Remote Sensing of Environment 269 (2022) 112772

Contents lists available at ScienceDirect

Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning

Lukas Winiwarter [a,*], Alberto Manuel Esmorís Pena [b], Hannah Weiser [a], Katharina Anders [a,c], Jorge Martínez Sánchez [b], Mark Searle [a], Bernhard Höfle [a,c,*]

[a] 3DGeo Research Group, Institute of Geography, Heidelberg University, Germany
[b] Centro Singular de Investigación en Tecnoloxías Intelixentes, CiTIUS, USC, Spain
[c] Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Germany

ARTICLE INFO

Edited by Jing M. Chen

Keywords:
Software
LiDAR simulation
Point cloud
Data generation
Voxel
Vegetation modelling
Diffuse media

ABSTRACT

Topographic laser scanning is a remote sensing method to create detailed 3D point cloud representations of the Earth's surface. Since data acquisition is expensive, simulations can complement real data given certain premises are met: (i) models of 3D scene and scanner are available and (ii) modelling of the beam-scene interaction is simplified to a computationally feasible while physically realistic level. A number of laser scanning simulators for different purposes exist, which we enrich by presenting HELIOS++. HELIOS++ is an open-source simulation framework for terrestrial static, mobile, UAV-based and airborne laser scanning implemented in C++. The HELIOS++ concept provides a flexible solution for the trade-off between physical accuracy (realism) and computational complexity (runtime, memory footprint), as well as ease of use and of configuration. Features of HELIOS++ include the availability of Python bindings (`pyhelios`) for controlling simulations, and a range of model types for 3D scene representation. Such model types include meshes, digital terrain models, point clouds and partially transmissive voxels, which are especially useful in laser scanning simulations of vegetation. In a scene, object models of different types can be combined, so that representations spanning multiple spatial scales in different resolutions and levels of detail are possible. HELIOS++ follows a modular design, where the core components of platform, scene, and scanner can be individually interchanged, and easily configured. HELIOS++ further allows the simulation of beam divergence using a subsampling strategy, and is able to create full-waveform outputs as a basis for detailed analysis. We show how HELIOS++ positions among other VLS software in terms of input model support and simulation of beam divergence in a literature survey. We also perform a direct comparison of simulations with DART, where we employ a scene from the Radiative Transfer Model Intercomparison (RAMI). This example shows that HELIOS++ takes about 10 times longer than DART for parsing and preparing the 3D scene, but performs about 314,000 times faster in the beam simulation, achieving 200,000 rays/s. Comparing HELIOS++ to its predecessor, HELIOS, revealed reduced runtimes by up to 99%. Virtually scanned point clouds may be used for a broad range of applications as shown in literature. We could identify four main categories of use cases prevailing at present, which benefit from simulated LiDAR point clouds: data acquisition planning, method evaluation, method training and sensing experimentation. We conclude that a general-purpose LiDAR simulator can be employed for many different scientific applications, as long as it is ensured that the simulation adequately represents reality, which is specific to the given research question.

1. Introduction

Simulation of physical processes is often carried out when experiments are not feasible or simply impossible, or to find parameters that produce a certain outcome if inversion is non-trivial. In virtual laser scanning (VLS), simulations of LiDAR (Light Detection and Ranging) generate 3D point clouds from models of scenes, platforms, and scanners (Fig. 1) that aim to recreate real-world scenarios of laser scanning acquisitions. Such simulated point clouds may, for certain use cases, replace real data, and may even allow for analyses where real data

* Corresponding authors at: 3DGeo Research Group, Institute of Geography, Heidelberg University, Germany.
E-mail addresses: lukas.winiwarter@uni-heidelberg.de (L. Winiwarter), hoefle@uni-heidelberg.de (B. Höfle).

https://doi.org/10.1016/j.rse.2021.112772
Received 23 January 2021; Received in revised form 13 October 2021; Accepted 23 October 2021
Available online 12 November 2021

capture is not feasible, e.g., due to technical, economical or logistic constraints, or when simulating hardware which is not yet existing. However, there are use cases where VLS is not appropriate, for example, when analysing effects only partially modelled in the simulation such as penetration of the laser into opaque objects. In a similar argument, (passive) photogrammetry is inadequate for the reconstruction of a non-textured flat area, but a useful method for many other tasks. Therefore, VLS can be seen as a tool to acquire 3D geospatial data for certain use cases and under certain premises. These include:

(i) an adequate model of the 3D scene and the scanner, as well as the platform behaviour, and
(ii) a simplification of the real-world beam-scene interactions to a computationally feasible yet physically realistic level.

