hardware accelerated ray tracing [7] without interaction or required visualization. This improves the scalability and model development. Additionally, since rendering is computationally time consuming and since

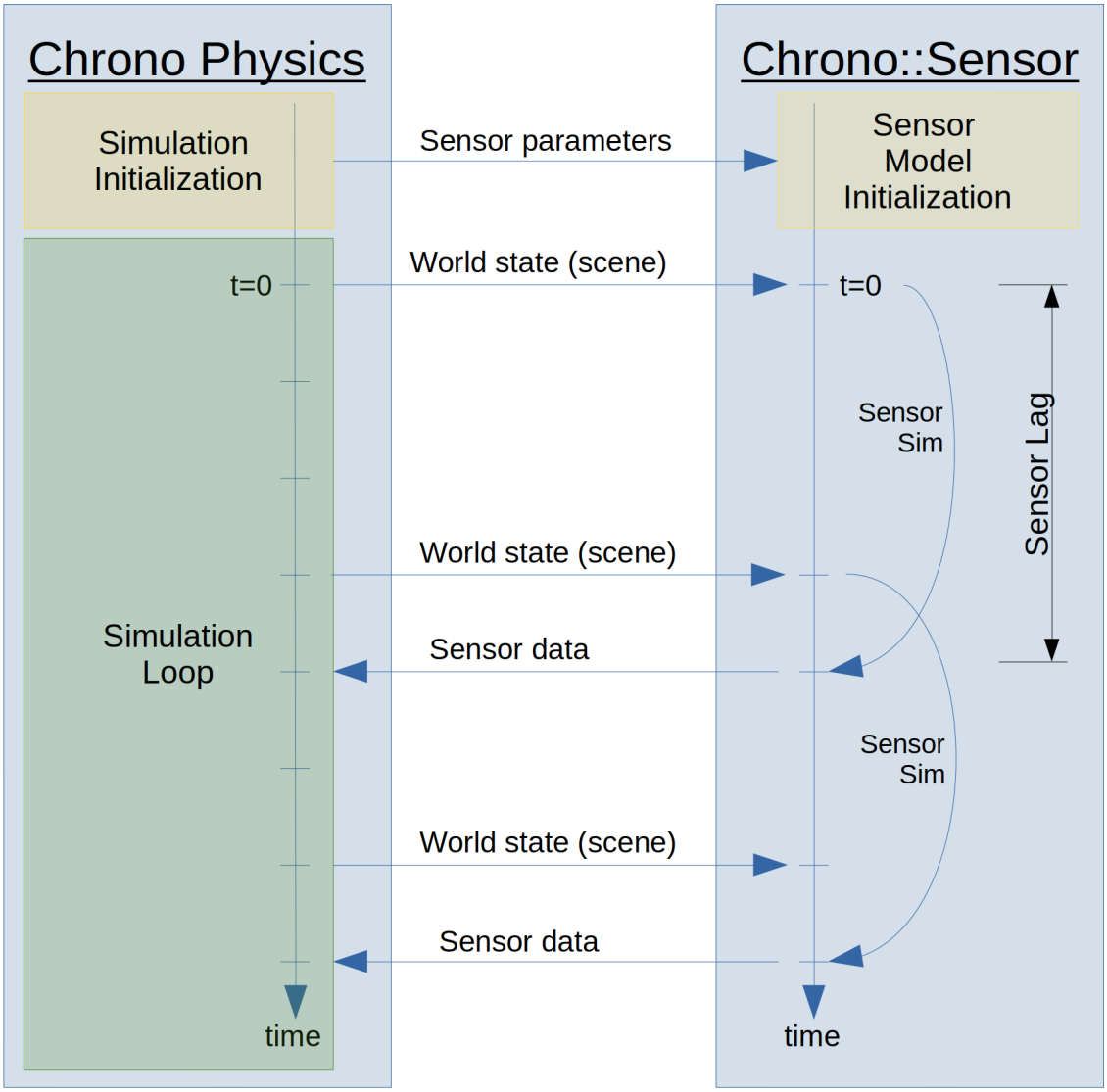


Figure 2.1: The connection between Chrono physics and Chrono::Sensor is tightly coupled, but is run at different time steps due to the small time step requirement of dynamics and the slow update frequencies of most sensors. Virtual world state information is sent from Chrono to Chrono::Sensor with data returned to the user at specific, and user-defined frequencies and delays.

these sensors operate at lower frequency than the dynamic simulation (5-60 Hz vs 300-1000 Hz), the rendering is done in parallel and synchronized when data should be returned to the user or when new data generation starts. This still follows the illustration in Figure 2.1 but is not performed in lockstep.

The framework for simulating camera and lidar is implemented in a generic manner to allow for extension and customization. Generically, these sensors are implemented using a filter graph with a

general sensor model shown in Figure 2.2. Each filter represents an operation on the data. In a generic sense, the filter graph includes data generation using any ray-tracing or physics-based model given. That generated data is subsequently processed by additional augmentation filters (i.e., noise addition), and finally a filter is applied to return the data to the user with a specified amount of lag.

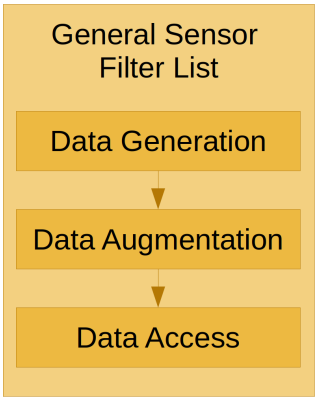


Figure 2.2: A general and typical example of filters that generate, augment, and return sensor data.

A concrete example is given in Figure 2.3, where a filter flowchart for a camera is illustrated. For this example model, data is generated in OptiX using ray tracing. That data is then processed to antialias the image, resize the image, add noise per pixel, apply gamma correction, introduce JPEG artifacts, and return the image to the user with lag. The filter-graph process mimics sensing pipelines as detailed later in the camera model discussion.

As an alternative approach, a deep neural network can be applied as one of the filters if the network has been trained to introduce realistic artifacts. The filter flowchart representing this approach is shown in Figure 2.4. The training and use of deep neural networks is a subject of further research into hybrid physics-based and machine learning approaches for simulating sensors. This fell outside the scope of this project.

In addition to model accuracy, solution efficiency is an important aspect when simulating autonomous vehicles. This is considered through the implementation of a separate render thread and library use. The camera and lidar models implemented in this framework can make efficient use of hardware resources by avoiding unnecessary bottlenecks. A typical camera or lidar filter implementation is processed entirely on the graphics processing unit (GPU) with synchronization only occurring when virtual world state or

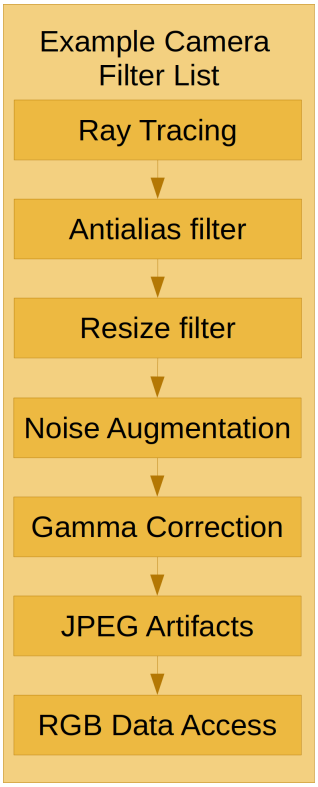


Figure 2.3: A concrete example of a possible setup for a camera sensor that generates an image using ray tracing, anti aliases the image before resizing, applies multiple augmentation filters for noise and distortion, and returns data to the user.

synthetic sensor data needs to be communicated. An example of this is illustrated in Figure 2.5 where each filter, including data generation, is performed and limited to the GPU.

Since autonomous vehicles often incorporates tens of sensors and simulations can involve multiple vehicles, the total number of sensors in a simulation can quickly balloon. In these cases, Chrono::Sensor can make use of additional hardware resources such as multiple GPUs when running on a remote server. The framework distributes the sensors among the GPUs to make efficient use of the resources available. Sensors are divided between GPUs based on update frequency in order to reduce the number of total scene updates as these points require synchronization between the physics simulation and the sensor data generation. A possible division of sensors between GPUs is shown in Figure 2.6.

Chrono::Sensor currently supports camera, lidar, GPS, and IMU sensors with the ability to extend and implement custom sensors using the existing framework. In addition to sensor-specific parameters,

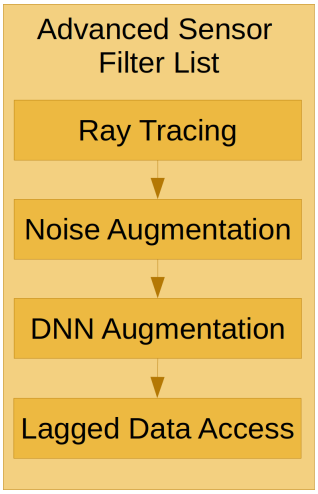


Figure 2.4: An advanced example that shows how Chrono::Sensor allows the use of a deep neural network as a sensor data augmentation filter.

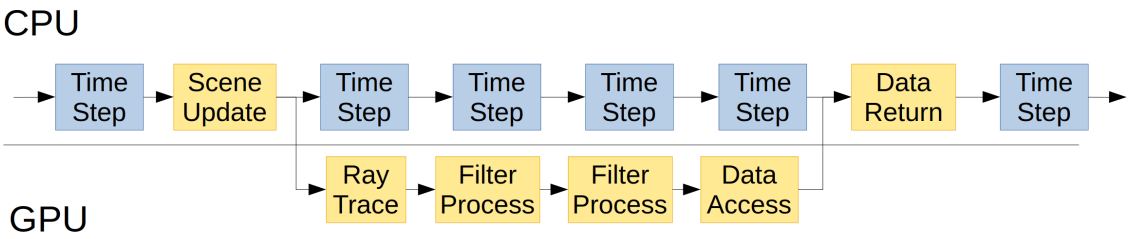


Figure 2.5: An example command process in Chrono/Chrono::Sensor shows the overlap of computation made possible by efficient use of memory transactions and slower update rates in Chrono::sensor.

each sensor can be parameterized by update frequency, lag, and data collection time. The sensors can then be attached to a body within the Chrono simulation for generating data. The sensor parameters along with the collection of filter graphs represent a model of a specific sensor.

