3D Mapping for high-fidelity unmanned ground vehicle lidar simulation

Brett Browning, Jean-Emmanuel Deschaud, David Prasser and Peter Rander

The International Journal of Robotics Research 2012 31: 1349
DOI: 10.1177/0278364912460288

The International Journal of
Robotics Research
31(12) 1349–1376

DOI: 10.1177/0278364912460288
ijr.sagepub.com

3D Mapping for high-fidelity unmanned ground vehicle lidar simulation

Brett Browning, Jean-Emmanuel Deschaud[1], David Prasser and Peter Rander

Abstract

High-fidelity simulation is a key enabling technology for the widespread deployment of large unmanned ground vehicles (UGVs). However, current approaches for lidar simulation leave much to be desired, particularly for scenes with vegetation. We introduce a novel 3D mapping technique that learns high-fidelity models for geo-specific lidar simulation directly from pose tagged lidar data. We introduce a novel stochastic, volumetric model that captures and can reproduce the statistical interactions of lidar with terrain. We show how to automatically learn the model directly from 3D mapping data collected by a UGV in the target environment. We extend our approach using terrain-classification techniques to develop a hybrid surface–volumetric model that combines the efficiency of surface modeling for areas that are well approximated by large surfaces (e.g. roads, bare earth) with our volumetric approach for more complex areas (e.g. bushes, trees) without sacrificing overall fidelity. We quantitatively compare the performance of our approach against more conventional methods on large outdoor datasets from urban and off-road environments. Our results show significant performance gains using our volumetric and hybrid approaches over the state-of-the-art, laying the ground work for truly high-fidelity simulation engines for UGVs.

Keywords

Mapping, mobile and distributed robotics SLAM, range sensing, sensing and perception computer vision, animation and simulation, simulation, interfaces and virtual reality, field robots, field and service robotics

1. Introduction

Unmanned ground vehicles (UGVs) have significant potential to be widely deployed in dirty, dull, and dangerous tasks in commercial and defense applications such as mining, agriculture, oil and gas, metal production and so on. A critical challenge, however, is establishing both qualitative and quantitative reliability and performance metrics. Unfortunately, field testing is notoriously expensive and is a significant barrier to entry. Given the incremental decision-making process of an autonomous robot and difficulty in controlling the conditions for large-scale outdoor field tests it can be intractable to perform statistically significant, repeatable testing. Safety and expense concerns also prevent potentially destructive testing that can be critical to understanding the boundaries of performance and reliability.

Simulation can augment field tests by providing a safe, low-cost mechanism for extensive repeatable and destructive testing. For example, in the aerospace industry new planes are tested significantly in simulation before validation field tests, thereby reducing time to deployment and greatly increasing confidence in the system (Georgiadis et al., 2008). Moreover, if a simulation environment mimics a specific test course in the real world, which we call geo-specific simulation,[2] simulation and field test results can further be corroborated providing greater evidence of system performance.

Unfortunately, current state-of-the-art simulation methods are either of insufficient fidelity or do not scale to the large terrains needed for UGVs in most applications (i.e. many square kilometers). Although there has been significant progress in the computer graphics field for simulating realistic imagery that captures artifacts such as scattering, diffuse illumination (Greenberg et al., 1997), and to a lesser degree vegetation (Tan et al., 2007), much work still remains. Similarly, for radar sensors, which are perhaps the most studied sensing system, there have been a number of efforts to realistically simulate the returns from

National Robotics Engineering Center, Carnegie Mellon University, USA

Corresponding author:
Brett Browning, National Robotics Engineering Center, Robotics Institute, Carnegie Mellon University, 10 40th Street, Pittsburgh, PA 15201, USA.
Email: brettb@cs.cmu.edu

vegetation and ground surfaces which are typically treated as clutter (Holtzman et al., 1978).

For laser-based range sensors such as lidar, the most commonly used sensor in UGV applications, the situation is less developed. Current robotics simulators (e.g. Koenig and Howard, 2004; Carpin et al., 2007; Quigley et al., 2009; Gonzalez et al., 2009) rely on surface models to represent terrain and simulate lidar using ray tracing or z-buffering, often with additive sensor range error (Goodin et al., 2009). The fidelity of this modeling and simulation approach depends greatly on the environment. Indoor, underground, and urban environments often work well because they primarily contain relatively large, solid objects. In contrast, off-road terrain is a greater challenge because it often contains large amounts of vegetation in which the widths of objects (e.g. small tree branches, leaves, and grasses) are often much smaller than the beam width of the lidar. This situation is one example of the well-known mixed-pixel problem (Tuley et al., 2005): there is more than one 'correct' range value because the small geometries reflect only a portion of the beam, allowing other parts of the beam to propagate further and generate additional returns.

To demonstrate this behavior, we collected more than 1000 range scans of a small tree, shown in Figure 1, from a stationary lidar.[3] The curve on the left shows the normalized histogram of returns from a beam repeatedly hitting a relatively small portion of the tree of similar size to the laser spot. The curve shows a single clear peak well approximated by a single surface with additive noise. The graph on the right, on the other hand, shows data for a nearby beam angle for the same laser, which exhibits dramatically different behavior: it appears bimodal and asymmetric. The branches and main 'trunk' of the plant are smaller in diameter than the laser beam width, and neither spot corresponds to the main trunk. Instead, the response is due to the complicated interaction of the lidar beam with multiple surfaces in the tree and is reminiscent of the complex additive and subtractive interactions in radar returns on larger targets. In short, a simple additive sensor noise model alone is poorly suited to capturing such large variation.

This problem can be overcome by using multi-sample ray tracing methods to model the non-zero laser beam width and by building geometric models at the scale of the blades of grass (Goodin et al., 2009), which does generate more realistic behavior in mixed-pixel scenarios. Automated tools (e.g. http://www.bionatics.com/) can even simplify the process of constructing sophisticated geometries of real-world objects. Of course the models grow dramatically – what might have been just a few polygons in a solid-ground simulation could become millions of polygons per cubic meter making simulation times slow or unwieldy – but that may be acceptable if high-performance computing systems are available. The greater challenge is the effort required to make the virtual model match the real world. Many semi-automated techniques can reconstruct large-scale models of building exteriors, roads, and other larger solid objects but these techniques perform poorly when attempting to reconstruct sub-pixel geometries of something as mundane – and common – as knee-high grass (see Section 6).

We propose an alternative model that extends 3D mapping to build a data-driven model for simulation that efficiently accounts for the variability of the scene and can be automatically estimated from measured lidar data. Our approach represents the world as volumes with a stochastic model for range returns and permeability which together captures the statistical distribution of the lidar range directly into the model without recovering surfaces. Range noise and mixed-pixel effects are both built into the model without the need to explicitly recover them. This model is substantially simpler to estimate and lends itself to straightforward simulation using ray tracing. The simulation curves of Figure 1 (blue in the color version, dark in the print version) are histograms of 150,000 simulated range returns striking the two same parts of the tree used to generate the real histograms (red/light), demonstrating that our approach can reproduce the statistics of real data to a high degree of accuracy. We introduce a set of techniques that enable stochastic volumetric models to be learned from large-scale datasets, and further show that our methods scale efficiently to large terrains. For simple surfaces, such as roads, this volumetric approach may not be warranted and the simpler, surface mesh representation may perform as well and with more efficient use of memory and computation. Thus, we extend our model representation and construction tools to a hybrid model that automatically creates surface models where they suffice and uses volumetric models where needed.

To validate our work, as sufficiently accurate ground truth is unavailable, we develop a set of machine learning inspired quantitative performance metrics. We evaluate our volumetric and hybrid models along with the more traditional surface models on a range of real-world datasets drawn from off-road and urban environments.

Thus, we make four contributions in this work.

- An automated 3D mapping system to construct models of geo-specific terrain for high-fidelity lidar simulation.
- A hybrid approach that uses volumetric models where needed and uses surface models where volumetric models offer no clear advantage.
- A range of quantitative metrics to validate the simulation approach on large real-world datasets where no explicit ground truth is available (as is typically the case).
- Quantitative evaluations of our approach and others found in the literature on a range of datasets using these metrics.

In Section 2 we review the broad set of techniques that have been developed for sensor simulation before describing our modeling and simulation techniques in Sections 3 and 4, respectively. We then present our evaluation metrics

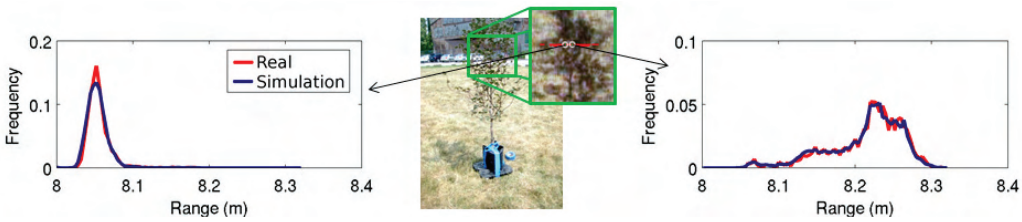


Fig. 1. Normalized histograms of lidar range measurements on two different points of a tree. Approximately 1500 range returns were used to generate each real curve (red in the color version, light in the print version). The blue/dark line histograms were produced from 150,000 simulated range returns using our volumetric model, described later. The inset image, taken from a co-calibrated camera nearly co-located with the lidar, shows the laser scan path and the two markers show the beam angles selected.

in Section 5 and the performance evaluations in Section 6. Finally, we conclude the work in Section 7.

2. Related work

Lidar range sensing is a topic that has been studied in a number of different areas including robotics, computer vision, graphics, and remote sensing. Lidar has been most heavily studied in the field of remote sensing, with extensive studies on atmospheric propagation, scattering phenomenon, and interaction with vegetation, primarily driven by its usefulness for aerial measurements (e.g. Jones and Vaughan, 2010).

Simulating a pulsed, time-of-flight lidar requires modeling of the physical generation and propagation of the light pulse out of the sensor, the propagation of laser light through the atmosphere, the object–terrain interaction leading to reflection or back scatter, the return journey back into the sensor, and the physical mechanism for sampling and reporting the detected range. At every step of this process, unmodeled or poorly calibrated deviations can occur that lead to errors in the sensed range. Mallet and Bretar (2009) have studied the physical sensor errors, which can be simulated as in Kukko and Hyyppa (2009). The physics of atmospheric propagation, absorption, and scattering are reasonably well understood (Hulst, 1981), and empirical data for some robot sensors exists (Ryde and Hillier, 2009; Dima et al., 2011).

In this paper, we focus on the interaction of lidar with the surfaces in the terrain. Ignoring atmospheric effects, for surfaces with approximately Lambertian reflectance and with surface geometries much larger than the lidar beam width, the range error is dominated by the limits of the physical sensor and are small and approximately Gaussian. As a result, representing the surface as a triangular mesh and performing ray tracing (or z-buffering), with optional additive Gaussian noise, produces a reasonable approximation of reality. This approach is commonly used in mainstream robot simulation environments (Koenig and Howard, 2004; Carpin et al., 2007; Gonzalez et al., 2009), and performs well for materials such as bare soil, dry cement, brick walls, and so on. Straightforward extensions to the ray tracing approach can model highly specular (e.g. quartz, metal) or transparent (e.g. water, glass) surfaces. However, the approach fails to capture the intricate behavior of vegetation.

Lidar interaction with vegetation is complicated due to the complex, multi-path reflection, absorption, and transmission that happens within plant cells [e.g. in the leaf cell structure (Jones and Vaughan, 2010)]. Furthermore, the dense geometries of plant surfaces create additional complex multi-path reflections. This latter effect can be seen on any surface boundary, as shown by Tuley et al. (2005) who demonstrated that lidar beams striking two or more surfaces at different depths can produce one or more returns that in some cases may not correspond to the range of either surface, an effect they called "mixed-pixel returns". For vegetation, with its complicated, and small surface geometries that are on the scale of lidar beam width, mixed-pixel returns can dominate causing non-trivial range distributions as shown in Figure 1.

Simple approaches to compensating for this behavior, such as modifying the additive noise model, do not work as they do not correctly capture the angular dependence of the returns. Using fine grained triangular meshes combined with dithered multi-ray, ray traced simulations can approximate the behavior (Goodin et al., 2009) but at substantial computational cost. Consider for example, the amount of memory and computing resources required to simulate $100\,\mathrm{km}^2$ of wild grassland. Estimating roughly 46×10^3 blades of grass per square foot[4] and say 20 triangles per blade of grass, would give roughly 10^6 triangles per square foot which is roughly 10^7 per square meter and a massive 10^{15} for $100\,\mathrm{km}^2$.

A second limitation is the generation of the model. For generic, non-geo-specific models, one can use approaches such as the Lindenmayer-systems (L-systems) (Lindenmayer, 1968) to generate realistic looking vegetation. However, while there has been work to estimate a tree's large-scale structure from lidar data (Livny et al., 2010), there are no techniques that would allow one to tune an L-system virtual tree to match real data from a specific location (i.e. a geo-specific scene). Secondly, approaches like L-systems do not approach the diversity of leaf structure found in real terrains and cannot capture the complex

reflectance properties of real leaves which are often laser wavelength dependent.

In this work, we attempt to fill this niche by learning models for lidar simulation from collected data in a terrain that capture the complexities of vegetation and yet still represent 'simple' surfaces well. We describe our approach in the following sections.

3. Scene modeling from 3D mapping

We use a data-driven approach to scene modeling for lidar simulation that extends from the surface representation often used in robot 3D mapping and perception systems. We first describe our overall approach and fundamental assumptions and then describe the different models that we have developed. Section 4 details how to draw samples from each of these models.

3.1. Data-driven modeling overview

Geo-specific modeling for lidar simulation requires constructing a model that captures how the sensor interacts with the vehicle, atmosphere and terrain for the target environment. Such a model could be built manually from real measurements, but is very time consuming to do so and would be of limited accuracy particularly for vegetation scenes. Instead, we pursue an automatic model building approach that uses real sensor measurements from the target environment. That is, we aim to automatically build a model from globally registered data collected by a vehicle equipped with a calibrated lidar sensor with a good pose estimation system (e.g. a high-quality INS) that is driven around the target environment.[5] Figure 2 shows a schematic that overviews the modeling and simulation process.

In more detail, we assume training data is a set of beams with origin, direction and range and that we can convert this to raw 3D points as needed (for beams with ranges less than the maximum range of the sensor). For efficiency, we often directly store the 3D point that corresponds to the end of the ray. In our logging system, we also colorize each point based on its appearance in a color camera. We utilize this color only for display purposes in this paper. It is a simple extension to our approach to include point color, however, and will be described in other work. Figure 3 shows two examples of point cloud inputs.

Rather than pursuing a pure data-driven approach where all sources of sensing behavior are encoded in the sensor–terrain interaction model, we break our approach up into different model components derived from the physics of the sensing process (this is similar to Glennie, 2007). Figure 4 shows a simplified schematic of the lidar sensing process that we use to decompose the problem into pose, sensor, atmospheric propagation, and sensor–terrain interaction modeling.