VLS can easily and cheaply produce large amounts of data with very well defined and known properties of the point cloud. Additionally, values of the parameters describing the virtual scene environment and its objects are available. Such parameters (e.g., tree attributes such as the stem diameter of the 3D object used in the simulation) can be extracted from the scene model automatically and without errors. These parameters can in turn be used for training or validation of algorithms that attempt to extract them from point cloud data. Due to the low cost compared to real data acquisitions, VLS can be combined with Monte-Carlo simulations to solve non-continuous optimisation problems on scan settings and acquisition strategies. In a research workflow, VLS experiments may be employed to identify promising candidate settings before carrying out a selected number of real experiments that are used to answer the respective research questions. Such applications can be united in a general-purpose LiDAR simulator, which is not only designed for one use case but shall serve the majority of use cases.

This paper aims to answer the question of the usability of VLS in remote sensing research. For this, we present a novel take on ray tracing-based VLS covering the given premises (i–ii) and will identify use cases that benefit from VLS. This LiDAR simulator is implemented as the open-source software package HELIOS++ (Heidelberg LiDAR Operations Simulator ++)[1]. HELIOS++ is the successor of HELIOS (Bechtold and Höfle, 2016), with a completely new code base, implemented in C++ (whereas the former version was implemented in Java). HELIOS++ improves over HELIOS in terms of memory footprint and runtime, functionality, correctness of implemented algorithms, and usability, making it versatile and highly performant to users.

We first motivate the need for a novel VLS framework by a survey of previous approaches to laser scanning simulation and their applications, and point out the unique features of HELIOS++ (Section 2). In Section 3, we present the architecture, design considerations, and implementation details of HELIOS++ that are relevant for scientific use. We then show different types of applications and conduct a systematic literature survey of uses of HELIOS in Section 4. A comparison of HELIOS++ with HELIOS and the state-of-the-art Monte-Carlo LiDAR simulator DART is provided in Section 5, and conclusions are drawn in Section 6.

2. Existing implementations and state-of-the-art virtual laser scanning

Simulating a process within a system always involves a simplified substitute of reality. The complexity of this substitute depends on the process understanding, on computational considerations, and on the specific problem that is to be solved. Approaches with different levels of simulation complexity exist regarding (i) the type of input scene model (e.g., 2.5D digital elevation models (DEMs) vs. 3D meshes) and (ii) how the interaction of beam and object is modelled (e.g., single ray/echo vs. full-waveform). An overview of publications of these approaches is listed in Table 1. The overview is a result of a literature survey involving the simulation of LiDAR, especially targeting vegetation, and aims to represent at least one of each of the complexity levels.

A simple airborne laser scanning (ALS) simulator is presented by Lohani and Mishra (2007) for use in research and education. Their tool comes with a user-friendly graphical user interface (GUI) and allows selecting different scanner and trajectory configurations. This simulator