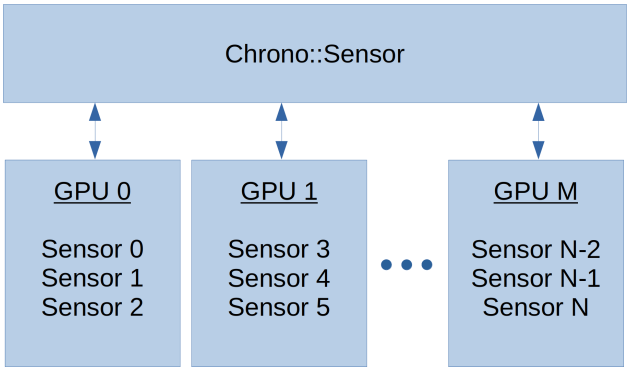


Figure 2.6: The use of multiple GPUs can allow Chrono::Sensor to simulate many sensors in the same environment by distributing sensors among different GPUs. Currently, the framework attempts to place sensors on GPUs by update frequency to lower the frequency of scene updates, which are the bottleneck of the sensor framework.

3. Camera Modeling and Simulation

3.1 Camera Model Background

In order to model a camera to produce realistic synthetic image streams from a virtual environment that are equivalent to real data produced by sensors on an autonomous agent, we must first understand the process by which physical camera sensors produce data. A diagram representing the steps is shown in Figure 3.1 and described in the literature [8, 9].

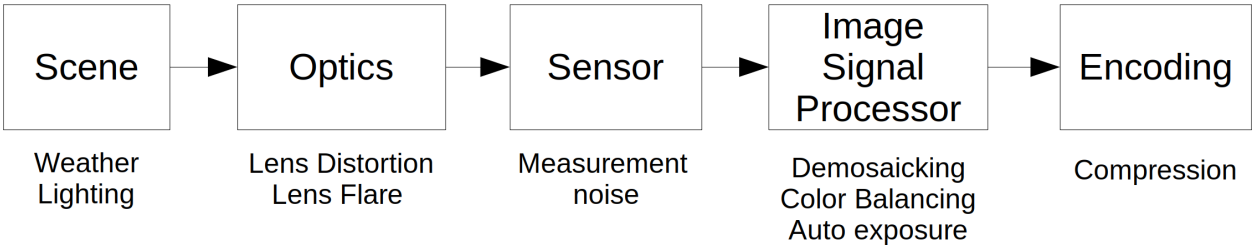


Figure 3.1: Image pipeline that shows data acquisition from scene through to image output for perception.

For a physical sensor, the scene is independent of the camera but can have significant impact on image processing due to varying atmospheric conditions and lighting. The light from the scene passes through the optical system to focus larger quantities of photons on the image sensor. Because of this focusing, light can be obtained from wider angles causing lens distortion, and imperfect material reflections can cause ghost artifacts called lens flare.

The image sensor, which measures photon intensity levels via an array of sensor pixels, incurs what amounts to measurement error. With the measured values per pixel, the raw image is passed through an image signal processor (ISP), which is on-device and can include operations such as demosaicking, color balancing, and auto exposure, among others. The data is then compressed in image format, or more commonly in autonomous applications, is encoded as video for streaming to a primary computational resource for computer vision and perception tasks. Each of these processes must be modeled and simulated in order to produce realistic sensor virtual data. These processes and their computational equivalents are further described in the literature [8, 9].

Within the light acquisition process, lens systems, which are used to accumulate additional light, often introduce distortion that can result in measured data that significantly differs from the ground

truth environment. These distortions include radial and tangential distortion, chromatic aberration, vignetting, depth of field, and lens flare. Radial and tangential distortion are non-uniform changes in the direction of light paths commonly seen in wide-angle or fish-eye lenses. With calibration and estimation, these are often reduced in computer vision applications. Chromatic aberration is an effect caused by wavelength-dependent refraction that creates a positional offset of light and pixel colors. Vignetting is a dimming of light near the edges of an image due to the aperture limiting the viewing angle of light to a pixel. Depth of field is the distance at which objects are in focus, with objects away from this distance appearing blurry on the sensor. Lens flare can introduce ghost objects, often blurry regular-shaped light patches stemming from undesired multi-path reflections of intense light. Each of these distortions is specific to the sensor, lens system, and processing algorithms.

3.2 Camera Model Implementation

Generation of rendered data from OptiX is performed through custom ray-tracing kernels that implement physically based materials. The custom implementation allows for balancing accuracy and performance for the given application. Additionally, custom ray-tracing kernels also mean custom ray launch kernels, allowing custom lens models to be implemented directly in the data generation step without needing to introduce additional pre- or post-processing steps. The rendering step for the camera sensor is parameterized by the update rate, image width, image height, horizontal field of view (FOV), sensor lag, exposure time, super-sampling factor for antialiasing, and lens type. The camera is also given an object to which it is attached and a relative position and orientation for attachment. As part of this project, the primary artifacts modeled in the camera simulation were wide-angle lens distortion and noise. These can have significant impact on the quality of data that is available for perception. The lens type parameter defines the model to use for generating rays.

On the modeling of radial and tangential lens distortion, significant work found in literature has been done both to recreate distortions as well as remove distortions from existing images. A summary of classical models is given by Sturm et al. [10], overviewing approaches such as pinhole, fish-eye, and polynomial models. More recent models and techniques for improving distortion modeling are reviewed by Tang et al. [11]. As part of this research, a wide-angle-lens model was implemented based on geometric considerations of a single spherical lens. The model, known as the FOV model, is originally given by

$$r_2 = \frac{\tan(r_1 \tan(\omega))}{\tan \omega} \ , \tag{3.1}$$

where ω is the FOV, r_1 is the undistorted radius, and r_2 is the distorted radius. This model is based purely on geometric considerations of a single spherical lens. While an estimate of the physical parameter may suffice, further calibration is often necessary.

In order to allow the model to be based on the camera's horizontal FOV, and to enforce that the resulting simulated FOV reproduces, exactly, the specified field of view, the implementation is modified to scale the distortion for consistency. Additionally, the distortion is used to modify ray directions rather than modify pixel locations. The resulting model follows

$$\begin{aligned} r_1 &= \sqrt{(x_1^2 + y_1^2)} \\ r_2 &= \frac{\tan(r_1 \tan(\omega))}{\tan \omega} \\ s &= \frac{\tan(\tan(\omega))}{\tan \omega} \\ x_2 &= \frac{x_1 r_2}{r_1 s} \\ y_2 &= \frac{y_1 r_2}{r_1 s} \ , \end{aligned} \tag{3.2}$$

where x_1, y_1 is the undistorted ray direction, x_2, y_2 is the distorted ray direction, ω is half the FOV, and r_1, r_2 are intermediate calculations and correspond to the radius of distorted and undistorted directions.

The basic noise model that still represents true noise seen on image sensors comes from the EMVA Standard [12] model, which specifies that pixel noise can be estimated and modeled as a pixel-dependent Gaussian distribution expressed as

$$\begin{aligned} I_n(p) &= I(p) + \eta, \qquad \eta \sim \mathcal{N}(0, \sigma_p^2) \\ \sigma_p^2 &= K^2 \sigma_d^2 + \sigma_q^2 + K(\mu_p - \mu_{p,dark}) \ , \end{aligned} \tag{3.3}$$

where $I_n(p)$ is the intensity of the noisy pixel p; $I(p)$ is the intensity of the ground truth; η is the noise sampled from a normal distribution parameterized by σ_p which is a function of the sensor gain K, the sensor readout noise σ_d, and the dark noise and shot noise which come from Poisson distributions parameterized by $\mu_{p,dark}$ and μ_p respectively. This statistical model represents data collection illustrated in Figure 3.2.

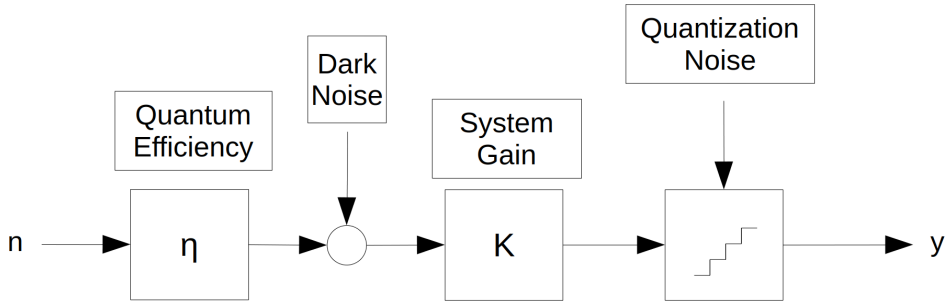


Figure 3.2: Noise model based on EMVA standard.

The original pixel-dependent model was modified by Liu et al. [13] to account for a plausible local spatial and chromatic correlation by approximating the effect of demosaicking. The resulting model follows:

$$\begin{aligned} I &= f(L + \eta_s + \eta_c) + \eta_q \\ \eta_s &\sim \mathcal{N}(0, L\sigma_s^2) \\ \eta_c &\sim \mathcal{N}(0, \sigma_c^2) \ , \end{aligned} \tag{3.4}$$

where I is the noisy image, f is the camera response function (CRF), L is the photon intensity, and σ_c and σ_s are tunable parameters. In order to convert the clean image to photon intensity, the inverse camera response function is applied.

Chrono::Sensor provides several implementations of image noise proposed in the literature. As many of the higher-fidelity image-denoising methods are computationally expensive or require difficult-to-estimate parameters, two basic solutions are provided as initial models for the camera sensor. The first, which is often insufficient in image denoising but may be useful in some perception applications when training for robustness, is additive white Gaussian noise. This noise is sampled independently from a Gaussian

distribution whose parameters are constant across all pixels. The second model, which is based on the standard model and subsequent augmentations found in the literature [12, 13, 14, 15, 16], is written as

$$\begin{aligned} I &= L + \eta \\ \eta &\sim \mathcal{N}(0, L\sigma_1^2 + \sigma_2^2) \ , \end{aligned} \tag{3.5}$$

where I is the noisy pixel, L is the ground truth pixel value, and σ_1 and σ_2 parameterize the distribution dependent on the pixel intensity L. The parameters σ_1 and σ_2 can be estimated for a given camera by using mean removal of noise and estimating noise magnitude based on correlation with intensity. Further model enhancements are in development, including additional ISP modules such as gamma correction and compression artifacts.