Modeling pose is a well studied problem that has been explored elsewhere (Hong et al., 2005) and is beyond the scope of this work. We consider simplified sensor models that we describe more fully in Section 3.2. The physics of atmospheric propagation of electromagnetic radiation relating to scattering, scintering and beam divergence is also well understood (Hulst, 1981), although geo-specific empirical parameters may be challenging to identify. Empirical models for the impact of various inclement weather conditions, such as dust, have also been studied in a number of cases (Ryde and Hillier, 2009; Dima et al., 2011). Although atmospheric interactions are crucial to representing real-world environments, in this work we focus on sensor–terrain interaction and therefore ignore these effects. Our simulation approach, however, (see Section 4) is extendable and can include these atmospheric effects. The interaction of lidar with the physical terrain, which we call the sensor–terrain interaction, is complex and except in simplified scenarios, poorly understood. Representing the sensor–terrain interaction is therefore the core component of this paper and is discussed in Section 3.3.

3.2. Sensor model

For UGVs, most lidar systems consist of one or more pulsed infrared lasers, typically in the 900–950 nm range although other wavelengths have been used, with one or more matched detectors. Real lasers have non-zero beam width that changes during the course of atmospheric propagation due to dispersion (Hulst, 1981) and/or beam focusing. Indeed, it is this non-zero beam width that causes complex return waveforms for objects with geometries approaching the scale of the beam width. These complex return waveforms translate into complex return ranges that are often called mixed-pixel returns (Tuley et al., 2005).

The sensor measures range by measuring the time between the emitted pulse and the 'pulse' of the backscattered light reflected from some remote surface(s). The sampling electronics that measures the time-of-flight for the pulse varies, and frequently the details are not made publicly available due to competitive reasons. Some systems return a single pulse, while others may return multiple pulses. The transit time of the pulse is converted to a range with knowledge of the speed of light as: $r = \frac{1}{2}cT$, where r is the range, c is the speed of light in the atmosphere,[6] and T is the measured travel time. The sampling electronics introduce quantization errors, sampling errors, and bias that produce errors in the resulting range measurement. For a simple surface (e.g. a plane), typical lidars produce range measurements that have an RMS deviation in the order of 1–30 mm. Absolute range accuracy depends on the calibration of the device, as well as relevant thermal conditions and whether the sensor compensates for these.

Typically the laser(s) and detector(s) are rotated, or point at an intervening mirror that is rotated, to produce a *scan line* (a set of points in a plane) or a full *sweep* (a set of points spread around 360°). Here, we focus on the Velodyne HDL-64E sensor[7] (see Figure 5) and therefore consider sensor

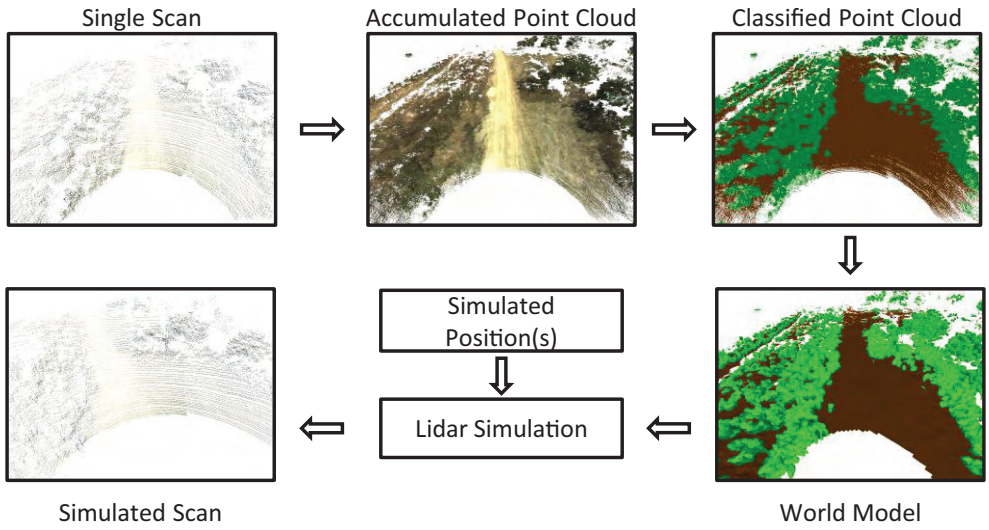
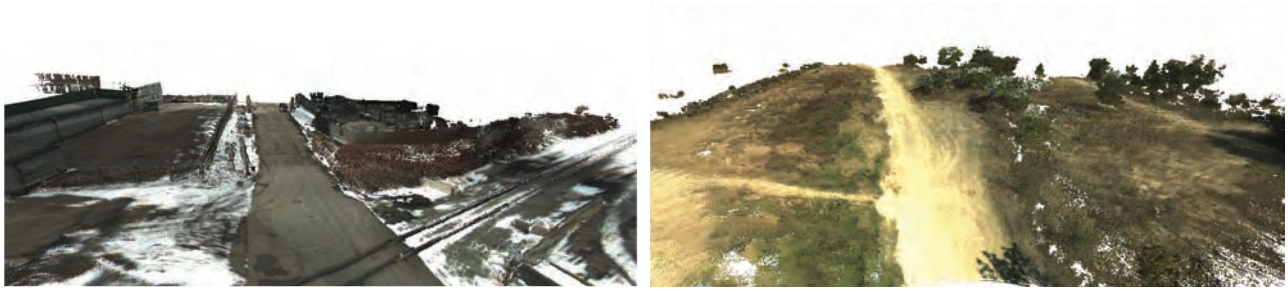


Fig. 2. Overview of the system. (Top left) Geo-registered scans of the target environment are captured and processed offline. (Top middle) Accumulating the data produces a geo-registered point cloud. (Top right) The point cloud may be processed via classification and optionally filtering to mark points that are surfaces versus non-surfaces. (Bottom right) The simulation world model is then built (Volumetric, Surface, or Hybrid). (Bottom middle) Returns from the sensor at novel poses can then be simulated using the world model to produce simulated scans for autonomy systems or visualization (bottom left).



(a) Urban scene (b) Off-road scene

Fig. 3. The two main datasets used for the moving platform results. (a) The raw input point clouds for an urban scene captured during the winter with some light snow accumulation (the white parts). (b) An off-road scene with trails and vegetation captured during the summer. Note, the white portions in the off-road scene are actually gaps in the point clouds due to occlusions and the white background is showing through.

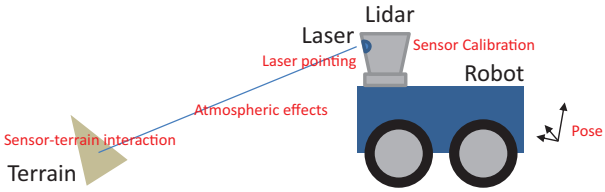


Fig. 4. Simplified schematic showing the main components of the physical lidar sensing process for a pulsed lidar such as the Velodyne HDL-64E.

sweeps through a full 360° rotation. Some devices employ additional rotation axes either in the device or externally (e.g. Thrun et al., 2003). In all cases, the angular orientation of the laser(s)/detector(s) when sampling is measured by encoders or similar devices. We use a simple abstracted representation that models the laser/detector as co-located devices. Each individual laser/detector has a position with respect to the vehicle, based on the calibration model of the sensor (i.e. the rigid body transform from the laser/detector to the vehicle reference point) and its current pointing angle. We do not model calibration error although this is a straightforward extension.

We have explored approaches that model pointing angle quantization error caused by encoders, non-zero beam width, and beam dispersion using a truncated cone with an

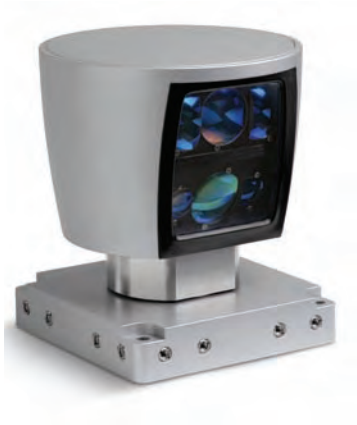


Fig. 5. The Velodyne HDL-64E lidar used in this work.

elliptical cross section. For the test results reported in Section 6, however, we disable the model quantization error, as the 'ground truth' data used for testing was captured with the same physical system. This means the ground truth data is imperfect and is corrupted by pose error, non-zero beam width and so on. Since we use the sensor pose to sample from the simulation model and compare it to the collected 'ground truth' data, simulating added pose/sensor noise will result in double counting the error. Thus, we disable it for those experiments and enable it for truly synthetic simulations.

3.3. The sensor–terrain interaction model

The core part of our approach is to model the sensor–terrain interaction. If we had a perfect model of the world, modeling the sensor–terrain interaction would still be non-trivial because the complexity of real-world, outdoor scenes produces very complex backscattered reflections. For example, most leaves are not a single reflecting surface but instead have a complex 3D structure that produces a complex backscattered waveform to the detector (Kim et al., 2010). For geo-specific modeling from data collected with a lidar, there is insufficient signal in the lidar returns to directly model the fine scale structure (i.e. we cannot observe where the leaves are or their detailed structure). Instead, we can only build a model that captures and reproduces the range return characteristics of the sensor–terrain interaction without modeling its fine-scale structure. To make matters worse, small changes in viewpoint lead to very different return distributions (see Figure 1). Such viewpoint changes are unavoidable even if the vehicle is in exactly the same position due to pointing and sampling errors. The end result is that we can only hope to build a model that captures the sensor–terrain interaction in a statistical sense, i.e. produces the same range sample probability distributions.

We consider three core types of representations to model sensor–terrain interactions: points, surfaces, and volumes. We use these core representations in the following models:

- A point-based model, which in essence is a nearest-neighbor representation.
- A surface model, without permeability, as commonly used in conventional simulators.
- A surface model with permeability that better captures behavior seen most with vegetation and chain-link fences.
- Our new stochastic volumetric model that uses regularly sampled voxels with embedded Gaussian distributions to better represent complex geometries such as vegetation.
- A variation on the stochastic volumetric approach that uses irregularly sampled Gaussians to better fit to the data, but at added computation cost to estimate.
- A hybrid approach that combines the benefits of both surface and volumetric representations.

Each of these models contain a stochastic component to simulate range uncertainty in the model. In some cases, the models also contain a notion of permeability. Real lidar beams typically penetrate foliage, and non-dense objects such as chain-link fences. While a very fine-grained model could properly capture this behavior, the results would be computationally demanding. Instead, augmenting the simpler geometry model with a notion of permeability (or transparency) allows us to encode this behavior without significant computational or estimation cost. Figure 6 shows a schematic of what permeability modeling aims to achieve. Figure 7 shows an example of the impact of permeability modeling on the resulting simulation[8]. We describe the details of the permeability approach for each model below.

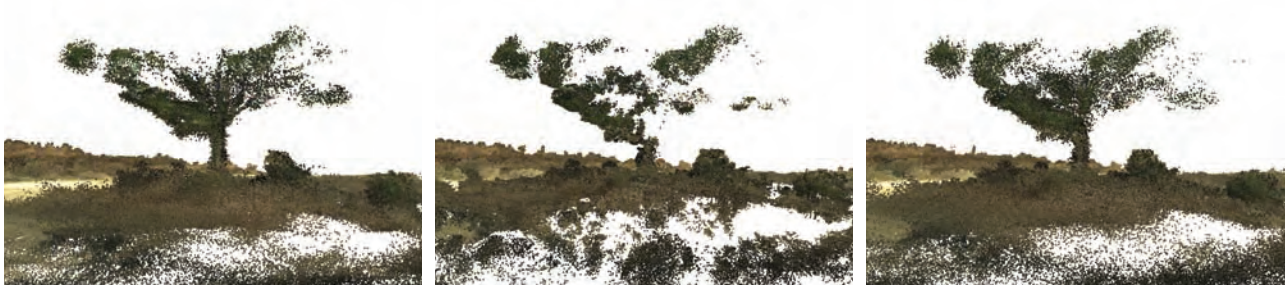
3.3.1. The point-based Nearest Neighbors model The simplest representation is a point-based model that is in essence a nearest-neighbor representation. In short, we accumulate points and the associated beam information from the raw data. These points directly become 'the model' used in the sampling process described in Section 4.2. The internal representation is simply the set of points, although in practice we use a voxel grid to provide fast data access to query for nearest neighbors when tracing simulated lidar beams through the model and store the estimate of the each point's normal to simplify the calculation of angle of incidence as described in Section 3.3.2. Estimation of this model simply means storing the observed points.

3.3.2. Surface model Surface modeling is the conventional approach used in game engines, and most robot simulators currently available. In practice, these models however are usually built by hand. We develop an approach to automated modeling that derives from many of the techniques explored in computer graphics and computer vision.

The core representation is a 3D triangle mesh, although a polygon mesh or soup could also be used. Each triangle in the mesh represents an observed surface in the environment as discovered in the data. To estimate the surface mesh

(a) Environment (b) Input point cloud (c) Estimated model and simulation

Fig. 6. The different modeling and simulation techniques. (a) A hypothetical environment consisting of two trees, a chain-link fence and two walls. The vehicle has lidar scans in a horizontal plane. (b) A synthetic point cloud generated by the lidar from the position in (a). (c) A 2D schematic of the models presented in this paper. The lines in (a) and (c) show some example simulated lidar beams generated from a new location. Parallel to the beams are indicators of the simulated beams PDFs.



(a) Real data to simulate (b) Simulated data without permeability (c) Simulated data with permeability

Fig. 7. Permeability has a tremendous impact on vegetated scenes. (a) The raw data captured at a novel pose position of a tree in the off-road scene. The vehicle was located off to the left of the scene. (b) The simulated data from the same pose with no permeability modeling. (c) The simulated data with permeability enabled. Clearly, permeability allows partially occluded parts of the tree at the rear (to the right) to be sensed as they are in reality.

from a point cloud we robustly estimate an implicit surface on a regularly sampled voxel grid and then turn this into a triangle mesh using fast marching cubes. Specifically, we use the Robust Implicit Moving Least Squares (RIMLS) algorithm (Öztireli et al., 2009) with the marching cubes technique of Lorensen and Cline (1987). RIMLS requires an estimate of the local surface normal for each data point. We estimate this using a weighted, local robust plane fit. To limit the extent of the surface we only evaluate the implicit function on grid nodes with at least three neighbor points. We call this approach the *Surface* model.

One problem with this approach is that for vegetation and objects such as chain-link fences, it fails to capture the ability of the lidar to pass through the object. We extend this approach with a simple permeability model to capture this property *without* necessitating a much finer grained model.

Although one could conceivably create a Bi-directional Reflectance Distribution Function (BRDF) (Nicodemus, 1965) and transmittance function (BTDF), it is difficult to see how to practically measure these values in real-world outdoor scenes at scale. Instead, we introduce a simple model whereby each triangle in the model has an associated permeability. The permeability is a Bernoulli distribution that captures the probability of the ray to pass through the triangle rather than 'hitting' it and backscattering to the lidar. That is, each triangle t_i has an associated parameter $\rho_i \in [0, 1]$, where ρ_i is the probability of a ray passing through the triangle.

We estimate the permeability parameter ρ_i for each triangle by counting how many beams generate a hit near the triangle, N_h, and how many miss or pass through the triangle, N_m. Hits and misses are determined by comparing the measured range of each beam, r, and the distance along the ray, d, at which the ray intersects the triangle. A hit is counted when $|r - d| < \Delta_r$ and a miss when $r - d > \Delta_r$. For this calculation a lidar beam with no return is considered to have $r = \infty$. The parameter Δ_r accounts for differences due to sensor noise and model construction error. The permeability is then evaluated with: $\rho_i = \frac{N_\mathrm{m}}{N_\mathrm{h}+N_\mathrm{m}}$.