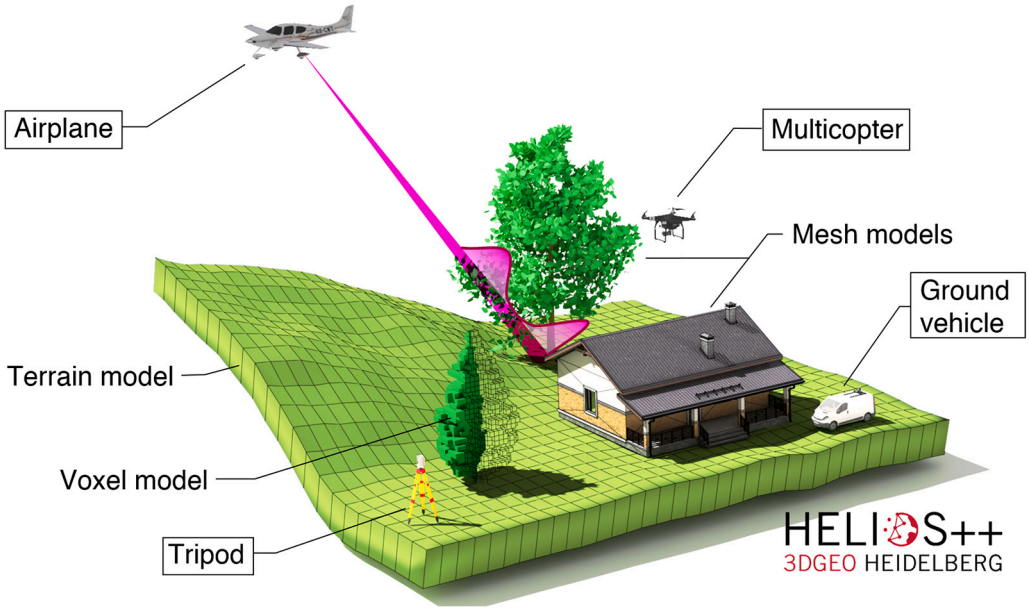

Fig. 1. Schematic concept of HELIOS++, showcasing platforms (boxed labels) and object models composing a scene (non-boxed labels). A variety of model types to represent 3D scenes are supported: terrain models, voxel models (custom .vox format or XYZ point clouds) and mesh models. As platforms, four options are currently supported: airplane, multicopter, ground vehicle and static tripod. A schematic diverging laser beam and its corresponding waveform (magenta) is shown being emitted from the airplane and interacting with a mesh model tree and the rasterised ground surface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

[1] HELIOS++ is available on GitHub (https://github.com/3dgeo-heidelberg/helios) and is licensed under both GNU GPL and GNU LGPL. HELIOS++ is also indexed with Zenodo (Winiwarter et al., 2021).

Table 1
Overview of virtual laser scanning simulators and associated publications. For each simulator, a check mark (✓) is added if they support simulation of finite (non-zero) beam divergence ("Beam div.") and full waveforms ("FWF"). Scene representation may be in full 3D or 2.5D, i.e., raster-based.

Publication	Platforms	Beam div.	FWF	Scene	Comments
North (1996), North et al. (2010)	satellite	✓	✓	3D	FLIGHT model
Lewis (1999)	ALS	✓	✓	3D	librat model used by Calders et al. (2013) & Disney et al. (2010)
Tulldahl and Steinvall (1999)	ALS	✓	✓	3D	for bathymetry
Ranson and Sun (2000)	ALS (nadir)	✓	✓	3D	
Holmgren et al. (2003)	ALS			3D	
Agrawal et al. (2004)	ALS			2.5D	
Lovell et al. (2005)	ALS			3D	
Goodwin et al. (2007)	ALS			3D	LITE model
Lohani and Mishra (2007)	ALS			2.5D	
Morsdorf et al. (2007)	ALS	✓	✓	3D	using POVray
Kim et al. (2009)	ALS			3D	
Kukko and Hyyppä (2009)	ALS, MLS	✓	✓	2.5D	
Hodge (2010)	TLS	✓	✓	2.5D	
Kim et al. (2012)	ALS	✓	✓	3D	
Wang et al. (2013)	TLS			3D	
Gastellu-Etchegorry et al. (2015)	satellite, ALS, TLS	✓	✓	3D	DART model
Bechtold and Höfle (2016)	ALS, TLS, MLS, ULS	✓	✓	3D	HELIOS
Dayal et al. (2021)	ALS			3D	Limulator 4.0