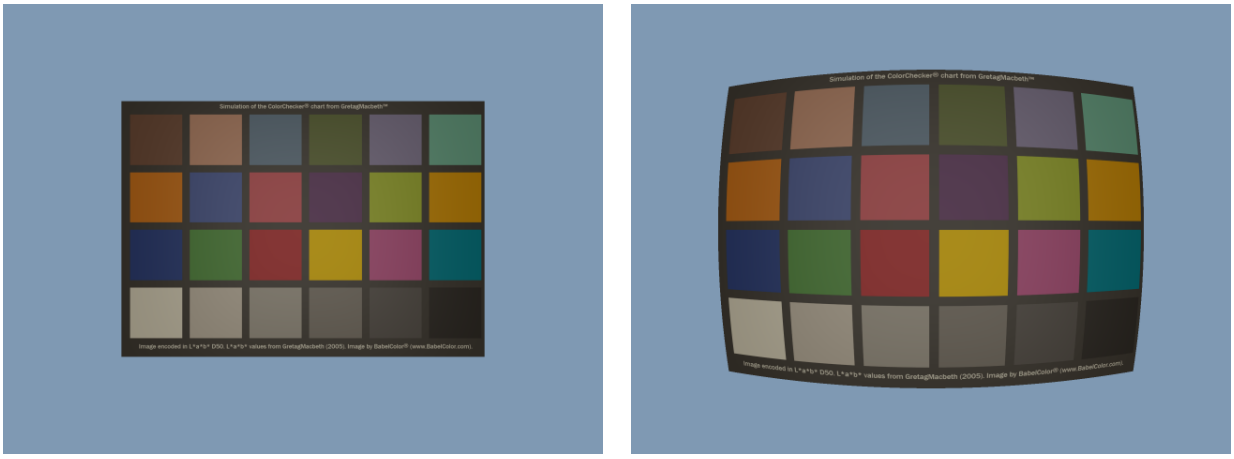
3.3 Camera Simulation Results

To demonstrate lens distortion, which can have a significant impact on the information perceived through a camera sensor, particularly for wide angle lenses, the FOV model defined in Eq. 3.1 was used. The target camera is an ELP 2 megapixel camera with a 2.1 mm lens. The lens has an FOV of 1.408 radians (80.7°). The FOV model, which is derived from geometric considerations, uses $\omega = 1.408/2 = 0.704$.

The results from Chrono::Sensor are shown in Figure 3.3 with an undistorted version on the left, and a distorted version on the right. Further work stemming from this research will look into validating this model and understanding the errors and their impact on realism.

In addition to demonstrated distortion, a qualitative comparison of the camera noise models is made based on empirical image noise. Using the Smartphone Image Denoising Dataset (SIDD) [17], noisy and clean image pairs can be used to estimate noise distributions. While often used for training denoising algorithms, these pairs are used in this demonstration to estimate noise levels and introduce artificial noise to synthetic images.

For the additive white Gaussian noise (AWGN) model, the standard deviation, assuming identically distributed Gaussian noise, can be estimated by subtracting the clean image from the noisy image and



(a) Undistorted image using a pin-hole camera model.

(b) Distorted image using a modified field of view model with single physics-based input parameter.

Figure 3.3: A comparison of undistorted and distorted images using pinhole and FOV models. The distortion seeks to replicate an ELP 2MP camera with a 2.1 mm lens and ∼80° FOV.

calculating the standard deviation. The AWGN that best approximated the empirical noise had a standard deviation of 0.0093 for images in the range of 0 to 1.

For the intensity-dependent model, the parameters are obtained by binning the pixels based on intensity. The standard deviation for each intensity can then be estimated, and a regression will yield a linear relationship between variance and intensity. For the sample images, the linear regression yielded a relationship of $\sigma^2 = 0.0325^2 I + 0.00627^2$. The correlation is shown in Figure 3.4. The standard deviations were calculated using 143.8 million noisy pixel values in the linear RGB space as true image noise is measured at the image sensor (RAW image).

Sample noisy images based on the estimated parameters are shown in Figure 3.5. The simulated images are generated at the same resolution as the real data for ease of comparison. Since the images are obtained at high resolution and down-scaling distorts noise, a zoomed comparison is given in Figure 3.6, where individual pixels are visible. While the standard model cannot account for spatial, chromatic, and temporal correlation, the dataset used has little compression or post-processing, resulting in a more representative domain for the standard model. Even so, the simulated images have noticeably finer noise than the real image. In autonomous vehicles applications, further research is needed to understand the level of fidelity required from the noise model for meaningful simulations. In addition, further comparisons

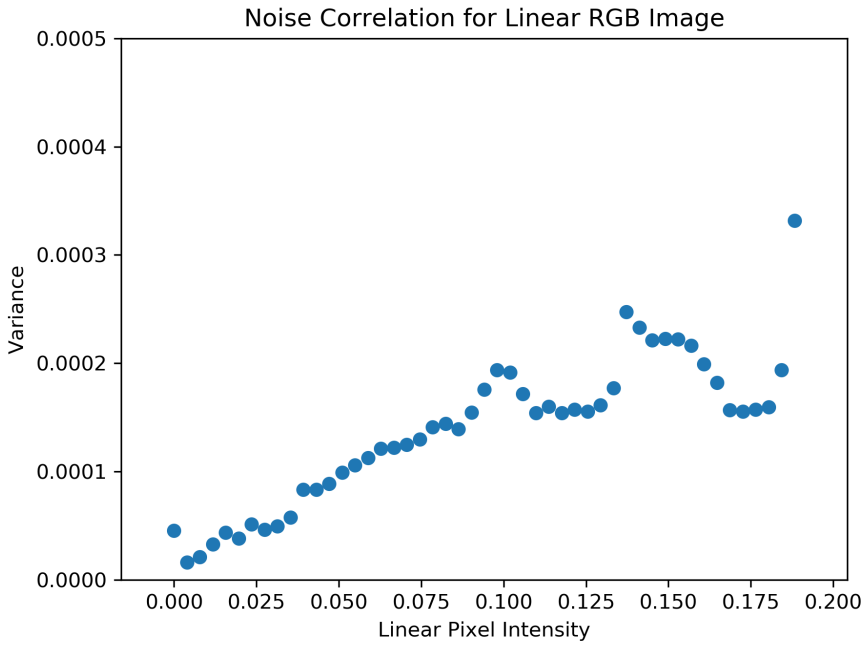
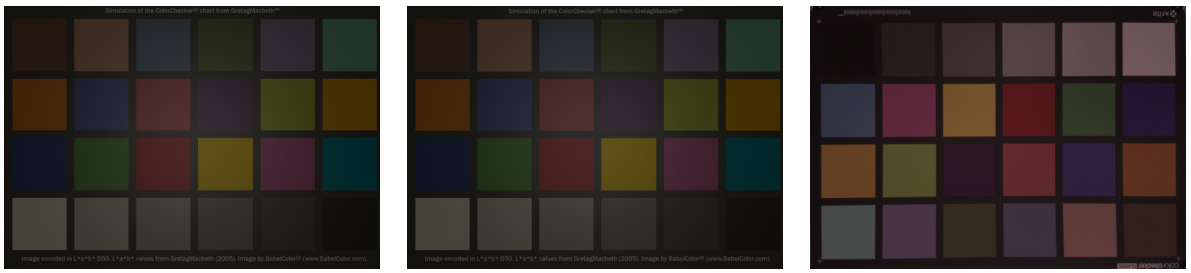


Figure 3.4: Correlation between variance and color intensity in linear images from SIDD.

between the noise models are needed, and validation will require implementation of additional ISP modules and improvement of the scene similarity to remove compounding effects.



(a) Additive white Gaussian noise, $\sigma = 0.0093$.

(b) Intensity-dependent Gaussian noise, $\sigma^2 = 0.0325^2 I + 0.00627^2$.

(c) Real image noise example from SIDD.

Figure 3.5: Comparison of full-resolution noisy images.

The underestimation of noise due to assuming AWGN is explained by the majority of pixels having low intensity. This means that since there is a correlation, the AWGN assumption was a low estimate and exposed the weaknesses in the identical distribution assumption. Only visual comparison can be made at this time, with the standard model appearing to be closer to reality. While correlation components

are known to be missing, realism benchmarks are needed to understand the usefulness of each of these models in simulations for evaluating autonomous navigation and perception.

(a) Additive white Gaussian noise, $\sigma = 0.0093$.

(b) Intensity-dependent Gaussian noise, $\sigma = 0.0499I + 0.00666$.

(c) Real image noise example from SIDD.

Figure 3.6: Comparison of a small section of noisy images illustrating differences in model assumption.

In addition to model accuracy, performance and scaling results are important to understand the support for simulating many vehicles and sensors.

As the addition of image noise requires random sampling from unique distributions, it is important to understand the performance impact of the noise model. A scaling analysis was conducted to compare the generation of noised and un-noised images. The analysis was performed on an Nvidia RTX 2080ti GPU for a camera operating at 30 Hz within a simulated scene that included the ColorChecker [18] pattern. Figure 3.7 shows the increase in wall time needed to perform 10 seconds of simulation with increasing image size. While the noisy images often require additional computation time, the impact is negligible and independent of scene complexity.

Since the ray-tracing algorithm is computationally complex, it is also important to contextualize the performance of the simulation when the camera takes higher resolution or, analogously, multiple samples are used per image pixel for super-sampling. Since the implemented super-sampling algorithm requires an image-reduction step, both scaling studies are provided and can be seen in Figures 3.8 and 3.9. All simulations were run for 10 simulation seconds with the wall time required to complete the simulation recorded. Each simulation included a single high-resolution visualization mesh with 173k triangles. For super-sampling, the image resolution was fixed at 640x360 to match the total samples from the image size scaling.