The surface model also requires an estimate of the sensor noise produced by the surface. A simple model of the sensor noise is an additive Gaussian noise term, with a single value for the standard deviation, σ, which is constant across the entire model. In practice, the range return variance increases based on the angle of incidence. This has been studied for airborne operations (Gardner, 1992), and here we use the formulation by Baltsavias (1999) for a

pulsed laser. Collecting terms and simplifying, the impact of the angle of incidence on lidar range variance is:

$$\sigma^2 = \sigma_0^2 + \sigma_a^2 \left[\frac{\sin\theta}{\cos^2\theta} \right]^2 \quad (1)$$

where θ is the angle of incidence between the simulated beam and the local surface normal. For the triangle mesh surface representation, this is the triangle's normal. σ_0 defines the collected terms that define the sensor range return variance for a perpendicular surface $\theta = 0^\circ$. σ_a collects the terms defining the effect of increasing angle of incidence, which we treat as constant (this is not strictly true but is reasonable to a first-order approximation). Note, our use of σ_0 and σ_a differ from that of Baltsavias (1999) as we learn these parameters from the training data and therefore collapse the extra terms together.

Rather than measuring the parameters σ_0 and σ_a, which may be difficult in practice, we directly estimate these from the training data. We estimate these terms to fit the residual difference between the ranges in the training data and the local surface constructed from the training data. Note, we restrict this test to those parts of the model that are classified as a surface (see Section 3.3.4). Once estimated, these properties of the model are held fixed across environments.

3.3.3. The stochastic volumetric model Our stochastic volumetric model is motivated by the observation that for roads and other large-geometry surfaces, points tend to cluster along the surface. For vegetation and other terrain features that have many small geometries, the returns instead are spread throughout the *volume* bounded by the outer extents of the terrain geometry. Figure 1 demonstrated this effect with a 1D slice of this volume. Second, we observe that the return ranges vary considerably with very small changes in sensor pointing angle. As a result, with sensor and pose uncertainty, we can only hope to simulate the characteristics of the sensor–terrain interaction in a statistical sense. Thus, our proposed model tries to capture the volumetric nature of the data, along with the statistical properties of the range returns.

Representation: Our approach represents the terrain using a set of volumetric elements, where each volume contains a probability distribution to encode the spatial distribution of the point returns therein. Although, we have explored a number of volumetric models, including voxels and tetrahedrons, the approach we describe here makes use of Gaussian distributions to represent the point densities, and also to encode the spatial extent of the volume. Although simple, it produces surprisingly good results.

We model a volumetric element v_i with a Gaussian distribution to describe the the 3D point distribution inside the volume. That is, points $\boldsymbol{q} \in \Re^3$ inside the volume v_i are distributed according to $\boldsymbol{q} \sim N(\boldsymbol{\mu}_i, \Sigma_i)$, where $N(.)$ is the normal distribution with mean $\boldsymbol{\mu}_i \in \Re^3$ and covariance $\Sigma_i \in S^3$. This approach is similar in spirit to the Normal Distribution Transform (NDT) used for point cloud registration (Magnusson et al., 2007).

The spatial extent of the volume is formed by an iso-surface at a fixed Mahalanobis distance. That is, the volumetric element is a 3D ellipse defined by $\{q | q \in \Re^3 \wedge d_{v_i}(q) < \tau\}$. Here, $d_{v_i}(q)$ is the Mahalanobis distance of point q defined for the Gaussian for volume element v_i as:

$$d_{v_i}(q) = \|\boldsymbol{q} - \boldsymbol{\mu}_i\|_{\Sigma_i} = \sqrt{(\boldsymbol{q} - \boldsymbol{\mu}_i)^T \, {\Sigma_i}^{-1} (\boldsymbol{q} - \boldsymbol{\mu}_i)} \quad (2)$$

The iso-surface is defined by the same threshold τ for all volume elements, which we typically set to 3.5 (with that value 99.7% of the distribution will be covered by the volume). To summarize, we represent each volume with a mean and covariance $(\boldsymbol{\mu}_i, \Sigma_i)$. The mean and covariance also define the spatial extent of the volume. This corresponds to $3 + 6 = 9$ parameters in total as the covariance matrix is symmetric.

We incorporate a permeability term for each volumetric element in a similar fashion to the surface model. As before, for a volumetric element v_i, we represent permeability as a Bernoulli distribution with parameter $\rho_i \in [0, 1]$. Thus, rays pass through the volume with a probability of ρ_i.

Estimating the volumetric model: To estimate the volumetric elements from data, we explore two approaches: partitioning with a regular voxel grid, and top-down hierarchical clustering.

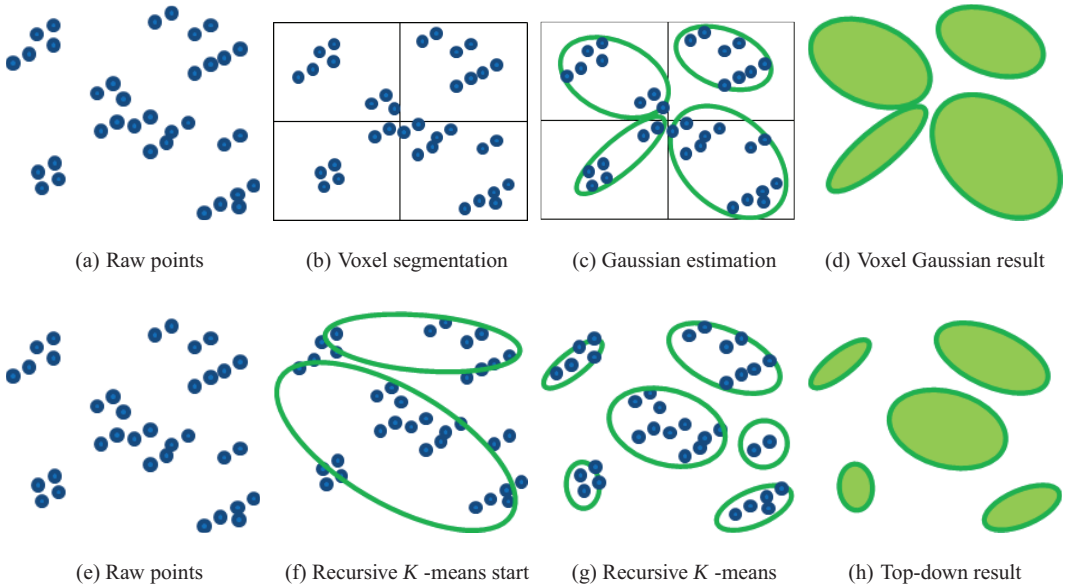
We call the first approach the *Voxel Gaussian* method. It takes the raw point cloud data and partitions it into regular voxels of side s. Within each voxel, a volumetric element is instantiated if there are sufficient points N_v. The volumetric element is then estimated using the standard maximum likelihood framework. For points $\boldsymbol{p}_1, ..., \boldsymbol{p}_N$ observed in the voxel, we calculate the volumetric element as:

$$\boldsymbol{\mu}_i = \frac{1}{N} \sum_k \boldsymbol{p}_k \quad (3)$$

$$\Sigma_i = \frac{1}{N} \sum_k (\boldsymbol{p}_k - \boldsymbol{\mu}_i)(\boldsymbol{p}_k - \boldsymbol{\mu}_i)^T \quad (4)$$

The upside of this approach is that it is extremely efficient to calculate and is highly parallelizable. The downside is that the voxel partitioning can introduce unwanted artifacts by not following the natural distribution of the data. Figure 8, top row, presents the different steps of the voxel segmentation.

Our second model overcomes this limitation by top down, hierarchical clustering. We call this the *Top Down* model. Following the approach of Kalaiah and Varshney (2005), we use hierarchical K-means to partition the data. That is, starting with all of the data, we run K-means to partition it into K-clusters. We then recursively repeat the process on each of the K-clusters. This process continues for each cluster until there are an insufficient number of points N_p in a cluster to warrant continuing.



(a) Raw points (b) Voxel segmentation (c) Gaussian estimation (d) Voxel Gaussian result

(e) Raw points (f) Recursive K -means start (g) Recursive K -means (h) Top-down result

Fig. 8. The two voxel estimation processes. The top row (a)–(d) shows the Voxel Gaussian method where points are partitioned into regular voxels (b), where a Gaussian is estimated via Maximum Likelihood in each voxel with sufficient points (c), producing the final Voxel Gaussians (d). The bottom row (e)–(h) shows the Top Down algorithm starting with the same points (e), which are recursively partitioned via K-means (f)–(g). A Gaussian is estimated for each leaf cluster resulting cluster using Maximum Likelihood producing the result (h).

The original work of Kalaiah and Varshney (2005) uses a simplified version of K-means, with $K = 2$. At each level, the two cluster means are initialized along the principal direction of the points and are then refined by assigning each point to the closest cluster and then updating the means with the new assignments. This two-means clustering is stopped after 30 iterations. We obtained better results by using a variant with $K = 10$ and the random partition method for initialization (Hamerly and Elkan, 2002). Cluster means are initialized by selecting random points. Cluster assignments and means are updated iteratively based on closest proximity using Euclidean distance. Contrary to Kalaiah and Varshney (2005), we track the mean positions that produce the minimum within-cluster variation and use multiple restarts to ensure that we have not converged to a local optima. When a cluster ends up with no assigned data, we drop the class. As above, clustering is then recursive and terminates when there is an insufficient number of points N_p. This is the *Top Down* model we use in the remainder of this paper. Figure 8, bottom row, presents the different steps of the Top Down segmentation.

Once the volumetric elements are estimated, we calculate permeability in a similar way to to the surface permeability described above (Section 3.3.2). That is, we count the number of beams that pass through the volume, as defined above, but do not terminate inside the volume (N_m). Similarly, we count the number of beams that terminate inside the volume (N_h). To efficiently evaluate whether a beam passes through the volume, we calculate the minimum Mahalanobis distance between the ray and the ellipsoid (see Figure 9). If this distance is closer than the iso-surface threshold defined above, then it passes through the volume.

Consider the ray $\boldsymbol{p} = \boldsymbol{p}_0 + t\hat{\boldsymbol{r}}$, with $t > 0$ and $\boldsymbol{p}_0$ is the ray origin (the sensor) and $\hat{\boldsymbol{r}}$ is the unit normalized pointing direction of the ray. The closest point on the ray, in terms of the Mahalanobis distance to a volumetric element with mean $\boldsymbol{\mu}_i$ and covariance Σ_i is:

$$t_\mathrm{min} = \arg\min_t \|(\boldsymbol{p}-\boldsymbol{\mu})^T \Sigma_i^{-1}(\boldsymbol{p}-\boldsymbol{\mu})\| \quad \text{subject to} \quad \boldsymbol{p} = \boldsymbol{p}_0 + t\hat{\boldsymbol{r}} \tag{5}$$

It is straightforward to show that this has a solution in closed form given by:

$$t_\mathrm{min} = \left[\hat{\boldsymbol{r}}^T \Sigma_i^{-1} \hat{\boldsymbol{r}}\right]^{-1} \hat{\boldsymbol{r}}^T \Sigma_i^{-1}(\boldsymbol{\mu}_i - \boldsymbol{p}_0) \tag{6}$$

As before, we estimate permeability as $\rho = \frac{N_\mathrm{m}}{N_\mathrm{h}+N_\mathrm{m}}$.

3.3.4. The hybrid surface and volumetric model As will become clear from the experimental results, surface models are compact and are good at representing terrain that is approximated well by a coarse grained triangle mesh (e.g. roadways). Our volumetric model outperforms the surface model on all other terrains, and comes close to the performance of surface models on large surfaces, but requires more memory and computational resources. We therefore introduce a hybrid model that combines both surface and volumetric elements using the same representations defined above.

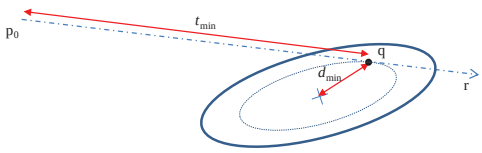


Fig. 9. Schematic showing a 2D version of the lidar ray and Gaussian volume intersection. The lidar ray originating from p_0 and propagating along direction r has the minimum Mahalanobis distance ($d_{\min}$) to the Gaussian volume at the point q. This range for this point q is the distance along the ray $t_{\min}$ defined in equation (6). The minimum Mahalanobis distance iso-contour is shown as the thin ellipse. The thick ellipse shows the iso-contour that forms the boundary of the volume.

To build the model, we first segment the point cloud into surface and volumetric datasets. We identify these using conventional 3D point cloud classification techniques, such as the terrain classification method described in Tuley et al. (2005) and Lalonde et al. (2006). This approach is straightforward, but is likely sub-optimal as terrain classification works using an arbitrary, user-defined notion of surface versus non-surface and does not take into account how well a particular part of the terrain can be modeled. We use this technique, but will investigate more powerful and appropriate approaches in future work. For the Hybrid model, we only require two model classes making the problem simpler compared to the full terrain-classification problem.

We form a shape descriptor for each point calculated by first performing PCA on the 3D positions of the local scatter matrix for all points within a fixed distance, d, of the target point. The ratio of the Eigenvalues, provides an estimate of how spherical the local neighborhood is. The spherical variation is calculated from the sorted Eigenvalues, $s = \frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$, with $\lambda_1 \leq \lambda_2 \leq \lambda_3$. s has a maximum value of $\frac{1}{3}$ when the local point cloud has a spherical shape. For points with too few neighbors, calculating the Eigenvalues leads to very noisy estimates for s. That is, small changes in the points lead to drastically different values for s. In these cases, it is unreliable to estimate the shape and instead we utilize a *default* value for s, which we term s_{d}. The minimum number of neighbors is set to $N_{\min}$.

To further smooth the data, without resorting to a more complex classification method such as Conditional Random Fields or learned variants (Munoz et al., 2009), we use an edge-preserving bilateral filter (Tomasi and Manduchi, 1998) over the initial spherical variation to produce a final, smoothed, score for each point, s'. Our filter is similar in spirit to the mesh denoising approach of Fleishman et al. (2003) except we filter the sphericalness scores not the points themselves. The bilateral filter smooths over local sphericalness scores using two Gaussian weighting functions; one a function of Euclidean distance and the other a function of score similarity. Thus, the smooth score s'_i is calculated over the local neighborhood as:

$$s'_i = \frac{\sum_j w_{ij} s_j}{\sum_j w_{ij}} \tag{7}$$

Where the weight w_{ij} for point j with respect to the point being smoothed i, is a Gaussian weighting of the Euclidean distance and score similarity:

$$w_{ij} = \exp\left(-\alpha_{\mathrm{d}} \frac{|\boldsymbol{q}_i - \boldsymbol{q}_j|^2}{d_{\mathrm{f}}^2} - \alpha_{\mathrm{s}} \frac{(s_i - s_j)^2}{(1/3)^2}\right) \tag{8}$$

Here, q_u is the 3D location for the u^{th} point and s_u is its sphericalness score. α_{d} is the distance weighting factor and α_{s} is the score similarity weighting. These are fixed and are global. We normalize scores by the maximum sphericalness value $\frac{1}{3}$. Similarly, we normalize distances by a constant d_{f}. As the Gaussian has infinite support, for practical speed reasons we limit the neighborhood to the nearest N_{f} neighbors that are also within a distance threshold d_{f}. That is, we consider all points q_j that are in the N_{f} nearest neighbors of q_i and $|q_i - q_j| < d_{\mathrm{f}}$. Rather than trying to manually set the above parameters: $s_{\mathrm{d}}, N_{\min}, N_{\mathrm{f}}, d_{\mathrm{f}}, \alpha_{\mathrm{d}}, \alpha_{\mathrm{s}}, s_{\mathrm{vol}}$, we learn these from a hand-labeled dataset. We use differential evolution (Storn and Price, 1997) to train the parameters using the labeled data due to its ability to handle both continuous and discrete parameters.