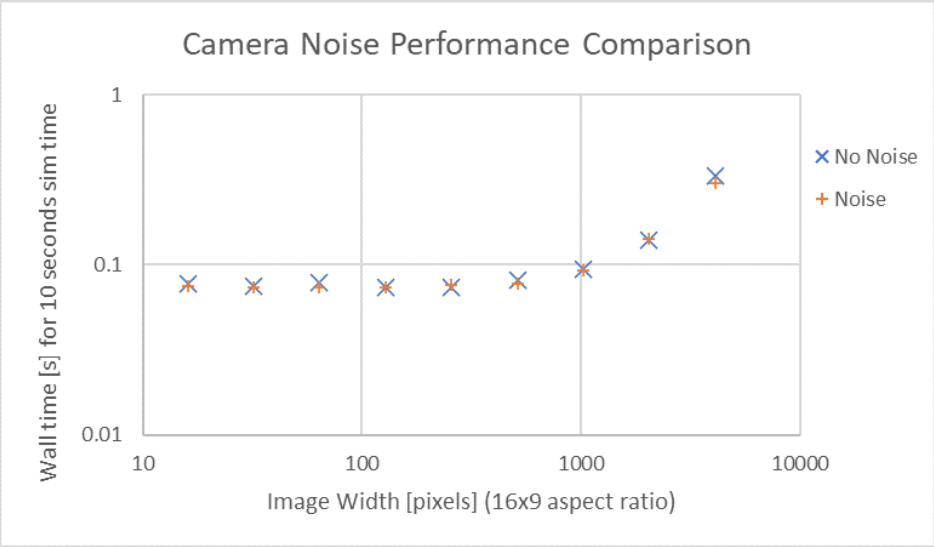


Figure 3.7: Scaling of a single camera with and without noise shows little impact of noise augmentation on simulation performance. Configuration: 16x9 camera aspect ratio, 30 Hz update frequency, 10 seconds of simulation time, color calibration object scene, RTX 2080ti GPU.

Figure 3.8: Scaling of a single camera shows the performance impact of increasing image size. Configuration: 16x9 camera aspect ratio, 30 Hz update frequency, 10 seconds of simulation time, 173k triangle scene, RTX 2080ti GPU.

The final performance results for camera simulation shown here are for inclusion of multiple cameras in the simulation. Each camera had a resolution of 1280x720 with a 30 Hz update rate. The scene was

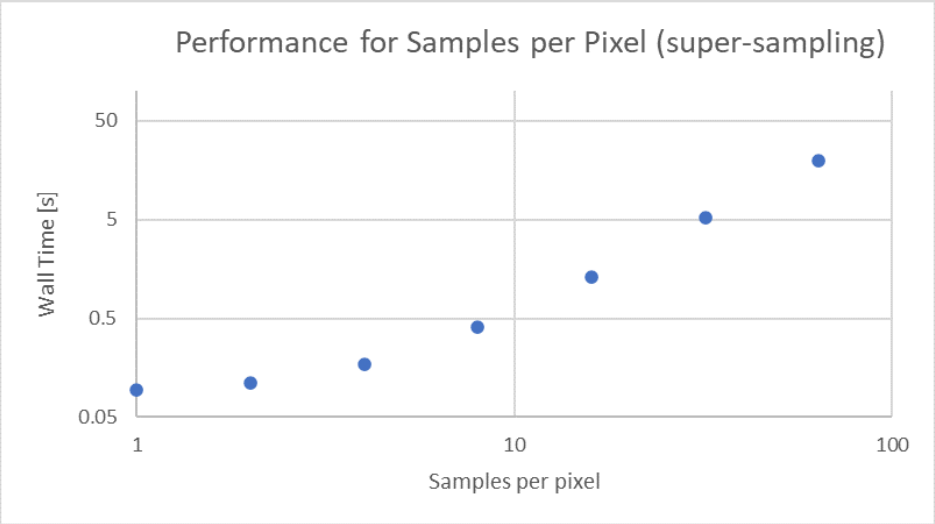


Figure 3.9: Scaling of per-pixel samples for a single camera shows impact of super-sampling on simulation performance. Configuration: 640x360 resolution, 30 Hz update frequency, 10 seconds of simulation time, 173k triangle scene, RTX 2080ti GPU.

a single high-resolution mesh with a duration of 10 simulation seconds. The results in Figure 3.10 show the computation time required for including additional cameras. The scaling included up to 256 cameras on an RTX 2080ti.

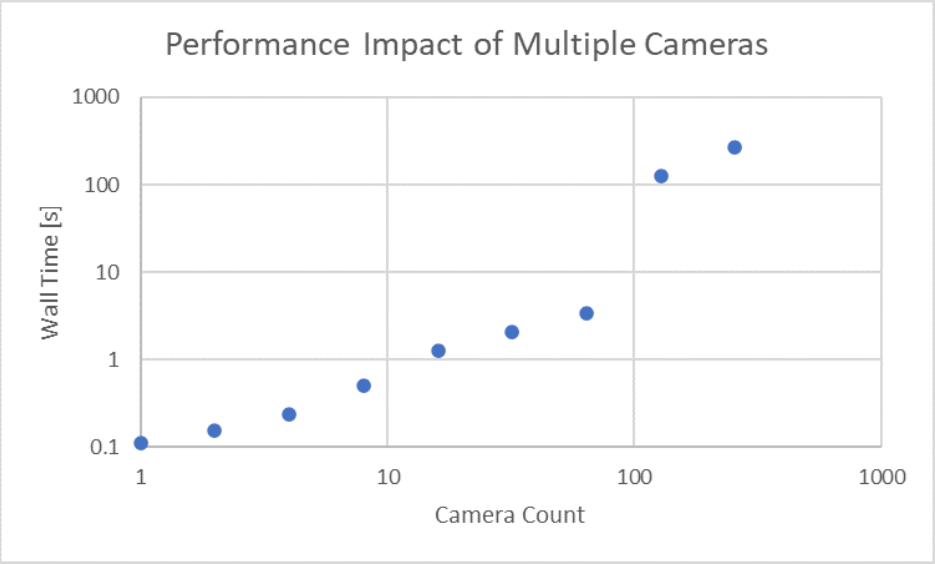


Figure 3.10: Scaling of cameras in a single scene shows the capability and performance impact of many cameras. Configuration: 1280x720 resolution, 30 Hz update frequency, 10 seconds of simulation time, 173k triangle scene, RTX 2080ti GPU.

4. Lidar Modeling and Simulation

4.1 Lidar Model Background

Since lidar plays a key role in perception and planning algorithms, the lidar must model accurate beam returns in the highly complex virtual environments used for testing and training autonomous navigation. As automotive and robotic applications often rely on long-range and relatively low-cost lidar, beam divergence plays a significant role in the collected data. Tightly coupled with beam divergence, the power distribution across the beam spot is often modeled using a Gaussian distribution [19], adding complexity to a simulated sensor. In addition, since the lidar beam spot grows over distance, methods have been implemented on-chip to utilize the beam spread to compute specific or multiple returns (i.e., first return, last return, strongest return, first and strongest, etc.) [2]. The interplay between beam spread and return type is illustrated in Figures 4.1 and 4.2, where return type can result in difference data when multiple objects are reflected.

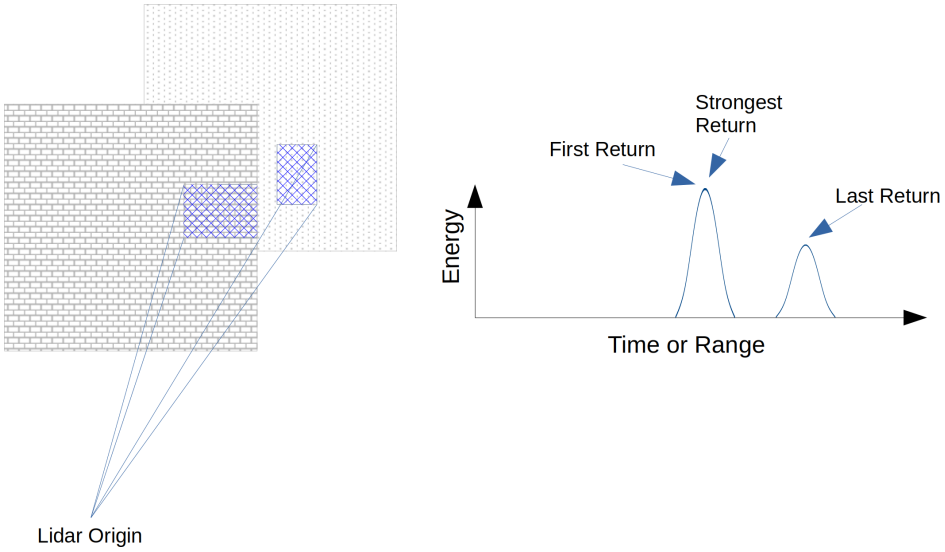


Figure 4.1: Multiple or partial return phenomena utilized by lidar to detect multiple objects per beam. In this example, the strongest return will also be the first return. Modified from velodynelidar.com [2].

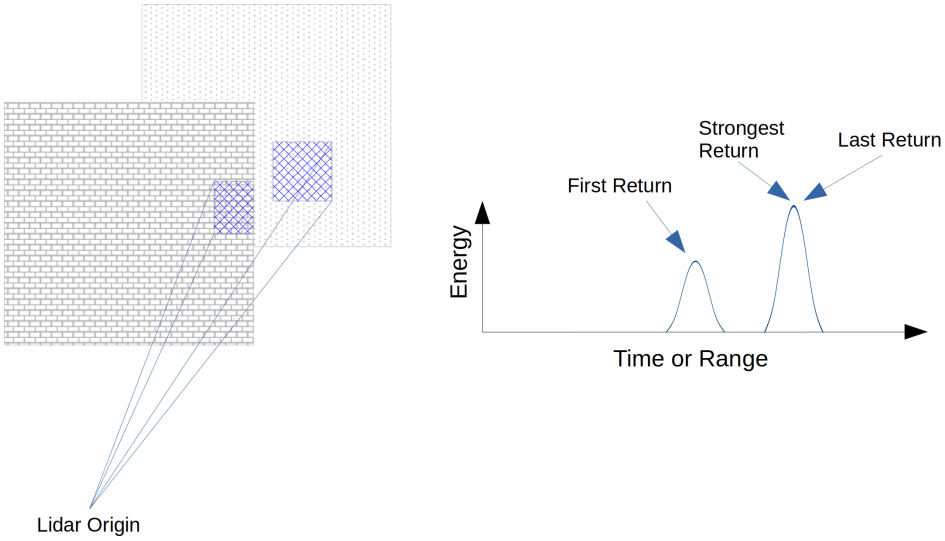


Figure 4.2: Multiple or partial return phenomena utilized by lidar to detect multiple objects per beam. In this example, the strongest return will also be the last return. Modified from velodynelidar.com [2].

4.2 Lidar Model Implementation

For generating lidar data, the same ray-tracing methodology used in the camera sensor is applied, but because complete control is allowed over the material and the ray launch algorithm, the lidar implements its own rendering options. For materials, the encoded value is defined to be the intensity of the return from the direction of the ray. For ray launch, a vertical and horizontal angle is measured, alleviating all FOV issues associated with a traditional rendering plane. The lidar rays are then traced in the lidar-specific space, ignoring all parameters and materials associated with the camera. In this way, materials that depend on specific lidar wavelengths could be implemented. This is a source of potential expansion and research as little work regarding lidar-wavelength-specific materials can be found in the literature.

The Chrono::Sensor lidar is parameterized by update rate, width and height of the samples in a frame, horizontal and vertical FOV, sensor lag, data collection time, number of samples to use per beam, the beam divergence angle that is used for multiple returns, the return mode when multiple objects are seen, and the method for generation, which defaults to ray cast with potential for expansion to additional path-tracing models. The lidar can also be attached to an object in the Chrono simulation and can be mounted with a relative position and orientation to match the corresponding mounting configuration of a real lidar on a vehicle.

In accounting for multiple returns, a beam discretization model based on work by Goodin et al. [20] is extended in Chrono::Sensor. This leverages the ray-tracing algorithm to discretize the beam for each lidar ray. The beam discretization is parameterized based on the width of the beam, in number of samples, and the angle of divergence.

The original model upon which this implementation was based was shown to produce plausible results in complex grassy scenery [20]. Figure 4.3 illustrates the sampling of a rectangular beam pattern. A similar, but computationally intensive, discretization is discussed by Yin et al. [21] where multiple laser pulses are simulated, with each return modeled as a Gaussian distribution with approximated mean and standard deviation based on the return distance and material back scattering.

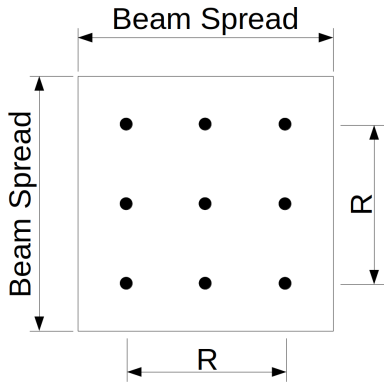


Figure 4.3: Sampling pattern for discrete lidar beam simulation. The discretization is parameterized by the beam divergence angle and the width in samples of the beam.

The current state of the art limits the sampling resolution to nine samples per beam to estimate a return pattern. As demonstrated here, nine samples provides a crude estimate of intensity for strongest return, but is a significant improvement from the single-ray beam approximation used in many game-engine-based simulators. That may in part help explain the apparently large realism gap with synthetic lidar data in combination with limited scene realism.

The implemented model uses the rectangular beam approximation illustrated in Figure 4.3. Augmenting work found in literature, this implementation parameterizes the model by the sample radius such that the discretization can be increased as part of the model.

The lidar noise model is implemented as a Gaussian distribution with independently parameterized distributions applied to range, intensity, vertical angle, and horizontal angle. The noise is applied directly to the raw measurements (range and intensity) rather than being applied to the processed point cloud.

This is critical since generation of a point cloud (x,y,z,intensity) is a post-processing step in lidar sensors. As noise from a real sensor is made at the time of measurement, the simulated lidar is implemented in the same order. The parameters for the distributions are user-defined and would need to be estimated from lidar data, or found in a data sheet.

4.3 Lidar Simulation Results

To demonstrate the lidar implementation, a comparison between beam discretization options is given below. Here, one, nine, and 81 samples are used to sample a single lidar beam. The one-sample case corresponds to most game-based simulation solutions; the nine-sample case corresponds to the state of the art found in the work by Goodin et al. [20]. To study the impact of multiple returns on a simulated point cloud, a single lidar beam is simulated, and a scenario is created that will force multiple returns. The scenario includes two walls at different distances that slide together in front of the lidar beam. The beam spot relative to the gap between the walls serves as the independent variable in the simulated experiment. The two walls are 50 and 60 m away from the lidar. The beam spot, based on the HDL-32E lidar with 3 mrad divergence [2] results in a beam spot width of 15 cm at 50 m. The setup is illustrated in Figure 4.4.

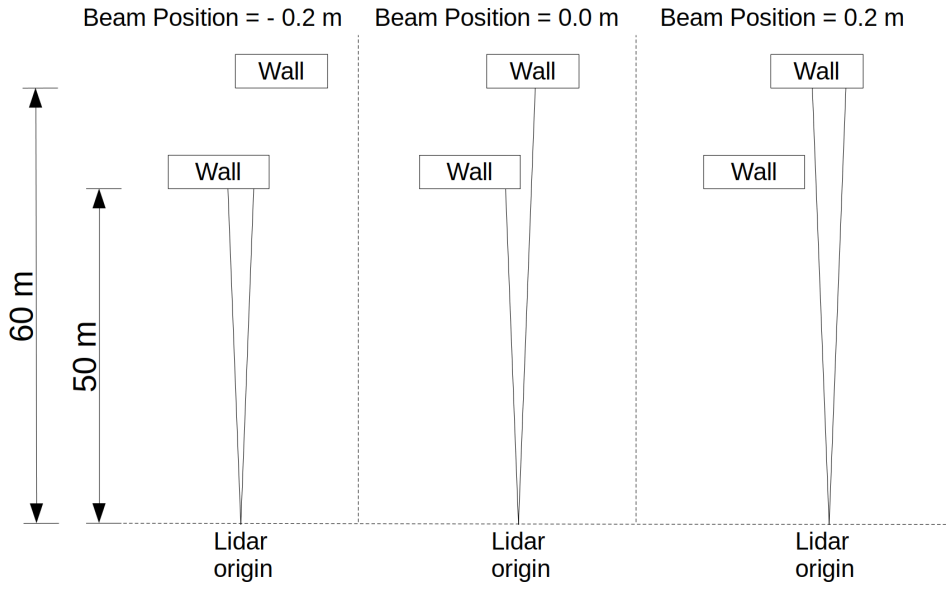
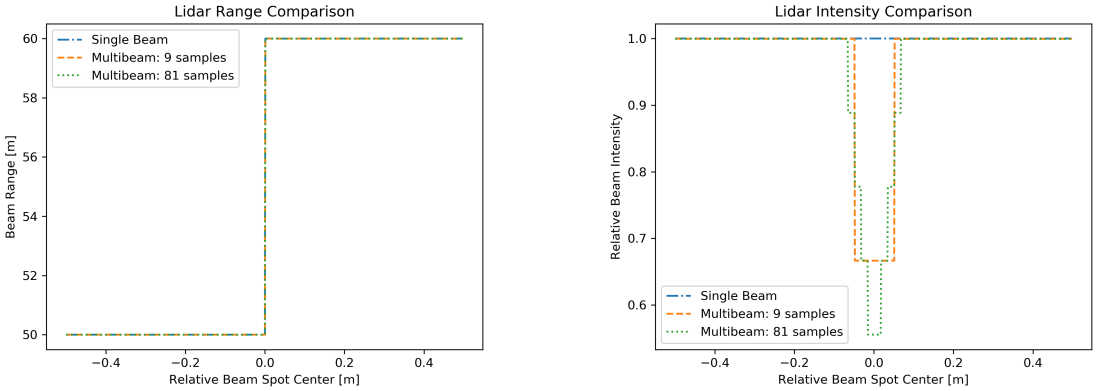


Figure 4.4: Setup for beam divergence and sampling discretization test.

As the gap in the wall slides past the beam spot, the range based on strongest return (only return for single ray) and intensity are plotted. The range in Figure 4.5a, shows the same behavior for each method as expected since the range will shift at the same position relative to the wall. The intensity in Figure 4.5b shows significant difference due to the sampling size. The expected strongest return should result in a relative intensity of 0.5 (neglecting power loss over distance and assuming perfect reflectance). The single ray will return full power since the wall is perpendicular to the ray and one wall is seen at each position. The nine-sample beam experiences a dip in power as the beam transitions from one wall to the other, but does so in a coarse manner as the beam only provides three samples in the lateral direction. The 81-sample beam, which has nine samples in the lateral direction, results in a fine discretization of intensity, nearly reaching the expected lower relative intensity of 0.5.



(a) Beam sampling shows no effect of wall-gap multi return scenario, but would have significant impact when simulating lidar in fine scene regions such as fences and vegetation.

(b) Even for the simple comparison of an offset wall, beam intensity is significantly impacted by beam sampling and divergence.

Figure 4.5: Comparison of sampling in the beam divergence model and its impact on returned range and intensity for a single beam with two offset walls at 50 and 60 m. Both walls are assumed to have perfect reflectance, and power dissipation is neglected.

This work shows that there is significant room for improvement in the beam divergence model, particularly by harnessing recent hardware advances in ray tracing. Future work should focus on understanding the impact of beam divergence on data realism and introduce a model that provides a significant level of realism without an excessive level of computational burden. Along with beam

divergence, power dissipation and scattering will be included in such a model so that the relative intensity provides a close match to reality.