Once smoothed, the score is then compared to a threshold s_{vol} to determine whether the point belongs to a surface or a volume. volumetric and surface models are then estimated for each respective set of points using the techniques described previously. Figure 10 shows an example of the Hybrid model, with permeability disabled.

This approach produces reasonable results, as we show in the results section. The keen reader will note however, that the classification objective is completely unrelated to the ability to *simulate* the target terrain. The ideal approach would be to classify points based on whether a surface or volumetric model would produce a more accurate simulation. There may be vegetation, for example, that is best modeled as a surface or regions of the ground better modeled as a volume. A more domain-specific classification method is an area for future work. The results presented in Section 6, however, indicate that any such errors are relatively small but more work is required to quantify this.

3.4. Summary of models

Table 1 summarizes the models we presented above, and their training methods.[9]

4. Lidar simulation

We now describe the simulation process for drawing lidar range return samples from each of the different models. We first provide an outline of the process which is generally

(a) Input data **(b)** Classified points **(c)** Hybrid model

Fig. 10. A rendering example of the Hybrid model. (a) The raw point cloud for a scene with a relatively flat, bare earth ground surface surrounded by vegetation. (b) The output of the point cloud classification and filtering process. Brown (dark) points are surface points while green (light) points are non-surface points. The surface points are used to estimate a single triangle mesh surface model, while the non-surface points are used to estimate a volumetric model. (c) A rendering of the triangle mesh in brown (dark) and the iso-contours for the volumes as green (light) ellipses.

Table 1. Summary of sensor–terrain interaction models (M.L. = Maximum Likelihood).

Model name	Core representation	Training method
Nearest Neighbors	Points	Raw data
Surface	Triangle mesh	RIMLS, marching cubes
Permeable Surface	Triangle mesh + permeability	RIMLS, marching cubes, permeability
Voxel	Gaussians + permeability	Regular voxel partition, M.L. Gaussian, permeability
Top Down	Gaussians + permeability	Hierarchical K-means, M.L. Gaussian, permeability
Hybrid	PermSurface + Top Down	Classification surface/volume, Permeable Surface, Top Down

common across models. We then detail each specific model and discuss some of the implementation issues.

```
1  def simulateLidar(pose,sensor,model):
2    # Input: Vehicle pose, sensor model, sensor-
         terrain interaction
3    # Output: Lidar ranges
4    sensor.sampleSensorPose(pose)
5    foreach lidar beam b:
6      # generate the ray traced beam from the
           sensor model
7      (p0,r)=sensor.sampleRay(pose,b)
8      # Repeat propagation until we get a hit or
           exceed the max range
9      t=0
10     range_b = ∞
11     while !hit and t<sensor.max_range:
12       # Find next object that hits
13       (m,t)=rayTrace(p0,r,model)
14       if finite(t):
15         hit = m.samplePermeability(p0,r)
16         if hit:
17           # Sample the range from the model
18           range_b=m.sampleRange(p0,r)
```

Listing 1. Basic ray tracing algorithm for lidar simulation.

4.1. Simulation overview

For simulation, we assume we are given the *true* pose of the vehicle and the intrinsic and extrinsic calibration and position (e.g. pointing angle) for the sensor. These of course may be quite different from what is reported to the vehicle perception and autonomy system.

For all models, we use the same basic framework: given a vehicle pose and laser/detector position, we sample a ray from the sensor model and then ray trace the trajectory of the ray through the sensor–terrain interaction model to find intersecting geometries. For each intersecting geometry, we evaluate whether the ray 'hit' the geometry. For models without permeability, this is trivial. For models with permeability, we must draw a sample from the Bernoulli distribution to determine whether the ray 'hits' or 'misses'. If it misses, we continue to the next intersecting geometry in turn. If it hits, then we draw a range sample in a model-dependent way. Listing 1 gives the basic algorithm.

We use a ray tracing approach primarily because of the presence of permeable structures, which negates the potential performance improvements offered by z-buffering techniques (Friedmann et al., 2010). We note however that with the prevalence of high-end Graphical Processing Units (GPUs) that support real-time ray tracing (e.g. with NVIDIA's Optix engine[10]), ray tracing can be done in real-time and it is highly parallelizable.

We now describe the details of how to draw a sample from each model type.

4.2. Nearest neighbors

There are multiple ways one could conceive of drawing samples from the nearest-neighbor representation. One method, similar in spirit to light field techniques (Levoy

and Hanrahan, 1996), is to find the *nearest* observed lidar beam. If the beam range was a no-return, then we report the same result. If the beam range was a successful return, then the distance between the 'hit' point and the simulated sensor projected onto the simulated ray is the sampled range return. Although correct in principle, this approach requires dense sampling of the space of possible lidar rays in the environment to be of practical use. Any significant deviation in vehicle position is likely to lead to lidar beams that are very distant from the model beams and are therefore unlikely to perform well.

A variation on this approach that does perform well, is to find the nearest *point* to the simulated lidar beam, and to use the distance from the sensor to the point, projected onto the ray, as the range return. Here, 'closest' means the point that is closer in terms of distance from the line segment that forms the beam. One issue with this approach is that, for an arbitrary ray, we can always find a closest point. Consider the case of a simulated lidar beam aimed at the 'sky' in the general vicinity of a tree. The closest point will belong to the tree and so the simulated tree will effectively grow to fill the sky region. To address this, we introduce a maximum threshold $d_{\max}$ on the distance from any point to the ray. Points that are outside this distance are not considered, and in practice this threshold is quite small.

For a sampled lidar beam $(\boldsymbol{p}_0, \boldsymbol{r})$, we find the nearest point by first projecting each point onto the ray and using that distance to find the closest point to the ray. For a point in the training data $\boldsymbol{q}_i$, as shown in Figure 11, $\boldsymbol{x}_i$ is the point on the ray that is closest to $\boldsymbol{q}_i$. The distance t_i along the ray to the point $\boldsymbol{x}_i$ is:

$$t_i = \boldsymbol{r} \cdot (\boldsymbol{q}_i - \boldsymbol{p}_0) \tag{9}$$

The point $\boldsymbol{x}_i$ is just $\boldsymbol{x}_i = \boldsymbol{p}_0 + t_i \boldsymbol{r}$, and the perpendicular distance is then:

$$d_i = \left\| \boldsymbol{p}_0 + t_i \boldsymbol{r} - \boldsymbol{q}_i \right\| \tag{10}$$

The simulation operation therefore returns the range for the minimum d_i subject to the constraint that d_i is less than the pre-defined threshold $d_{\max}$ and the range t_i is less than the sensor's maximum sensing range ($t_{\max}$). Thus:

$$\begin{aligned} \boldsymbol{q}^* &= \arg\min_i \left\| \boldsymbol{p}_0 + t_i \boldsymbol{r} - \boldsymbol{q}_i \right\|, \\ \forall\, i \text{ s.t. } & d_i < d_{\max}, \quad \text{and} \quad t_i < t_{\max} \end{aligned} \tag{11}$$

We simulate the same range noise as the Surface model by adding the additive Gaussian noise defined by equation (1) with the same parameters. The angle of incidence, θ, is the angle between the beam and the normal of the nearest point, which is computed from its local neighborhood with a kernel width of 0.4 m.

4.3. Sampling from surface and permeable surface models

For surface models, we use the conventional approach of ray tracing the beam until it intersects a triangle in the surface mesh. For permeable surface models, we draw a Bernoulli sample to determine if the beam intersects the triangle (a 'hit') or instead 'passes through'. If the ray passes through, then we continue tracing the ray until it 'hits' another triangle or passes beyond the maximum range of the sensor. If a hit is generated then the reported range of the beam is the distance at which it intersects the triangle. We add Gaussian noise to capture the range error induced by the lidar range measurement process.

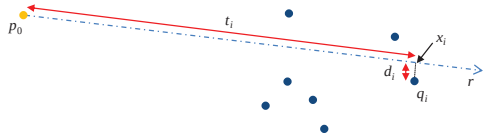


Fig. 11. Schematic of the nearest-neighbor sampling process. The ray travels from $\boldsymbol{p}_0$ along $\boldsymbol{r}$. $\boldsymbol{x}_i$ is the point on the ray closest to $\boldsymbol{q}_i$ and d_i is the perpendicular distance from $\boldsymbol{q}_i$ to the ray. The range returned is the distance from $\boldsymbol{p}_0$ to $\boldsymbol{x}_i$ for the point $\boldsymbol{q}_i$ where d_i is the smallest. The result is defined only if the perpendicular distance d_i is less than a threshold $d_{\max}$ and the range is less than the sensor range $t_{\max}$.

4.4. Sampling from volumetric models

For volumetric models, the simulated beam is ray traced to find the intersecting volume elements. We described earlier how to determine if the volumetric element is pierced by a ray (see Section 3.3.3). For each intersecting Gaussian we sample the Bernoulli distribution encoding the permeability to determine if it is a 'hit' in exactly the same way as for the surface models. Once a 'hit' is found, say with volume v_i, we draw a range sample from the Gaussian distribution $N(\boldsymbol{\mu}_i, \Sigma_i)$ *constrained* to be on the ray $\boldsymbol{p} = \boldsymbol{p}_0 + t\hat{\boldsymbol{r}}$. Substituting $\boldsymbol{p}$ into the Gaussian distribution and simplifying leads to a univariate Gaussian distribution for the range $t \sim N(\mu_t, \sigma_t^2)$, where:

$$\mu_t = \hat{\boldsymbol{r}}^T \Sigma_i^{-1} (\boldsymbol{\mu}_i - \boldsymbol{p}_0)\, \sigma_t^2 \tag{12}$$

$$\sigma_t^2 = \left[\hat{\boldsymbol{r}}^T \Sigma_i^{-1} \hat{\boldsymbol{r}} \right]^{-1} \tag{13}$$

We draw a sample from this univariate Gaussian distribution to produce the reported range. Unlike the Surface model we *do not* add additive range error. The range uncertainty is already encoded in the Gaussian point distribution and thus it is unnecessary. It is for this reason that we term the model a sensor–terrain interaction model.

4.5. Sampling from the Hybrid model

The Hybrid model really consists of a permeable surface model and a Top Down volumetric Gaussian model. As such, we sample from the model with a ray tracing approach where we test for intersecting geometry. For each

model–ray intersection, we sample from the appropriate model as described above. Because we process in monotonically increasing range, the shorter effective range value (depending upon sampling) is returned.

5. Simulation evaluation metrics

To evaluate the performance of the different models, we conduct a series of experiments where we build a model from one (or more) datasets in a given environment, which we refer to as the training datasets. We then collect real lidar returns from an independent run through the same environment, which we call the test data. We simulate lidar returns from the model under test using the poses from the independent run and compare these to the real lidar returns to assess the quality of each model.

Lidar sensing, however, is a noisy measurement process that is further compounded by mixed-pixel returns in the presence of small geometries, such as vegetation (e.g. see Figure 1). As a result, two successive measurements of a scene with the same device, from the same viewpoint, rarely return the same range values. Thus, for any high-fidelity, geo-specific modeling technique it is non-trivial to *quantitatively* evaluate how good the simulation method is. Qualitative comparisons are certainly useful, but do not carry the same weight of evidence.

For a given autonomy system, one approach is to compare the decisions a UGV will make in simulation versus in the real world when it is in the same location. Somewhat equivalently, for UGVs that plan over sensor-derived cost maps [e.g. Kelly et al. (2006) and Stentz et al. (2007)], we can compare the cost maps that are produced by the UGV perception system (Durst et al., 2011). However, this approach does not generalize to different autonomy systems since they may perceive the world at different levels of detail. Instead, we introduce a set of novel metrics to establish quantitative performance measures in a statistical sense.

5.1. Comparing range histograms

Different quantitative metrics can be conceived. A simple starting point would be to directly compare the range returns for each simulated lidar beam and its real lidar beam counterpart, which we call beam-to-beam comparisons. Unfortunately, since range returns can naturally vary quite considerably (see Figure 1), directly comparing range returns is likely to be arbitrarily bad even for comparing the same sensor in the same location. A better alternative is to take many samples for each pointing direction (e.g. having the vehicle stationary) and to compare the resulting histograms using a histogram or probability density function (pdf) distance metric. We need to take special care to account for the no-return scenarios. For histogram comparisons, we can achieve this by adding another bin specifically for no-return data.

To quantitatively compare histograms, there are a number of metrics that are possible. We use the Bhattacharyya distance metric (Bhattacharyya, 1943). Specifically, for normalized histograms $H = (h_1, ..., h_N)$ and $F = (f_1, ..., f_N)$ with N bins and $\sum_i h_i = 1$, $\sum_j f_i = 1$, we have:

$$D_{\mathrm{B}}(H,F) = -\log \sum_i \sqrt{h_i f_i} \tag{14}$$

The Bhattacharyya distance is always zero or positive ($D_{\mathrm{B}}(H,F) \geq 0$) and takes on a minimal value of zero for two perfectly matching histograms. For perfectly non-matching distributions (e.g. consider a point mass at different values in the two distributions so that $f_i > 0 \Rightarrow h_i = 0$ and $h_i > 0 \Rightarrow f_i = 0$), we get $f_i h_i = 0, \ \forall i$ producing a Bhattacharyya distance of infinity (∞). Thus, lower Bhattacharyya scores indicate a better histogram match and therefore that the two distributions are more likely to be the same.

The major limitation to this evaluation strategy is that it cannot be used for a moving platform as viewpoints will not be revisited. We now discuss an alternative metric for these scenarios.

5.2. Point cloud distance metrics

For mobile vehicles, we propose an alternative comparison: comparing the reconstructed point clouds. That is, for real (and simulation) lidar returns, we transform each range return into a 3D point in world coordinates (e.g. UTM). We then find the minimum distance between the real point cloud and the simulated one. While not strictly correct, as the histogram comparisons are, the comparison should give a measure of how similar the structure is. We consider two variations on this approach: comparing point clouds reconstructed from a single sweep (or scan), and comparing point clouds accumulated from a whole dataset.

More specifically, for two point sets $X = (x_1, ..., x_N)$ and $Y = (y_1, ..., y_M)$, where $x_i, y_j \in \Re^3$ are the 3D points. The point-cloud-to-point-cloud distance is defined as:

$$D'_{\mathrm{pp}}(X,Y) = \sum_i \min_j \|x_i - y_j\| \tag{15}$$

This is a non-symmetrical distance metric, so we use the worst-case distance:

$$D_{\mathrm{pp}}(X,Y) = \max\left(D'_{\mathrm{pp}}(X,Y), D'_{\mathrm{pp}}(Y,X)\right) \tag{16}$$

There are two different point set types that we consider. The first is to compare point clouds over a single sweep (a 360° rotation of the lidar) with 3D points projected into vehicle-centric coordinates. The idea being that *sweep-to-sweep comparisons* are relatively independent of vehicle pose estimates and are more robust to errors therein. The second type is to accumulate point clouds from a whole run and compare these, which we call *point-cloud-to-point-cloud comparisons*.