Since higher accuracy can be obtained for higher-sample lidars, the scaling in Figure 4.6 shows the computation time for increasing the number of samples per lidar. Since the reduction time for samples per beam is low, the performance for samples per beam and beams per scan are nearly equivalent given the same total number of rays launched by the ray-tracing algorithm. The results in Figure 4.6 give the overall impression of increasing the ray count for a lidar in Chrono::Sensor.

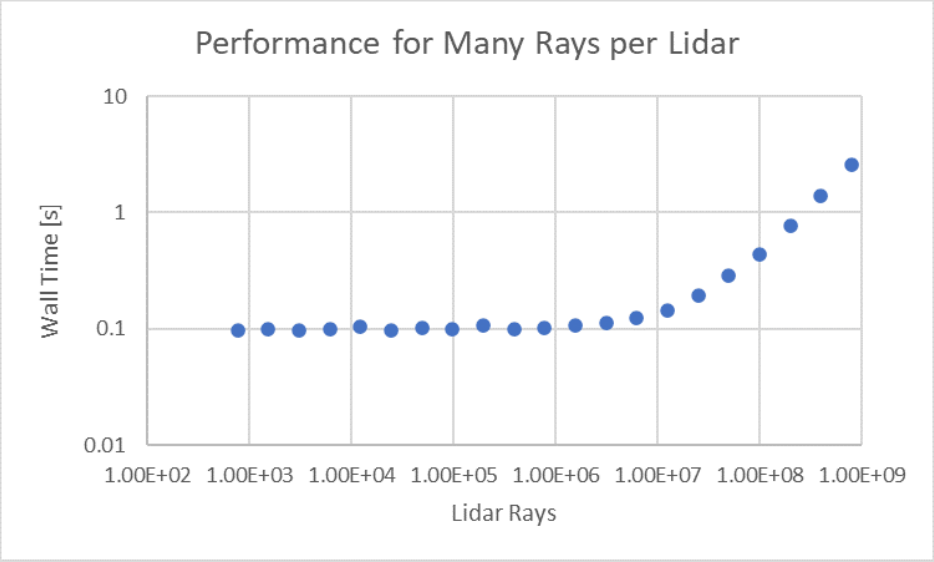


Figure 4.6: Scaling of rays for a single lidar showing the potential for increasing the resolution of a multi-sampled beams or many-beam lidars. Configuration: 5Hz update frequency, 10 seconds of simulation time, 173k triangle scene, RTX 2080ti GPU.

To show the performance burden for increasing the number of lidars in a simulation, a scaling analysis is shown in Figure 4.7. The number of lidars is increased from 1 to 256 with the time required to perform 10 seconds of simulation shown for each configuration. The scene includes the high-resolution mesh described previously. The update frequency of the lidar is set to 5 Hz, which is within the operating range of a Velodyne HDL-32E.

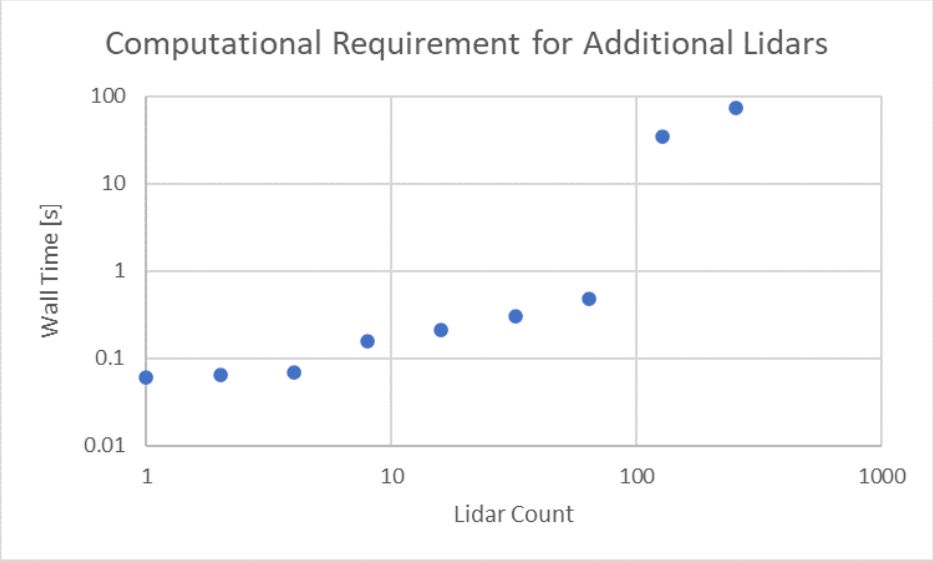


Figure 4.7: Scaling of lidar in a single scene shows the capability and performance impact of simulating many lidars. Configuration: resolution: 3800x48, 5Hz update frequency, 10 seconds of simulation time, 173k triangle scene, RTX 2080ti GPU.

5. GPS Modeling and Simulation

The Global Positioning System (GPS) is a localization instrument that allows robots and autonomous vehicles to pinpoint their location. The receiver listens to signals broadcast from GPS satellites to compute its own location. The receiver uses orbital and time information from the satellite to compute a distance to the satellite and through trilateration can pinpoint its own coordinates in space. This process is illustrated in Figure 5.1 for the two-dimensional case.

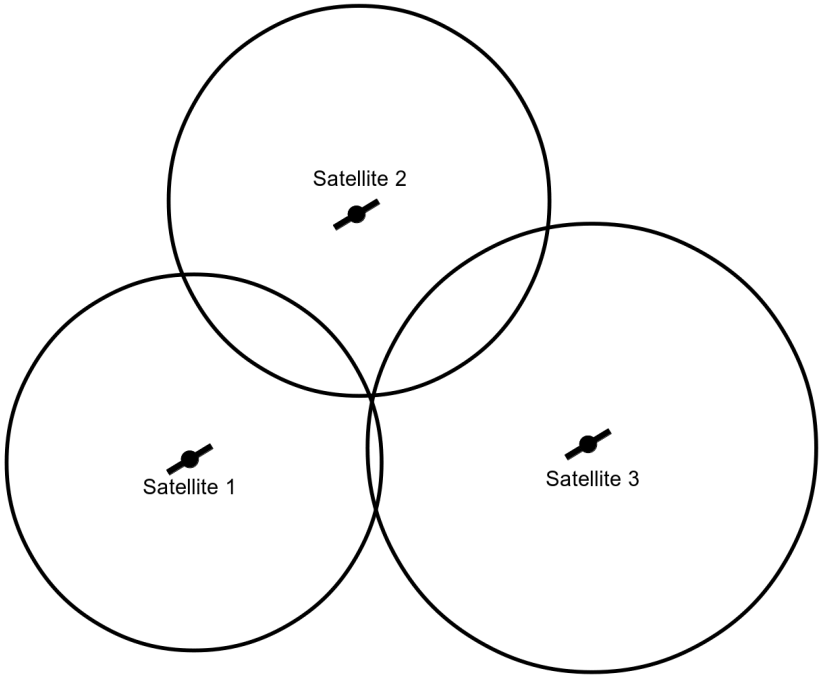


Figure 5.1: Illustration of two-dimensional GPS trilateration.

In three dimensions, a distance to a single satellite places the receiver on the surface of a sphere with radius equal to this distance. With a second satellite, the location is restricted to a circle in space. With a third satellite, this is narrowed to two points in space. With an assumption of interior or exterior point, these three satellites can be enough to pinpoint a location, although four satellites is the minimum, with no assumptions, to guarantee a known location.

The GPS model in Chrono::Sensor is parameterized based on the update rate, the sensor lag, the collection window, a reference GPS location for the origin of the simulation, and a desired noise model.

The sensor is then attached to a Chrono body with a relative position and orientation. Since the receiver orientation is irrelevant, the orientation component of relative attachment is ignored.

In Chrono::Sensor, GPS ground truth information is generated using a combination of the positional state of the object to which the sensor is attached, and the relative position of attachment. Because a GPS often has a low update rate and high lag, the positional data from the simulation is averaged over the specified data collection window before introducing noise. Once noise is applied, the data is held and only provided to the user once the lag time has elapsed. This model of lag is applicable to all sensors in Chrono. Using the GPS reference location and a spherical mapping from simulation coordinates to global coordinates, the modeled position is converted to latitude, longitude, and altitude. Importantly, the noise model is applied before conversion to enforce non-distorted distributions when simulating far from the equator.

The current GPS noise model uses independent Gaussian distributions parameterized independently for latitude, longitude, and altitude. Although higher complexity models exist that account for GPS lock and satellite visibility [22], many modern GPS combine information from an IMU, additional satellites, and complex filtering algorithms to considerably reduce noise. While noise still exists, it is often distorted beyond the modeling capability shown by Balaguer and Carpin [22]. The principle of tracking satellite positions to calculate visibility is also shown by Durst and Goodin [23] where paths from the receiver to satellite are traced. Then, based on the distances, noise models accounting for atmospheric effects, including viewing angle through the atmosphere, can be included. This high-fidelity model then performs a generic trilateration algorithm to mimic the error from conflicting satellite information. This introduces significant modeling complexity if receiver parameters and algorithms are unknown. Further work needs to be done to study the realism of the model presented by Durst and Goodin [23] compared to the highly accurate and filtered GPS used in automotive applications.

6. IMU Modeling and Simulation

An Inertial Measurement Unit (IMU) is commonly included on autonomous vehicles to assist in pose estimation. These sensors most commonly include a gyroscope and accelerometer, but can also include magnetometer. Currently, the accelerometer and gyroscope are included in the IMU implementation, with further work being done to include a magnetometer with noise and drift. An accelerometer returns a local translational acceleration in three directions. The acceleration is relative to free-fall, meaning zero acceleration is returned only if there are no forces acting on the sensor. A gyroscope measures an angular velocity about the three local axes.

The implemented IMU is parameterized by the update rate, the sensor lag, a data collection window, and a noise model. The ground truth data is calculated using internal state information from the Chrono body to which the sensor is attached and the relative attachment position and orientation. The ground truth data from Chrono is averaged over the data collection window, augmented with noise, and provided to the user after a period of time defined by the sensor lag.