One problem with these metrics is that they do not consider no-returns. We separately quantify these as follows.

Table 2. Hit/miss types of errors between real and simulated data.

Simulated Data	Real Data	
	Return	No-return
Return	Hit detected	False hit
No-return	False miss	Miss detected

5.3. Hit/miss comparisons

Not all range measurements produce a return value, either in simulation or reality. Thus, in addition to considering the point set comparison metrics, we also need to quantify how many no-returns are *consistent* between simulation and reality. A straightforward way to do this is to consider the four possible scenarios for matched simulated and real lidar returns as shown in Table 2. A low hit-detection rate suggests the model may be incomplete, while a low-miss detection rate may indicate that the part of the model is encroaching on free space.

5.4. Range return data likelihood

Another metric that can be illuminating is to leverage the generative probabilistic nature of the models and use a Bayesian framework to compare the models for given real lidar data. That is, for observed range data from real lidar returns $S = (s_1, ..., s_N)$, and model M, we can evaluate $P(M|S)$, which using the usual Bayes rule manipulations gives: $P(M|S) = \frac{P(S|M)P(M)}{P(S)}$. However, evaluating the model probability $P(M)$ is non-trivial. Thus, we settle for comparing the data likelihood – that is, evaluating $P(S|M) = \prod_i P(s_i|M)$. For the usual numeric reasons, we consider the negative log likelihood:

$$\begin{aligned} \ell(S|M) &= -\log \mathscr{L}(S|M) = -\log P(S|M) \\ &= \sum_i -\log P(s_i|M) \end{aligned} \tag{17}$$

This likelihood value is still often very large and can be hard to intuitively understand. Thus, we report on the average negative log likelihood by dividing this score by the number of beams considered.

One approach to evaluating this likelihood is to generate many samples and form a range histogram or use some other form of density estimation from the resulting samples. This is a computationally expensive process however. A better way is to exploit the probabilities encoded in the model that we use to generate the range samples. For the volumetric and surface models, we can generate the probability distribution function (pdf) for the range returns for given a lidar beam $(p_0, \hat{\boldsymbol{r}})$. For non-permeable surfaces, this is straightforward. We simply propagate the ray to the first triangle intersection. The range pdf for that ray is then the additive range uncertainty model centered at the ray–triangle intersection range.

For the permeable models, the situation is similar but we need to account for all possible range returns. Thus, rather than stopping the ray tracing at the 'hit', we instead continue tracing the ray through the whole model up to the maximum range of the sensor. While ray tracing, we keep track of all model elements that the ray intersects (in order), the associated range pdfs that each generates and their permeability probabilities. We can then form the full range pdf for the ray by assembling these probability distributions in a mixture model. However, we need to account for the preferential role model elements at closer range have. Consider two consecutive model elements intersecting the beam, m_1 and m_2 at ranges r_1 and r_2 with $r_2 > r_1$. To reach m_2, the ray must *first* pass through m_1. Thus, the contribution of m_2 is reduced by the permeability (or lack there of) for m_1. Given the set of model elements along the ray $m_1, ..., m_N$, each with permeabilities $\rho_1, ..., \rho_N$ and range pdfs $P_1(r), ..., P_N(r)$, we can form the full range pdf as a Gaussian mixture:

$$\begin{aligned} P(r) &= (1-\rho_1)P_1(r) + \rho_1[(1-\rho_2)P_2(r) && (18)\\ &\quad + \rho_2[\ \ldots\ ]] && (19)\\ &= (1-\rho_1)P_1(r) + \rho_1(1-\rho_2)P_2(r) + \rho_1\rho_2 \\ &\quad (1-\rho_3)P_3(r)\ldots && (20)\\ &= w_1P_1(r) + w_2P_2(r) + \ldots w_NP_N(r) + w_{N+1}P_\infty \\ & && (21) \end{aligned}$$

Thus, the final result is a mixture distribution. Note, we have explicitly accounted for the probability of the ray passing through to a no-return. With some algebra we can analytically evaluate the mixture weights w_i as:

$$w_1 = 1 - \rho_1 \tag{22}$$

$$w_i = (1-\rho_i)\prod_{j=1}^{i-1}\rho_j, \quad i = 2, \ldots, N \tag{23}$$

$$w_{N+1} = 1 - \sum_i^N w_i \tag{24}$$

For the permeable surface model the model element range pdf is a 1D Gaussian. For the volumetric models, the model element range pdf, once projected onto the ray, is also 1D Gaussian albeit one customized to that particular ray. Once we form the full range pdf, evaluating the negative log likelihood of the data is straightforward.

6. Experiments and results

To evaluate the performance of each of the models described in Section 3 we conducted three experiments: one to quantify the performance of the volumetric models on a variety of representative targets in a static scene; the second to compare the different models in urban and off-road scenes; the third to demonstrate the system operating as a lidar simulator at larger scales. In all cases, the models are

Fig. 12. The data-collection system consists of a Velodyne HDL-64E rigidly mounted to a color camera ring – 6 Point Grey Firefly cameras mounted at 60° intervals – and a NovAtel SPAN INS GPS/IMU. A well-registered, rigidly mounted base station at the test site provides differential GPS updates to the INS. All sensors intrinsics and extrinsics are calibrated and brought into a single coordinate frame. The sensor system is mounted on the top cabin of an LMTV truck although other vehicles could be used. Imagery, range, and pose data is time tagged and logged at sensor rate via a computer (not shown). The colorized point clouds and simulation results shown in this paper are produced by offline processing of these data logs.

constructed from training data collected in the target environment by a high-end data-collection system (described below). An independent dataset is collected in the same environment for testing. In most cases, we sample the model using the vehicle/sensor poses from the test dataset and compare the outputs to the raw test data. Note, this is a slightly different scenario than pure simulation because we do not actually know the ground truth for the vehicle pose, sensor intrinsics/extrinsics and so forth. However, it is as a good test of the system as is practical for large-scale terrain.

Figure 12 shows the data-collection system used for all the results shown here. The system consists of a Velodyne HDL-64E (see Figure 5) mounted rigidly to a NovAtel SPAN RTK 2 cm GPS/IMU unit and a ring of six color cameras (Point Grey Firefly cameras) arranged with optical centers on a circular ring at roughly 60° intervals. The velodyne collects on the order of 1.33 million lidar measurements per second using 64 individual lasers. For our datasets we only use the forward, roughly 190° field of view due to an occluding vehicle fixture (the rear cabin). We use the three forward-looking color cameras to colorize lidar points, although in this paper these colors are only used for display purposes and are not a core part of the algorithm. A relatively straightforward extension to the models can account for color, but this is beyond the scope of this paper. Positioning is provided by the NovAtel unit combined with differential GPS (dGPS) corrections. dGPS corrections are provided over wireless from a nearby GPS base station which is rigidly attached to infrastructure ensuring its long-term stability and is well calibrated with a base position averaged over many months of activity. The typical pose

Fig. 13. Top row from left, the planar target and the high-, medium- and low-density trees. Bottom left, the tall grass and right, the short grass.

error is on the order of 2 cm with heading error at approximately 0.1° and pitch/roll at 0.02°, or better. The whole assembly is rigid and carefully calibrated into one coordinate frame using custom software. The assembly can then be mounted onto any vehicle and for the results reported here, an LMTV (as shown) was used as the base platform. Data is logged in real time for offline use.

6.1. *Static tests on specific terrain types*

In this set of experiments, we compare the volumetric and surface models with a static observer (i.e. the vehicle and sensor are stationary) for a range of specific terrain samples. The goal is to assess whether the volumetric approach is warranted and to scientifically assess the benefits (or lack thereof) for each model.

6.1.1. Experimental setup We collected a large amount of lidar data from a single, static position thereby allowing each laser to perform the same measurement of the same target repeatedly. Thus, we are able to build true sample range histograms and use the full suite of metrics developed in Section 5.

We used six specific targets to make up the scene, as shown in Figure 13. The targets include a pair of intersecting planes, a tree with low-density foliage, medium-density foliage, and high-density foliage, and tall grass. Also in the scene is mowed grass. Each of these targets was approximately 8 m away from the lidar.

For all targets, approximately 5 mins of lidar data was recorded producing approximately 1500 range measurements for each reported orientation of the lidar. One fifth of each target's dataset was used as training data for model building and the remaining four-fifths as test data.

Table 3. Bhattacharyya distances between the real and simulated data for surface and volumetric modeling. Lower numbers are better. Bold numbers are the best for that target.

Target	Surface	Volumetric
Plane	**0.019**	0.068
High-density tree	0.111	**0.074**
Medium-density tree	0.896	**0.125**
Low-density tree	0.622	**0.054**
Tall grass	0.888	**0.095**
Short grass	1.229	**0.091**

6.1.2. Comparison results For these experiments, we used the regular Voxel method to estimate the volumetric Gaussians (with a voxel size of 5 cm) and compared it to a surface model constructed with a grid resolution of 1 cm and a kernel size of 0.04 cm for a high-quality reconstruction. For the surface model, constant variance additive noise is used with $\sigma = 0.005$ m, which was estimated using a planar target approximately orthogonal to the laser beam.

Figure 14 shows range histograms of the real and simulated data generated using surface and volumetric models for four of the targets. These 2D histograms are horizontally binned by laser angle (at the resolution of the encoder: 0.09° for the Velodyne HDL-64E) and vertically binned by range (at the resolution of the lidar's range returns, 2 mm). Effectively, each histogram is a planar slice through the target.

The first two columns of Figure 14 show that the planar target and the high-density tree are simulated satisfactorily by both the surface and volumetric models, although both models do introduce some slight smoothing into the outline of the high-density tree. However, for the low-density tree and the tall grass the Volumetric model produces a better approximation of the real data than the Surface model.

The performance of the two models against the complete set of targets is summarized in Table 3. This table shows the Bhattacharyya distance between the 2D histograms of the simulated and real data using the same bins as before [see equation (14)]. An additional range bin for each orientation is used to represent range measurements that do not result in a hit to ensure that the comparison will account for differences in object permeability produced by the modeling methods. The metric comparisons indicate that, except for the planar target, the Volumetric model produces a more faithful simulation. The Surface model does not work well on the more permeable targets, although its particularly poor performance on the short grass is likely due to the low grazing angle the grass presents to the lidar.

From these results we can be reasonably confident that the volumetric models are better suited to modeling permeable vegetation. Additionally, the volumetric models perform surprisingly well on planar surfaces, with a small loss of performance compared to surface models. This might suggest that one should just use volumetric models everywhere. While this approach is reasonable, we note two drawbacks. First, volumetric models are less compact and require more parameters per unit of volume than the corresponding surface models for terrain that is well approximated as a surface. For large models where modeling efficiency is of concern, this may be an important factor. A second issue occurs with data collection from a moving platform where surfaces have low grazing angles (high angle of incidence). For the system described earlier, this can frequently occur on bare ground. In these situations, volumetric models tend to produce simulated returns that exhibit more range variation than a real sensor shows. Surface models instead exhibit range variation more consistent with the physical sensor. We examine this in more detail below (see Figures 15 and 16, for example). We introduce the Hybrid model (Section 3.3.4) to gain the best of both volumetric and surface techniques. We examine all three approaches on larger datasets collected from moving platforms in the following section.

6.2. Comparing models for urban and off-road scenes

In these experiments, we now consider automatically building models from training data collected from a *moving* vehicle. As such, the histogram comparison methods are no longer applicable (see Section 5). We focus on two terrain types: off-road terrain, and urban terrain. Both datasets were collected in the Pittsburgh area albeit at different times of year and season.

6.2.1. Experimental setup For both the urban and off-road environment, we collected two datasets during two separate passes by the vehicle driving a very similar route on each pass. The first pass was used as training data to build the models and the second pass was used as the test data set to validate the simulation. For each sampled range measurement in the test set, a simulated beam is produced using the logged vehicle and sensor pose.

For the Surface model and the Nearest Neighbors model, the additive Gaussian noise is simulated as varying with angle of incidence according to equation (1). The two constants of equation (1), σ_0 and σ_a, were calibrated from the mobile data using the residual range differences between the test data and the surface reconstruction of a sealed road. From this data σ_0 was found to be 0.015 m and σ_a to be 0.0014 m. σ_0 is larger than the equivalent value used in the static experiments ($\sigma = 0.005$ m) which can be attributed to the presence of positioning and pointing errors in the mobile data.

Section 6.5 summarizes the values of the model estimation and simulation parameters. In general, the models are estimated at a coarser scale than in the previous experiment.

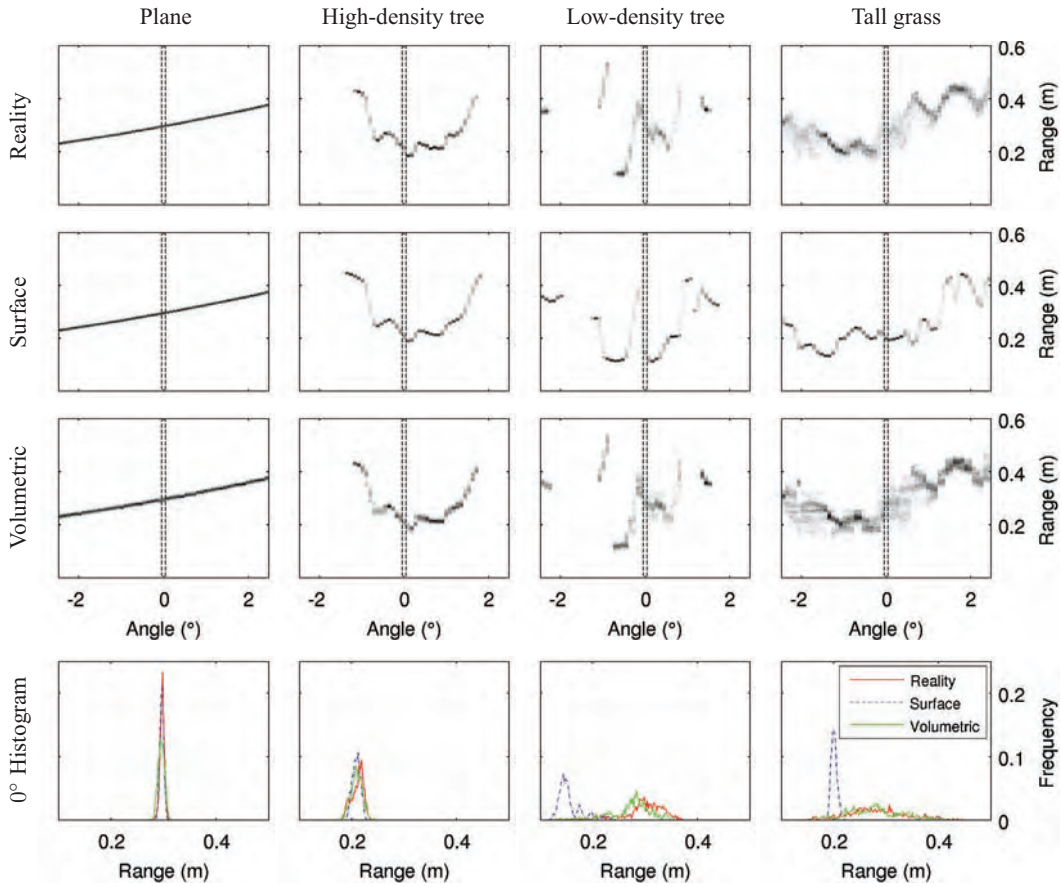


Fig. 14. Real and simulated results for four of the test targets. Each image is a 2D histogram of bearing and range values from an individual laser. Each image shows a cross section of 5° of angle and 0.6 m in range. The bottom row shows the 1D range histograms for 0° of bearing (a vertical cross section taken along the center of the 2D histograms). The real data is the dark (red) line, the Surface model is the dashed line (blue) and the Volumetric model is the light (green) line.