The IMU noise model implemented in Chrono::Sensor is based on work by Shah et al. [24] which showed promising results for recreating accelerometer and gyroscope noise using a Gaussian model with drifting parameters. The noise model for the gyroscope is given by,

$$\begin{aligned} \omega_{output} &= \omega + \eta_a + b_t, \qquad \eta_a \sim N(0, r_a) \\ b_t &= b_{t-1} + \eta_b, \qquad \eta_b \sim N(0, b_0\sqrt{\frac{dt}{t_a}}) \ , \end{aligned} \tag{6.1}$$

where ω_{output} is the gyroscope reading and ω is the ground truth angular velocity relative to the sensor. Noise variance r_a, bias b_0, and time constant t_a are user-defined parameters; η_a and η_b are random variables, sampled from a Gaussian distribution as parameterized above, accounting for variation in noise and bias drift [24].

Accelerometer noise follows a similar distribution, but the raw data calculation must account for gravity:

$$
\begin{aligned}
a_{output} &= (a - g) + \eta_a + b_t, \qquad \eta_a \sim N(0, r_a) \\
b_t &= b_{t-1} + \eta_b, \qquad \eta_b \sim N(0, b_0 \sqrt{\frac{dt}{t_a}}) \ ,
\end{aligned}
\tag{6.2}
$$

where a_{output} is the accelerometer reading, a is the ground truth translational acceleration of the sensor, g is the gravitational acceleration, and dt is the update period. Noise variance r_a, bias b_0, and time constant t_a are user-defined parameters; η_a and η_b are random variables, sampled from a Gaussian distribution as parameterized above, accounting for variation in noise and bias drift [24]. The current model could be expanded as shown by Durst and Goodin [23] to implement an approach that additionally incorporates the effect of temperature on the sensor noise. Chrono::Sensor allows for this expansion by implementing a custom sensor that can use the same framework for generating sensor data and applying augmentation filters.

Simulated IMU results are shown for a stationary sensor, equivalent to an IMU resting on a table. The virtual sensor noise is set with accelerometer parameters: $r_a = 0.0075$, $b_0 = 0.001$, and $t_a = 0.1$, and gyroscope parameters: $r_a = 0.001$, $b_0 = 0.0$, $t_a = 0.0$. The stationary results are compared to results from a stationary mobile phone accelerometer and gyroscope. This is not a validation study, and the mobile sensor deviates from automotive sensors due to calibration and lack of filtering, but shows example data generated by Chrono::Sensor. The comparison in Figure 6.1 shows example noise for a single channel of an accelerometer and gyroscope. The simulated data is plotted alongside the real data for visual comparison to see the quality of the data, not to show validation.

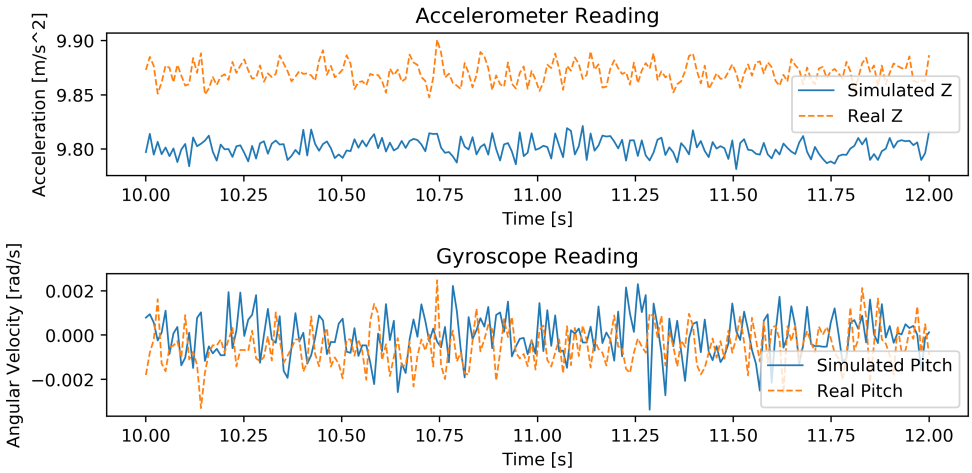


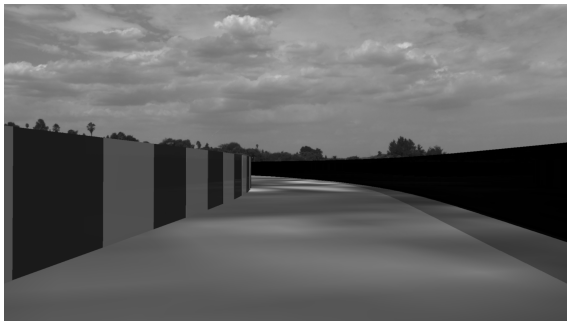
Figure 6.1: Demonstration and qualitative comparison of accelerometer and gyroscope noise between simulation and real IMU.

7. Demonstration of Technology

Two examples are discussed next to demonstrate the Chrono::Sensor platform. The first example shows a use for evaluating a navigation algorithm, and the second shows a demonstration of capability in a replica virtual environment.

For demonstrating evaluation, a scaled autonomous vehicle was placed in a closed track to evaluate the effectiveness of a lidar-only navigation strategy. The strategy used the simulated lidar in Chrono::Sensor to detect the track boundaries and generate a safe pathway between these clusters. The vehicle was able to safely navigate the previously unseen track. The camera output from the demonstration can be seen in Figure 7.1 shows an RGB third-person view and a greyscale first-person view from the front of the vehicle. The lidar output, which was used in the navigation algorithm, can be seen in Figure 7.2.

(a) Third-person view of scaled vehicle on closed track.

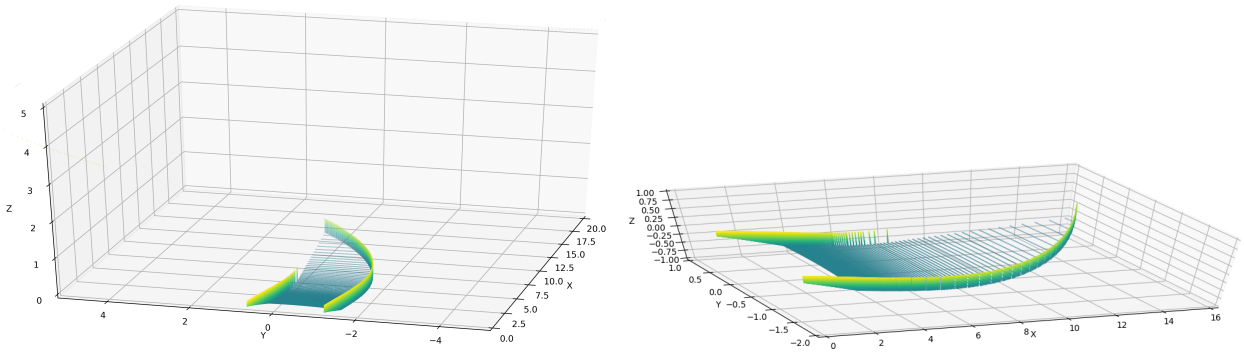


(b) Grey-scale image from on-board camera mounted to the scaled autonomous vehicle.

Figure 7.1: Simulated images from a demonstration of a scaled autonomous vehicle navigating a closed track using lidar to detect track boundaries.

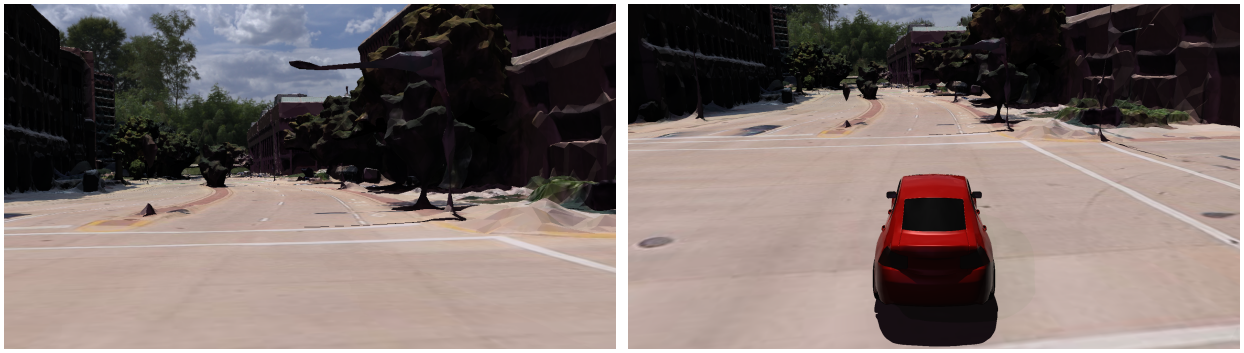
As an on-road scenario example, we provide a second demonstration using a virtual reconstruction of Park Street in Madison, WI, courtesy of Continental Mapping [25]. The demonstration shows the simulated camera output in Figure 7.3.

Simulated lidar, based on a virtual roof-mounted lidar with parameters representative of widely used automotive lidar [2], is shown in Figure 7.4. Figure 7.4a shows a three-dimensional rendering of the generated point cloud including details such as trees and buildings from the virtual environment. Figure 7.4b shows a top-down view of the simulated point cloud illustrating the sensor's inclusion of roadways and building outlines.



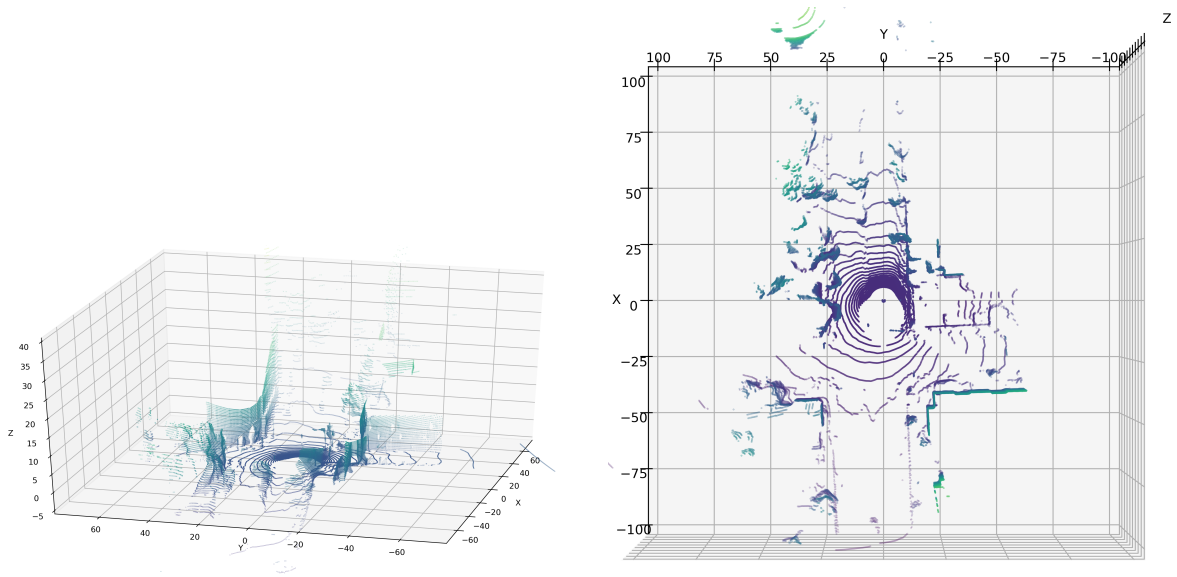
(a) (b)

Figure 7.2: Simulated point clouds from a demonstration of a scaled autonomous vehicle navigating a closed track using lidar to detect track boundaries. Point cloud color encodes height of the point for ease of perception.



(a) Hood-mounted camera (b) Third-person camera

Figure 7.3: Simulated camera output for a sedan navigating a virtual replica of Park Street in Madison, WI. Third-person view only for context.



(a) Three-dimensional view of rendered point cloud.

(b) Birds-eye view of simulated point cloud.

Figure 7.4: Simulated lidar data for a sedan navigating a virtual replica of Park Street in Madison, WI. Point cloud color encodes intensity and height for visual perception only.

8. Conclusions

As part of this SAFER-SIM project, we established a simulation framework for modeling and simulating sensors for training and evaluation of autonomous vehicles. The simulation framework leverages Chrono for generating ground truth dynamic data for interoceptive sensors. For exteroceptive sensors, the scene evolution is driven by the dynamics in Chrono and includes vehicle dynamics and contact. With Chrono::Sensor, users can add virtual sensors to existing simulations to train and evaluate algorithms for perception and navigation by incorporating the algorithm in a software-in-the-loop virtual test.

Currently supported sensors include camera, lidar, GPS, and IMU with the capability for a user to extend and implement a custom sensor that can leverage all existing components of the framework. Each sensor includes parameters to define the data collection and augmentation process. The camera and lidar sensors leverage hardware-accelerated ray tracing that allows for custom implementation of materials (for camera, lidar, or custom sensor), and efficient scaling and data augmentation. Each sensor model accounts for noise with varying levels of sophistication.

8.1 Outcomes

Outcome performance measures:

- We produced an open-source sensor simulation framework for autonomous vehicle simulation called Chrono::Sensor. This code will be provided alongside the open-source Project Chrono. Both Chrono and Chrono::Sensor are developed/augmented by the Simulation Based Engineering Lab at the University of Wisconsin-Madison.
- Chrono::Sensor is and will continue to be used in further work involving the research and development of a multi-agent connected autonomous vehicle simulator.
- Chrono::Sensor will be augmented and used in related research on improving and understanding sensor realism for reducing the simulation to reality gap.
- This project was the basis of a successful Ph.D. Preliminary Examination in May 2020.
- Chrono::Sensor is the subject of five conference presentations and submissions, one journal publication, and two pending journal submissions. [26, 27, 28]

8.2 Impacts

This research and the subsequent sensor simulation framework could make a difference by allowing researchers to better understand the safety of autonomous vehicles, improve autonomous vehicle perception and navigation, and demonstrate the capability of algorithms in safety-critical scenarios to the public at large. Specifically, the technology associated with this SAFER-SIM project, and the broader software to which this module belongs, is designed with the following intent:

- Allow researchers to better understand and improve the safety of autonomous vehicles by facilitating numerous iterative simulations in safety-critical scenarios.
- Allow for the understanding and demonstration of vehicle safety and capability for the public at large.

8.3 Future Work

We are in the process of:

- releasing the platform as open source by the end of summer 2020
- making further improvements to the simulation framework
- making further improvements to the sensor models
- providing support for additional sensors
- researching additional methods/models for sensor simulation

Chrono::Sensor is still in development and will be made public in the coming weeks. It will be released as a module in the open-source multi-physics simulation platform, Project Chrono [29]. The initial framework and models in Chrono::Sensor were made possible through this SAFER-SIM project, and the framework will be expanded as part of a Ph.D thesis. Future work centers around expanding the supported sensors to include a broader set of automotive sensors including radar, odometer, encoders, etc. Additionally, prepackaged solutions that represent commonly used sensors will be provided for ease of use.

Further research will also focus on benchmarking the realism of the current noise and distortion models, and seeking to develop new models where the current solutions fail to provide satisfactory realism.

Bibliography

[1] G. Girardin, "Road to robots. sensors and computing for autonomous vehicle," in *Autonomous Vehicle Sensors Conference*, 2018.

[2] "Velodyne lidar." `http://velodynelidar.com/`, 2018. Accessed: 2020-04-17.

[3] Project Chrono, "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems." `http://projectchrono.org`, 2020. Accessed: 2020-03-03.

[4] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut, "Chrono: An open source multi-physics dynamics engine," in *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science* (T. Kozubek, ed.), pp. 19–49, Springer, 2016.

[5] R. Serban, M. Taylor, D. Negrut, and A. Tasora, "Chrono::Vehicle Template-Based Ground Vehicle Modeling and Simulation," *Intl. J. Veh. Performance*, vol. 5, no. 1, pp. 18–39, 2019.

[6] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: A general purpose ray tracing engine," *ACM Transactions on Graphics*, August 2010.

[7] NVIDIA Corporation, "NVIDIA Turing GPU Architecture," 2018. WP-09183-001 v01.

[8] J. E. Farrell, P. B. Catrysse, and B. A. Wandell, "Digital camera simulation," *Applied Optics*, vol. 51, no. 4, pp. A80–A90, 2012.

[9] J. E. Farrell and B. A. Wandell, "Image systems simulation," *Handbook of Digital Imaging*, pp. 1–28, 2015.

[10] P. Sturm, S. Ramalingam, J.-P. Tardif, S. Gasparini, J. Barreto, *et al.*, "Camera models and fundamental concepts used in geometric computer vision," *Foundations and Trends ® in Computer Graphics and Vision*, vol. 6, no. 1–2, pp. 1–183, 2011.

[11] Z. Tang, R. G. von Gioi, P. Monasse, and J.-M. Morel, "A precision analysis of camera distortion models," *IEEE Transactions on Image Processing*, vol. 26, no. 6, pp. 2694–2704, 2017.

[12] EMVA Standard, "Standard for characterization of image sensors and cameras," *European Machine Vision Association*, vol. 3, 2010.

[13] C. Liu, R. Szeliski, S. B. Kang, C. L. Zitnick, and W. T. Freeman, "Automatic estimation and removal of noise from a single image," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 299–314, 2008.

[14] S. W. Hasinoff, F. Durand, and W. T. Freeman, "Noise-optimal capture for high dynamic range photography," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 553–560, IEEE, 2010.

[15] S. Guo, Z. Yan, K. Zhang, W. Zuo, and L. Zhang, "Toward convolutional blind denoising of real photographs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1712–1722, 2019.

[16] R. Jaroensri, C. Biscarrat, M. Aittala, and F. Durand, "Generating training data for denoising real rgb images via camera pipeline simulation," *arXiv preprint arXiv:1904.08825*, 2019.

[17] A. Abdelhamed, S. Lin, and M. S. Brown, "A high-quality denoising dataset for smartphone cameras," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[18] D. Pascale, "RGB coordinates of the Macbeth ColorChecker," *The BabelColor Company*, vol. 6, 2006.

[19] F. Castaño, G. Beruvides, A. Villalonga, and R. E. Haber, "Computational intelligence for simulating a lidar sensor," in *Sensor Systems Simulations*, pp. 149–178, Springer, 2020.

[20] C. Goodin, M. Doude, C. Hudson, and D. Carruth, "Enabling off-road autonomous navigation-simulation of lidar in dense vegetation," *Electronics*, vol. 7, no. 9, p. 154, 2018.

[21] T. Yin, J. Qi, J.-P. Gastellu-Etchegorry, S. Wei, B. D. Cook, and D. C. Morton, "Gaussian decomposition of lidar waveform data simulated by dart," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 4300–4303, IEEE, 2018.

[22] B. Balaguer and S. Carpin, "Where Am I? A Simulated GPS Sensor for Outdoor Robotic Applications," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pp. 222–233, Springer, 2008.

[23] P. J. Durst and C. Goodin, "High fidelity modelling and simulation of inertial sensors commonly used by autonomous mobile robots," *World Journal of Modelling and Simulation*, vol. 8, no. 3, pp. 172–184, 2012.

[24] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[25] "Continental Mapping." `https://www.continentalmapping.com/`, 2019. Accessed: 2019-04-19.

[26] D. Negrut, R. Serban, A. Elmquist, J. Taves, A. Young, A. Tasora, and S. Benatti, "Enabling artificial intelligence studies in off-road mobility through physics-based simulation of multi-agent scenarios," in *Proceedings of Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, Aug. 2020.

[27] A. Elmquist, D. Hatch, R. Serban, D. Noyce, and D. Negrut, "Sensing simulation for the virtual testing of autonomous vehicle safety and performance," in *Proceedings of Road Safety and Simulation Conference*, Oct. 2019.

[28] A. Elmquist and D. Negrut, "Methods and models for simulating autonomous vehicle sensors," *IEEE Transactions on Intelligent Vehicles*, pp. 1–1, 2020.

[29] Project Chrono, "ProjectChrono API Web Page." `http://api.projectchrono.org/`. Accessed: 2017-10-20.