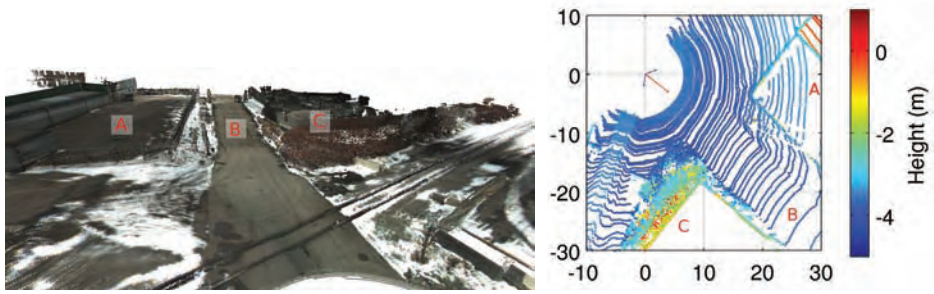
6.2.2. Qualitative comparisons We first compare the models by assessing qualitatively the reconstructed point clouds from each simulated model. Figure 15 shows a single scan from the urban test data set and the corresponding simulated scan using the presented modeling methods. The difference between the Impermeable and Permeable Surface models can be seen most clearly in Figures 15(h) and (i). The chain-link fence visible in the left camera image [Figure 15(a)], has a sparse structure that allows the laser beam to travel through the fence. Unable to account for this, the Impermeable Surface model cannot produce the ground strikes beyond the fence as seen in the real data. In contrast, all the other models are able to reproduce these ground strikes.

The simulations all produce noisier patterns on the ground than the real data, with the surface models having less apparent noise than the volumetric models [compare the pattern at $(20, -20)$]. This suggests that the volumetric Gaussians may not be as good a model as the surface models are for ground hit by the lidar at large angles of incidence. Bare ground will generally be classified as a surface, resulting in the use of a surface model, in the hybrid modeling approach. Thus for the Hybrid model, bare ground will typically perform as well as the surface model while vegetated areas will perform as well as the Volumetric model.

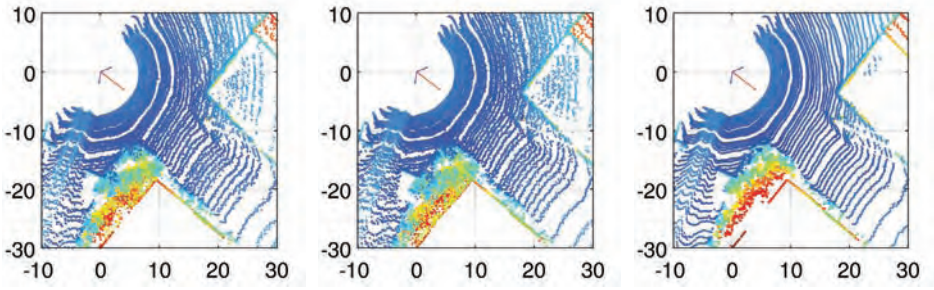
Figure 16 shows the reconstructed point clouds produced by the simulation models in the off-road environment. We chose this particular scene because it incorporates relatively flat roadway, with vegetation that typically is semi-permeable for lidar. The Impermeable Surface model [(Figure 16(e)] performs very poorly at reconstructing the real point cloud, this is most obvious around the crown of the tree in the center of the image. Adding permeability to the Surface model provides a significant improvement, but the basic mesh representation still produces planar artifacts that are not present in the real data. The volumetric models produce a much more convincing simulation, although the voxel structure is partially apparent in the Voxel Gaussian result.

6.2.3. Hit/miss statistics Tables 4 and 5 show the false hit and false miss rates (described in Section 5.3) for the urban and off-road environments. In both tables, it can be seen that the Impermeable Surface model and the Nearest Neighbors

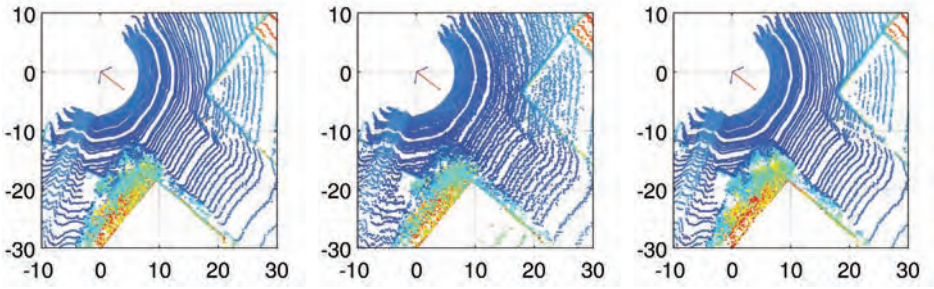
(a) Image from the left camera (b) Image from the middle camera (c) Image from the right camera



(d) Input point cloud (e) Real data



(f) Voxel Gaussian model (g) Top Down Gaussian model (h) Surface model



(i) Permeable Surface model (j) Nearest Neighbors model (k) Hybrid model

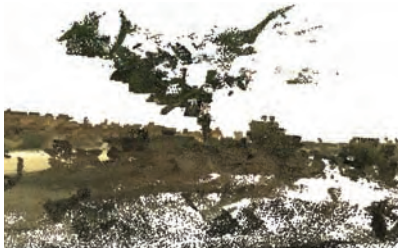
Fig. 15. Comparison between the real and simulated lidar returns for a 360° 'sweep' of the sensor on surfaces. The real data is collected from a second, unseen dataset (the test set) in a novel pose close to the original poses. (a)–(c) The three forward-looking cameras. (d) The input point cloud accumulated from multiple poses. (e) Top-down view of the real lidar returns from a novel pose for one sweep, colorized by *z*, with the *x*–*y* scale in m, with camera viewing directions shown. Note the areas labeled: **A** – a flat area partially occluded by a chain-link fence [see (a)]; **B** – a flat, asphalt roadway; **C** – a brick wall partially occluded by shrubs [see (c)]. The Voxel (f) and Top Down (g) models each show the area behind the chain-link fence and the building occluded by vegetation, but show increased noise on the roadway. The Surface model (h) fails on the chain-link fence area and vegetation occlusion, but has more reasonable noise on the roadway. The Permeable Surface model (i), Nearest Neighbors model (j), and Hybrid model (k) all do well on all three surfaces.

(a) Camera view

(b) Real data

(c) Simulated data with the Voxel Gaussian model

(d) Simulated data with the Top Down Gaussian model



(e) Simulated data with the Surface model without permeability

(f) Simulated data with the Surface model with permeability

(g) Simulated data with the Nearest Neighbors model

(h) Simulated data with the Hybrid model

Fig. 16. Off-road comparison of accumulated point clouds for a sequence of views of vegetation. All models built from the same training data. An independent test run, with a similar but not identical pose sequence was used for evaluation. (a) A picture of the environment from the vehicle camera, and (b) A lidar point cloud from the test run data viewed from the right of the trail in (a). (c) The simulated returns for the Voxel model using the poses but not the lidar returns from the test run. (d) The Top Down model simulated returns, (e) the Surface model, (f) the Permeable Surface model, (g) the Nearest Neighbors and (h) the Hybrid model. The surface models clearly do poorly in this case, although permeability does improve the result. The Volumetric and Nearest Neighbors models perform much better. Note the point color is for visualization only and does not come from the simulation.

model produce a near 100% hit rate, and simulate no-return cases badly. Adding permeability to the surface model produces the worst hit-detection rate which suggests that the estimation of surface permeability may be overestimated. The Hybrid model has almost the same results as the Top Down model, which it uses to represent volumes. This indicates that the regions selected for surface modeling are not the regions for which the Surface model has trouble estimating permeability. From this it can be concluded that it is vegetation that causes this problem.

While the miss-detection rate is low for all models in the urban data set, the number of misses as a fraction of all samples in the test data is also low, so these do not represent a large number of errors.

Table 4. Urban lidar hit/miss results for 6, 277, 193 simulated rays. 1.7% of the corresponding test rays were non-returns (best results are in **bold** and worst in *italics*).

Type	Model detection rate %					
	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbors	Hybrid
Hits	99.3	98.5	**99.8**	*95.9*	**99.8**	98.2
Misses	39.9	38.3	33.6	**46.3**	*33.3*	39.6

Table 5. Off-road lidar hit/miss results for 6, 298, 929 simulated rays. 10.1% of the corresponding test rays were non-returns (best results are **bold** and worst are in *italics*).

Type	Model detection rate %					
	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbors	Hybrid
Hits	98.6	98.1	99.8	*93.8*	**99.9**	97.9
Misses	67.1	66.5	54.5	**74.8**	*52.9*	66.6

6.2.4. Distance metrics Tables 6 and 7 show the sweep-to-sweep and point-cloud-to-point-cloud distances and the average of the negative log likelihood values for each ray. In both environments, the best sweep-to-sweep results come from the permeable surface. The Hybrid model, however, has similar sweep-to-sweep results and a better accumulated point cloud.

The Nearest Neighbors model provides good performance for point-cloud-to-point-cloud comparisons but has a poor sweep-to-sweep distance. This is because the Nearest Neighbors model does not properly model object permeability. A point behind a solid object can still produce a hit if it is closer to the ray than any of the points representing the solid object. This can be seen in Figure 17, where the simulation produces points behind solid objects.

6.2.5. Data likelihood The histograms in Figure 18 show the log likelihood of every test beam being produced by the simulated model. The beams which have likelihoods greater than 0.1 can only occur when the test beam had no return and the model had a high probability of not generating a return. Test beams that do have returns have a lower likelihoods since they are drawn from a continuous distribution of possible ranges. As a consequence, the Impermeable Surface and the Nearest Neighbors models both have few beams in this range of likelihood since they have either 100% or 0% chance of generating a return. A likelihood value of 1 occurs only when the test data has no return and the model has no chance of generating a return.

The main difference between the models is apparent in the location of the peak of the likelihoods. The peak of the surface models is more likely than the peak of the volumetric models. However there are more test beams with lower likelihoods. The peak of the surface models is also relatively higher in the urban data set, indicating that more of this environment can be simulated by these models. The Hybrid model combines the peaks of both the volumetric and surface models, although the tail has a slower fall-off than the volumetric models.

6.2.6. Model efficiency The efficiency of the different models can be compared by examining the number of parameters used to represent each model. Although the selection of the model parameterization can change the actual memory usage, for example the specific triangle mesh representation, these figures provide a guide as to how large the model grows as a function of real data.

Table 8 gives the number of elements (triangles, Gaussians, points) for the Surface, Volumetric and Nearest Neighbors models respectively. Each triangle requires 9 parameters or 10 if adding permeability while each Gaussian requires 10 parameters. Triangles in a mesh can be represented more compactly by sharing corners, which would reduce the number of points stored, albeit with the cost of the additional indices. The Nearest Neighbors model simply keeps the 3D position of all of the points used for training, each of which requires three parameters. The efficiency of the Nearest Neighbors model could obviously be improved by sub-sampling and is only included for completeness. We can then calculate the number of parameters for each models with the number of elements in the model.

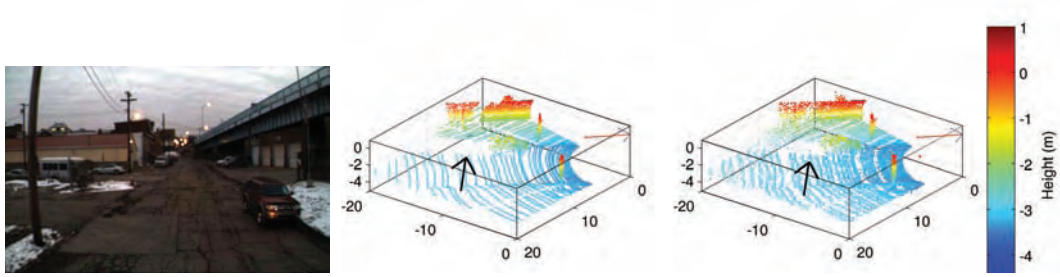
From Table 8 and 9, it is clear that while the Top Down model produces better results it does so by producing more than three times as many Gaussians as the Voxel method. The Hybrid model is also more efficient than the Top Down model it incorporates. The number of parameters required for representing triangles is a conservative estimate as many of the triangles share vertices which would reduce the complexity of their storage. Additionally mesh simplification could be employed to reduce the storage cost of the Surface model. These efficiency gains would also apply to the Hybrid model where the parts of the environment most amenable to mesh simplification are modeled as surfaces.

Table 6. Urban lidar simulation metrics (best results are in **bold** and worst are in *italics*). Note the Hybrid model performs almost as well as the best model for each metric.

Metric	Model distance					
	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbor	Hybrid
Real–sim sweeps distance [cm]	15.23	14.63	*81.44*	**12.72**	18.19	12.79
Real–sim point clouds distance [cm]	3.77	3.74	*45.12*	4.02	**3.47**	3.48
Average $-\log \mathscr{L}$	6.59	**6.54**	9.59	8.39	*9.65*	7.21

Table 7. Off-road lidar simulation metrics (best results are in **bold** and worst are in *italics*). Note the Hybrid model performs almost as well as the best model for the first metric.

Metric	Model score					
	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbors	Hybrid
Real–sim sweeps distance [cm]	17.46	17.43	*29.11*	**16.07**	22.91	16.65
Real–sim point clouds distance [cm]	4.61	4.55	*10.09*	6.17	4.62	**4.38**
Average $-\log \mathscr{L}$	6.84	**6.65**	*10.76*	9.52	10.50	7.22



(a) Forward camera (b) Real data (c) Nearest Neighbors model

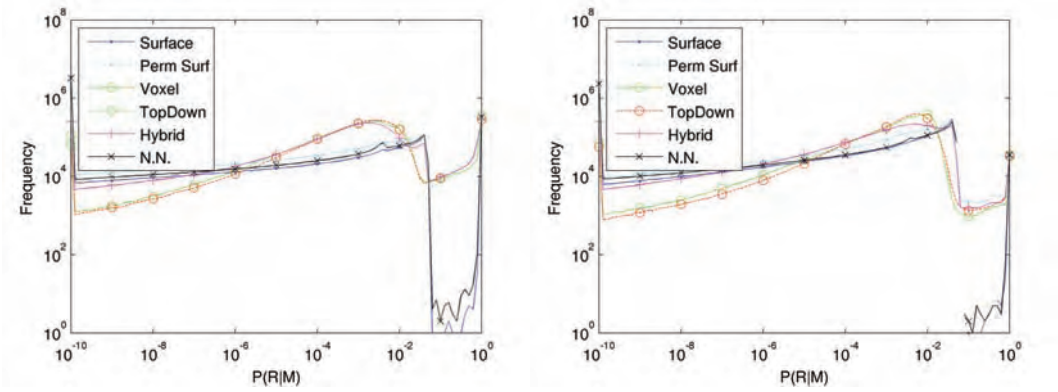
Fig. 17. Although the Nearest Neighbors model produces good results, it has some serious flaws. (a) An urban scene with the sensor camera and (b) from the side the real point returns for a sweep colorized by height. The scene includes a car which occludes the ground behind the vehicle (black arrow). A Nearest Neighbors model is trained on a sequence of images from a vehicle traveling down the road. (c) The simulation using the Nearest Neighbors model for the sensor pose shown in (a) and (b). The training dataset is more extensive and includes views of the ground behind the car from different poses. The Nearest Neighbors model does not reason about this occlusion and generates returns behind the car [black arrow in (c)]. The same effect occurs but is less noticeable in the shadow of the telephone pole. The Volumetric and Hybrid models do not suffer from this problem.

Table 8. Number of parameters in the urban lidar simulation results (g = Gaussians, t = triangles, pt = points).

	Model size					
Metric	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbors	Hybrid
Elements	154,586 g	579,349 g	356,999 t	356,999 t	16,748,967 pt	192,064 g + 225,698 t
Parameters	**1,545,860**	5,793,490	3,212,991	3,569,990	50,246,901	4,177,620

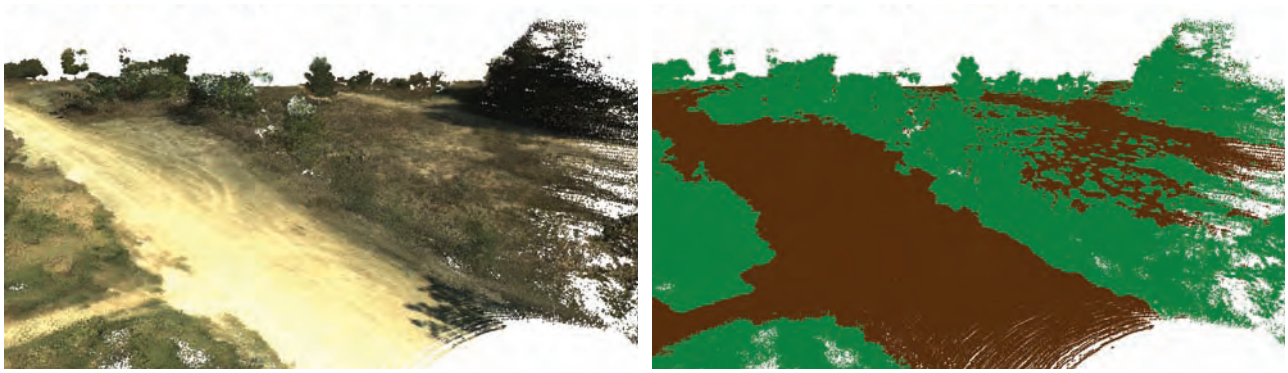
Table 9. Number of parameters in the off-road lidar simulation results (g = Gaussians, t = triangles, pt = points).

	Model size					
Metric	Voxel	Top Down	Surface	Permeable Surface	Nearest Neighbors	Hybrid
Elements	190,469 g	520,982 g	531,908 t	531,908 t	15,567,210 pt	377,678 g + 114,995 t
Parameters	**1,904,690**	5,209,820	4,787,172	5,319,080	46,701,630	4,926,730



(a) Off-road terrain (b) Urban terrain

Fig. 18. The data likelihood on an independent test dataset measures how likely the observed real returns are to have come from the particular simulation model. (a) The data likelihood histogram for the off-road scene and (b) the likelihood histogram for the urban scene. Both are plotted on a log–log scale. The *x*-axis represents the likelihood value, while the *y*-axis represents the frequency. A better model has more weight at higher likelihood values (Note the log scale enhances this effect). The Hybrid model performs the best in both settings. The volumetric models perform better in off-road, and are comparable in the urban scene. The Impermeable Surface model performs very badly as does the Nearest Neighbors model due to the incorrect handling of occlusions.



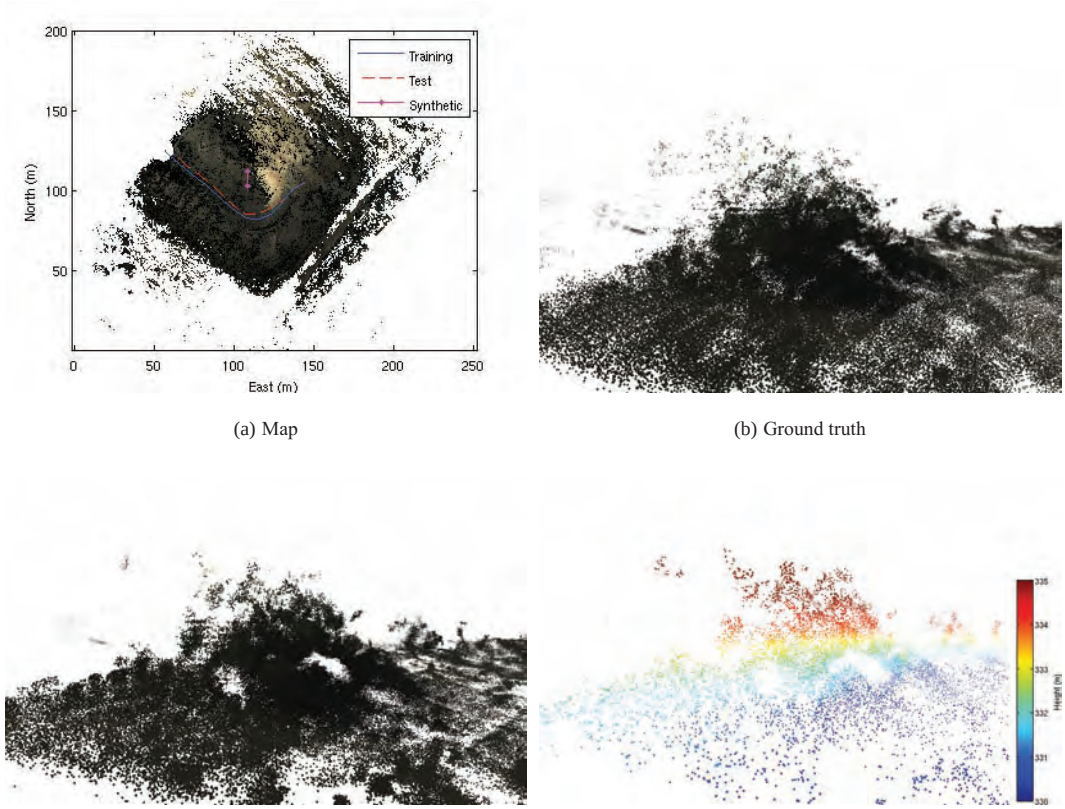
(a) Input point cloud (b) Classification result

Fig. 19. Example of the point cloud classification and bilateral filtering for the off-road scene for hybrid modeling. (a) The input point cloud. (b) The classified points with surfaces drawn in brown (dark points) and non-surfaces as green (light-colored points). The trail (bare earth) and vegetation (grass, tree, brush) are clearly labeled correctly. It is unclear if the borders are labeled correctly. Secondly the area to the top right is ambiguous terrain – it is mostly bare earth but has some trace, short grass.

6.2.7. Hybrid environment classification The images in the Figure 19 show the classification of the test data points into volume and surface classes, used by the Hybrid model estimation process. Table 10 shows the breakdown of the training data into the two categories. As discussed earlier, the classification method used is reasonable but is not optimal. State-of-the-art terrain-classification methods will make better use of local and global context (Munoz et al., 2009), however, for hybrid modeling we really desire to segment terrain based on how well the different model types will represent it and not on an arbitrary definition of terrain type. More importantly, mistakes in classification have different costs. Based on the results shown above, labeling a surface as a volume will not significantly degrade performance. However, labeling vegetation (non-surface) as a surface model will lead to a greater loss of performance. Thus, classification should incorporate a loss function that accounts for these differing types of mistakes. Future work will investigate the role of classification more thoroughly. We note, however, that the empirical results indicate that even with sub-optimal classification, the performance of the hybrid approach is comparable to the best of volumetric and surface modeling approaches in either terrain type.

6.3. Synthetic view generation

In the previous results, the displacement between the training and test trajectories is non-zero, but small (equivalent to executing the same path twice with slightly different noise).



(a) Map

(b) Ground truth

(c) Simulated from test path

(d) Simulated from synthetic path

Fig. 20. (a) Three trajectories shown with the training data: the vehicle position in the training dataset; a second test trajectory which has ground truth data; and an arbitrary synthetic trajectory. There is an approximately 4 m separation between the training and test trajectories. (b) Part of the ground truth data showing a small cluster of vegetation. (c) The point cloud obtained by simulating from the test trajectory. (d) The same region, now simulated using the synthetic trajectory. The point cloud has been colored by height as there is no real-world color information. The point cloud is less dense because the trajectory of the simulated sensor does not approach the target as closely.

Table 10. Breakdown of training data by beam. Each beam in the training data that is not a non-return produces a point which is classified as either being part of a volume or a surface. Beams with no-return mostly, but not entirely, correspond to the sky.

Environment	Non-return	Surface	Volume
Off-road	1,953,005 (11.1%)	5,325,607 (30.4%)	10,241,603 (58.5%)
Urban	365,297 (2.1%)	10,953,204 (64.0%)	5,795,763 (33.9%)

These results show operation with more significant differences, equivalent to making different control decisions. Figure 20 shows the results of simulating a path parallel to the training path. Also shown in Figure 20 is the result of simulating from an arbitrarily selected trajectory rather than from a real-world test trajectory. This is the normal operation of the system when used as a simulator, rather than simulating against real-world trajectories to evaluate the simulation. The simulated velodyne sensor traveled south at 3.0 m/s at a constant altitude. To create a pleasing point cloud, the sensor was randomly reorientated in pitch and roll for each scan, analogous to the small variations in vehicle orientation caused by undulating terrain.

6.4. Large-scale results

Figure 21 shows an overhead view of a point cloud produced by 400 m of simulated travel through the off-road environment. The simulation used the Voxel Gaussian model which was estimated from 80 million points. The distance between the real data and the simulated point cloud is 4.6 cm. The Voxel Gaussian model is particularly suited to scaling up to large environments as it depends on only local operations as opposed to the Top Down model, as a result it can be computed more efficiently and lends itself to parallelization. The Top Down model, with a careful implementation should enable scaling up if the data is first segmented

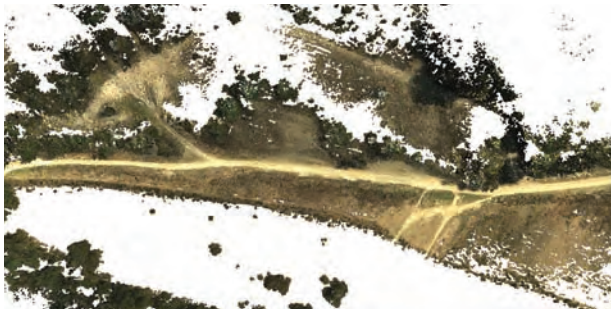


Fig. 21. Large-scale off-road simulated point cloud (400 m long).

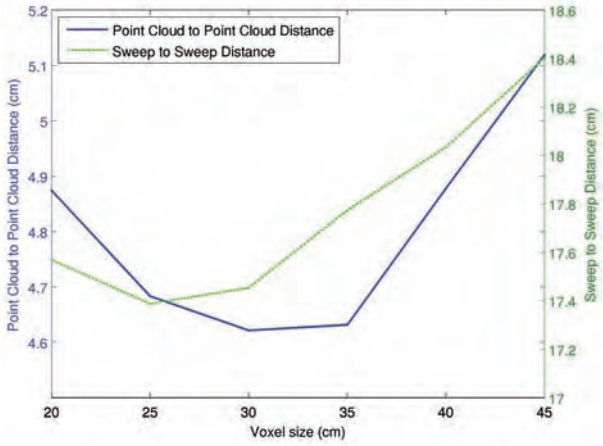


Fig. 22. Empirical sensitivity of modeling to the voxel size. The *x*-axis gives the voxel size (s). The right *y*-axis measures the sweep-to-sweep distance. The left *y*-axis shows the point-cloud-to-point-cloud distance. The graph clearly shows a sweet spot for the voxel dimension, but also that this sweet spot depends on the metric being used.

approximately. Care must be taken not to introduce artificial artifacts at boundaries however.

6.5. Model and simulation parameter summary

In the previous sections, we have introduced a number of different models. Each has a range of parameters associated with them that affect the performance and efficiency of the resulting simulation. As an aid for the reader, Table 12 lists the major parameters used for each model and their assigned values (where not learned from data).

Additionally, we have performed an empirical sensitivity analysis to the voxel size. Figure 22 shows the influence of the voxel size on simulation results. As the voxel size increases, the structure of the environment is smoothed decreasing simulation accuracy. However as the voxel size decreases, the number of points in each voxel falls reducing the accuracy of the shape estimation. For all the experiments reported above, we kept the voxel size at $s = 0.30$ m, which is close to the minimum of both the sweep-to-sweep and point-cloud-to-point-cloud distances.

A similar trade-off exists for the Top Down segmentation method and the parameter N_{p} (maximum number of points per Gaussian to stop recurrence) can be chosen using sweep-to-sweep and point-cloud-to-point-cloud distances. We found $N_{\mathrm{p}} = 120$, which means each Top Down Gaussian has a maximum of 120 points. For the Nearest Neighbors model, we used $d_{\max} = 0.01$ m and for the Surface model, the grid size was $s = 0.30$ m and the kernel size $K_s = 0.40$ m. All were based on empirical evaluations.

6.6. Discussion

We now bring together the different results to draw conclusions about which modeling approach is superior.

6.6.1. Model comparisons Table 11 summarizes the advantages and disadvantages of each model as shown by the results in this section. Overall, the results clearly indicate that impermeable surface models are good only for scenes that are well approximated by large planar segments. Such cases occur in some urban environments, and tunnel/mine environments such as underground mines where relatively smooth bare earth dominates the landscape. Note, urban scenes containing 'transparent' features such as chain-link fences will cause issues for impermeable surface models. Permeable surface modeling, which has relatively little additional cost, can address many of these issues provided the geometry of the scene is still well approximated by a surface.

The volumetric approaches perform well across the board. The volumetric approaches excel for small-scale geometries such as vegetation. Volumetric methods also perform *almost* as well as surface methods in scenes well approximated by a smooth surface. The caveat, however, is that volumetric models require more parameters (i.e. memory) to define the same area, and may have slightly increased range noise compared to surface models. For many scenes, this will not be an issue and may be addressed through cache-like mechanisms to reduce the in-core footprint in very large-scale simulations. The approach presented is highly parallelizable and, with appropriate segmentation of the data, could be spread across multiple machines not just multiple cores making it amenable to large-scale computing approaches. Although not explicitly analyzed, the run-time costs for simulation are dominated by the ray tracing operations and are therefore comparable to surface modeling. Recent advances in CPU techniques (e.g. NVIDIA's Optix package) make real time simulation possible, even for high sample rate sensors such as the Velodyne HDL-64E.

The hybrid approach attempts to get the best of both worlds: surface modeling for efficiency where it makes

Table 11. Advantages and disadvantages of the examined models.

Model	Advantages	Disadvantages
Surface	Good scan performance on ground areas.	Outperformed by Permeable Surface.
Permeable Surface	Good scan performance on ground areas.	Poor performance on vegetation, can overestimate permeability.
Voxel Gaussian	Efficient, highly scalable, good performance on vegetation.	Outperformed by Top Down Gaussian, can overestimate noise on surfaces.
Top Down Gaussian	Good performance on vegetation.	More complex to estimate and larger to store than Voxel Gaussian, can overestimate noise on surfaces.
Hybrid	Retains Surface model performance on ground and has volumetric Gaussian performance on vegetation.	Additional computation step that relies on terrain classification.

sense, and volumetric modeling everywhere else. Core to the success of the hybrid approach is to correctly use surface modeling *only* where it makes sense to do so. Our current approach makes this decision early in the process using relatively standard point cloud classification techniques. Although this decision process may not be optimal, the empirical results demonstrate that the system performs well at least on the scenes evaluated. Future work is required to validate this approach on a broader set of scenes and to investigate methods to make more optimal classification decisions.

6.6.2. Model limitations Our approach does not address a number of core issues in real-world UGV operations. Specifically, our approach does not address:

- **Reflective/refractive materials:** such as standing water (i.e. puddles) which can cause a range of complex lidar returns that can be very difficult to interpret without additional sensor input.
- **Atmospheric scattering and absorption effects:** such as fog, dust, hail, rain, smoke, etc. The physics of these phenomena are reasonably well understood, but obtaining realistic parameters for a given geo-specific simulation is non-trivial. Incorporating atmospheric effects is an area of future work.
- **Scene dynamics:** such as wind blown leaves or trees. Our current approach assumes static scenes. Future work would be required to extend the model to account for the correlations that occur for wind-blown artifacts. One approach may be to follow methods used for Radar modeling and explicitly develop a statistical model for temporal fluctuation induced by wind. Estimating the parameters of such a model for a geo-specific and weather-specific scene may be quite difficult in practice however.
- **Sampling limitations:** such as simulating from areas that are under sampled. The set of approaches presented here is similar in spirit to *view interpolation* approaches used in vision and robotics [e.g. Huber et al. (2009)]. As such, there is an inherent assumption that the part of the scene to be simulated has been appropriately sampled in the mapping process. Thus adequate mapping coverage is critical to the presented techniques. There may be value however, in blurring the boundary between geo-specific modeling and geo-typical by being able to *extrapolate* unseen parts of the scenery, but in a way that is consistent to the existing parts already mapped. It is unclear how to do this properly however, and this is an area of future work.

7. Conclusions

High-fidelity simulation of a lidar in outdoor environments is a challenging problem, and one that is inadequately addressed in state-of-the-art simulation packages. In this paper, we have introduced a novel volumetric approach to sensor–terrain interaction modeling for lidar sensors. Furthermore, we have introduced a novel data-driven learning approach that enables automatic construction of the model for a geo-specific scene based on logged data. To evaluate the approach, we have compared its performance against a range of surface-modeling approaches on a range of real-world scenes. The empirical results demonstrate that for most real-world terrain, our volumetric approach performs better than the corresponding surface-based, or nearest-neighbor techniques. Finally, we introduced a hybrid model that captures the best of both surface representations and volumetric representations and can be automatically constructed from data logs. Table 11 lists the major model types and their relative advantages and disadvantages.

Our future work will focus on scaling up this technique to enable real-time, high-fidelity simulations of large-scale scenes and generalizing the approach to enable extrapolation to different lidar sensors than those used to record the training data.

Table 12. The main parameters used for each respective model, and their setting for the results reported here. See Sections 3 and 4 for details.

Symbol	Description	Value
Nearest Neighbors (Section 3.3.1)		
$d_{\max}$	Maximum perpendicular distance that a ray can pass by a point and 'hit' it. Decreasing this number increases the ability of rays to pass through point cloud. Increasing it 'fattens' objects.	0.01 m
Permeable and Impermeable Surface (Section 3.3.2)		
s	Voxel size for marching cubes estimation. Determines size of triangles in mesh. Smaller triangles require more computation and memory. Larger triangles lose detail. Value chosen empirically and to match voxels for fair comparisons.	0.3 m
K_s	Kernel size for RIMLS. Larger kernels produce smoother results, but may 'round-off' detail. Smaller kernels may lead to gaps in the Surface model where points are sparse.	0.4 m
σ_0	Range variance for a perpendicular surface.	Learned
σ_a	Range variance dependent on incidence angle.	Learned
Δ_r	Offset for points considered to have hit triangle. Accounts for returns that are in front of estimated triangle. Relatively insensitive.	0.5 m
Voxel Gaussian (Section 3.3.3)		
s	Voxel size. See empirical analysis in Figure 22. Voxel size will depend upon data density.	0.4 m
τ	Mahalanobis distance for iso-contour defining volume size. Decreasing it too much can lead to gaps between Gaussians. Increase increases overlap, but permeability reduces any negative impact.	3.5
N_v	Minimum number of points in a voxel for estimation. Required to ensure Gaussians can be calculated. Also applies a simple filtering effect (e.g. for dust).	5
Top Down Volumetric Gaussian (Section 3.3.3)		
τ	Mahalanobis distance for iso-contour defining volume size. See the Voxel Gaussian model.	3.5
K	Number of means at each recursive iteration. Larger values of K split the data faster, but requires more computation. Smaller values are at risk of making a poor split early on, however, value is relatively insensitive. K chosen based on empirical performance.	10
N_p	Minimum number of points to continue recursive K-means. Larger values cause earlier termination of recursive K-means producing larger Gaussians. Smaller values produce smaller Gaussians. Empirically determined.	120
Hybrid model (Section 3.3.4)		
s_d	Sphericalness default value when insufficient number of neighbor points.	Learned
$N_{\min}$	Minimum number of neighbors required to calculate spherical-ness score.	Learned
N_f	Number of neighbors considered in bilateral filtering calculation.	Learned
d_f	Maximum neighbor distance for bilateral filtering.	Learned
α_d	Bilateral filtering weighting term for distance.	Learned
α_s	Bilateral filtering weighting term for score similarity.	Learned
s_{vol}	Score threshold to be considered a volume (non-surface).	Learned

Notes

1. This work was conducted at the National Robotics Engineering Center. Jean-Emmanuel Deschaud is now at Mines ParisTech, CAOR – Robotics Center, Mathematics and System Department, France.,
2. We will use the term geo-specific to refer to simulations that aim to realistically capture a geo-referenced real-world location. Geo-typical simulations on the other hand represent environments from a region or type (e.g. North East American forests), but have no correspondence to any real-world location.
3. The data capture system is shown in Figure 12.
4. A very rough estimate based on: http://answers.yahoo.com/question/index?qid=20080925182716AA9Nqgz.
5. For this work, we consider simulating a lidar of the same type as used to collect the data. In future work, we intend to expand the scope of this approach to consider simulating different lidars (geometry, sensing configuration and wavelength) that are used to collect data, but that is beyond the scope of this paper.
6. Strictly speaking this speed should be adapted for the given environmental conditions, but over the distances used for UGV sensors, 100 m, such variations are negligible
7. http://velodynelidar.com/.
8. The more complete results for this scene are shown in Figure 16.
9. Table 12 in the results section (Section 6) details the parameters for each model and the settings we used for the results in this paper.
10. http://www.nvidia.org/object/optix.

Funding

This work was sponsored by the US Army Engineer Research and Development Center (ERDC) under the cooperative agreement "Fundamental Challenges in World and Sensor Modeling for UGV Simulation" (grant number W912HZ-09-2-0023). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Government.

Acknowledgments

The authors thank Tom Pilarski, Scott Perry, and Cliff Olmstead for their generous support in maintaining and deploying the data-collection equipment used in this paper.

References

Baltsavias E (1999) Airborne laser scanning: basic relations and formulas. *ISPRS Journal of Photogrammetry and Remote Sensing* 54(2), pp. 199–214.

Bhattacharyya A (1943) On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society* 35: 99–109.

Carpin S, Lewis M, Wang J, Balakirsky S and Scrapper C (2007) USARSim: a robot simulator for research and education. In: *IEEE International Conference on Robotics and Automation (ICRA '07)*, Rome, Italy, 10–14 April 2007, pp. 1400–1405. Piscataway: IEEE Press.

Dima C, Wellington C, Moorehead S, Lister L, Campoy J, Vallespi C et al. (2011) PVS: A system for large scale outdoor perception performance evaluation. In: *IEEE International Conference on Robotics and Automation (ICRA'2011)*,IEEE Press, Shanghai, China, pp. 834–841.

Durst P, Goodin C, Gates B, Cummins C, McKinley B, Priddy J et al. (2011) The need for high fidelity robotics sensor models. *Journal of Robotics* 2011: 679875.

Fleishman S, Drori I and Cohen-Or D (2003) Bilateral mesh denoising. *ACM Transactions on Graphics* 22(3), pp. 950–953.

Friedmann M, Petersen K and von Stryk O (2010) Evaluation and enhancement of common simulation methods for robotic range sensors. In: *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'10)*, Springer, Darmstadt, Germany, pp. 63–74.

Gardner CS (1992) Ranging performance of satellite laser altimeters. *IEEE Transactions on Geoscience and Remote Sensing* 30(5), pp. 1061–1072.

Georgiadis S, Gunnion AJ, Thomson RS and Cartwright BK (2008) Bird-strike simulation for certification of the Boeing 787 composite moveable trailing edge. *Composite Structures* 86(1), pp. 258–268.

Glennie C (2007) Rigorous 3D error analysis of kinematic scanning lidar systems. *Journal of Applied Geodesy* 1(3), pp. 147–157.

Gonzalez JP, Dodson W, Dean R, Kreafle G, Lacaze A, Sapronov L et al. (2009) Using RIVET for parametric analysis of robotic systems. In: *Ground vehicle systems engineering and technology symposium (GVSETS '09)*, Detroit, Michigan.

Goodin C, Kala R, Carrrillo A and Liu L (2009) Sensor modeling for the virtual autonomous navigation environment. In: *8th IEEE Conference on Sensors*, Christchurch, New Zealand, pp. 1588–1592.

Greenberg DP, Torrance KE, Shirley P, Arvo J, Lafortune E, Ferwerda JA et al. (1997) A framework for realistic image synthesis. In: *24th annual conference on computer graphics and interactive techniques (SIGGRAPH '97)*, ACM Press/Addison-Wesley Publishing Co., pp. 477–494.

Hamerly G and Elkan C (2002) Alternatives to the *K*-means algorithm that find better clusterings. In: *11th international conference Information and Knowledge Management (CIKM'02)*, ACM, Maclean, VA, USA, pp. 600–607.

Holtzman J, Frost V, Abbott J and Kaupp V (1978) Radar image simulation. *IEEE Transactions on Geoscience Electronics* 16(4), pp. 296–303.

Hong S, Lee MH, Chun HH, Kwon SH and Speyer J (2005) Observability of error states in GPS/INS integration. *IEEE Transactions on Vehicular Technology* 54(2), pp. 731–743.

Huber D, Herman H, Kelly A, Rander P and Ziglar J (2009) Real-time photo-realistic visualization of 3D environments for enhanced tele-operation of vehicles. In: *12th International Conference on Computer Vision (ICCV'09), Workshops, (ICCV '09)*, pp. 1518–1525. Piscataway: IEEE Press.

Hulst H (1981) *Light Scattering by Small Particles*. Dover Publications.

Jones H and Vaughan R (2010) *Remote Sensing of Vegetation: Principles, Techniques, and Applications*. Oxford: Oxford University Press.

Kalaiah A and Varshney A (2005) Statistical geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics* 24: 348–373.

Kelly A, Amidi O, Herman H, Pilarski T, Stentz A, Vallidis N et al. (2006) Toward reliable off road autonomous vehicles operating in challenging environments. *The International Journal of Robotics Research* 25: 5–6.

Kim AM, Olsen RC and Borges CF (2010) Simulating full-waveform lidar. In: *15th SPIE Conference on Laser Radar Technology and Applications XV, Vol. 7648, SPIE, Orlando, Florida*. vol. 7684.

Koenig N and Howard A (2004) Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04)*,IEEE, Sendai, Japan.

Kukko A and Hyyppa J (2009) Small-footprint laser scanning simulator for system validation, error assessment, and algorithm development. *Photogrammetric Engineering and Remote Sensing* 75(9), pp. 1177–1189.

Lalonde JF, Vandapel N, Huber D and Hebert M (2006) Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics* 23(1), pp. 839–861.

Levoy M and Hanrahan P (1996) Light field rendering. In: *23rd annual conference on computer graphics and interactive techniques (SIGGRAPH '96)*, pp. 31–42. New York: ACM Press, New Orleans, Louisiana.

Lindenmayer A (1968) Mathematical models for cellular interactions in development. *Journal of Theoretical Biology* 18(3), pp. 300–315.

Livny Y, Yan F, Olson M, Chen B, Zhang H and El-Sana J (2010) Automatic reconstruction of tree skeletal structures from point clouds. *ACM Transactions on Graphics* 29(6), New York: ACM Press, pp. 151–159.

Lorensen W and Cline H (1987) Marching cubes: A high resolution 3D surface construction algorithm. *ACM Siggraph Computer Graphics* 21(4): 163–169.

Magnusson M, Duckett T and Lilienthal AJ (2007) Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics* 24(10), 803–827.

Mallet C and Bretar F (2009) Full-waveform topographic lidar: State-of-the-art. *ISPRS Journal of Photogrammetry and Remote Sensing* 64: 1–16.

Munoz D, Bagnell J, Vandapel N and Hebert M (2009) Contextual classification with functional max-margin Markov networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '09)*, pp. 975–982. Piscataway: IEEE Press.

Nicodemus F (1965) Directional reflectance and emissivity of an opaque surface. *Applied Optics* 4: 767–773.

Öztireli AC, Guennebaud G and Gross M (2009) Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum* 28: Wiley Online Library, pp. 493–501.

Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J et al. (2009) ROS: an open-source robot operating system. In: *IEEE International Conference on Robotics and Automation (ICRA'2009), Workshop on Open Source Software*, Kobe, Japan, May.

Ryde J and Hillier N (2009) Performance of laser and radar ranging devices in adverse environmental conditions. *Journal of Field Robotics* 26: 712–727.

Stentz A, Bares J, Pilarski T and Stager D (2007) The crusher system for autonomous navigation. In: *AUVSI's Unmanned Systems North America*,Washington D.C., USA.

Storn R and Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11: Springer, pp. 341–359.

Tan P, Zeng G, Wang J, Kang SB and Quan L (2007) Image-based tree modeling. *ACM Transactions on Graphics* 26(3), Article 87, pp. 1–7.

Thrun S, Haehnel D, Ferguson D, Montemerlo M, Triebel R, Burgard W et al. (2003) A system for volumetric robotic mapping of abandoned mines. In: *IEEE International Conference on Robotics and Automation (ICRA '03)*, vol. 3, IEEE Press, Taipei, Taiwan, September pp. 4270–4275.

Tomasi C and Manduchi R (1998) Bilateral filtering for gray and color images. In: *6th IEEE International Conference on Computer Vision (ICCV'98)*, Bombay, India, January, pp. 839–846. Piscataway: IEEE Press.

Tuley J, Vandapel N and Hebert M (2005) Analysis and removal of artifacts in 3-D LADAR data. In: *IEEE International Conference on Robotics and Automation (ICRA '05)*, Barcelona, Spain, April pp. 2203–2210.