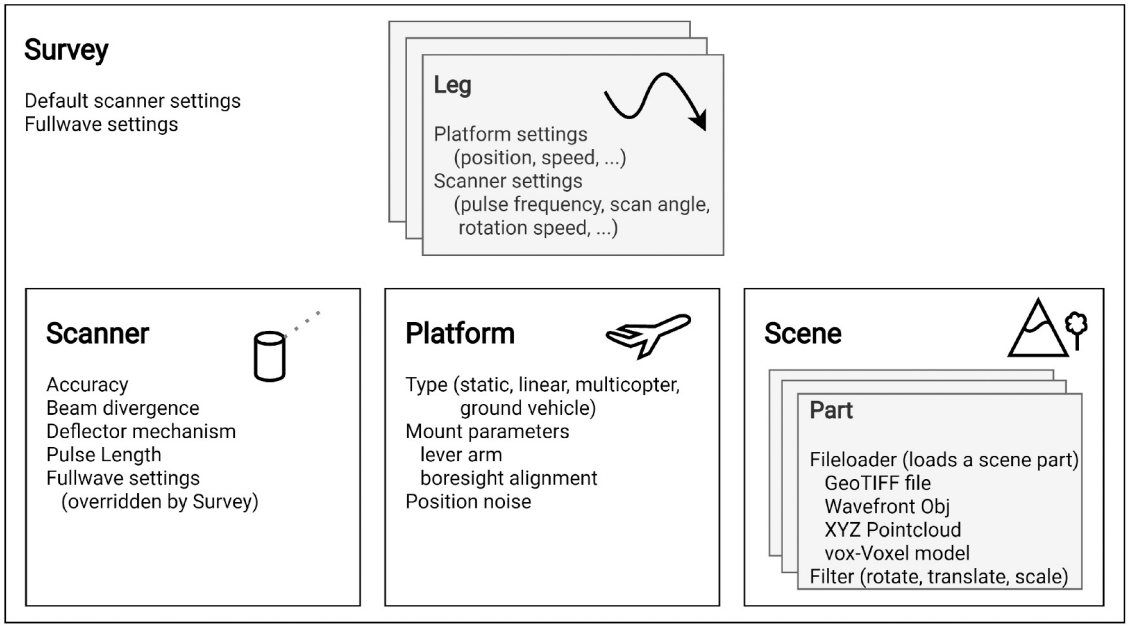

Fig. 2. Main components and file structure of HELIOS++ survey, scene, platform, and scanner. A survey consists of one or more legs, and a single scanner, platform, and scene, respectively. A scene is built up from one or more parts, which can be of different data type.

models the laser ray as an infinitesimal beam with zero divergence to simplify the ray tracing procedure. The scene is represented by a 2.5D elevation raster, which allows only a simplified representation of the Earth's surface.

More recently, Dayal et al. (2021) also focus on a positive user experience by providing a GUI to a general-purpose laser scanning simulator working with arbitrary 3D meshes as input data. They present their implementation for flight planning and simulation of photos, however also point out the lack of beam divergence modelling.

In a different application, the interactions between laser beams and forest canopies are investigated by Goodwin et al. (2007), Holmgren et al. (2003) and Lovell et al. (2005). They present approaches of combining 3D forest modelling and ALS simulation using ray tracing. As in Lohani and Mishra (2007), all these simulators model the laser beam as an infinite straight line, which intersects the scene in one distinct point.

In reality, laser beams are diverging conically, and full-waveform laser scanners can record the full waveform of the backscattered signal, providing information about the objects in a scene which are illuminated. This waveform from a finite footprint was specifically simulated by Ranson and Sun (2000) in the forestry context, and by Tulldahl and Steinvall (1999) for airborne LiDAR bathymetry. Morsdorf et al. (2007) use the ray tracing software *POV-Ray*[2] to model the waveform of laser scans of a 3D tree model from a combination of intensity and depth images.

Kukko and Hyyppä (2009) aim for a more complete and universal LiDAR simulator that considers platform and beam orientation, pulse transmission depending on power distribution and laser beam divergence, beam interaction with the scene, and full-waveform recording. Their approach models the physics involved in LiDAR measurements in high detail, and is demonstrated for a use case in forestry. Similar to the

[2] http://www.povray.org/

work by Lohani and Mishra (2007), they use 2.5D elevation maps to represent the scene. This makes the simulator useful for airborne simulations over terrain or building models, where the scene is scanned from above and scene elements are assumed to be solid and opaque. However, it is less suited for penetrable 3D objects such as vegetation or overhanging geometries, especially for the simulation of ground-based acquisitions which are less in a bird's-eye view perspective. Kim et al. (2012) present a similarly detailed simulator, which includes radiometric simulation and recording of the waveform as well as explicit 3D object representations. It is unclear if it also supports static platforms, such as TLS.

TLS simulations are the sole focus of some studies for specific applications, such as leaf area index inversion (Wang et al., 2013) or TLS measurement error quantification (Hodge, 2010). Wang et al. (2013) use a more simple model assuming no beam divergence, whereas the simulation described by Hodge (2010) includes both the modelling of beam divergence and recording of the waveform. Their simulation again uses 2.5D elevation models to represent the scene, which is appropriate for their particular objective of error quantification in high-resolution, short-range TLS of natural surfaces, specifically fluvial sediment deposits, of small scenes (area of 1 m $\times$ 1 m).

Established Monte-Carlo ray tracing simulators are being used and extended for airborne and satellite laser scanning simulations. The *librat* model (Calders et al., 2013; Disney et al., 2009, 2010), a modular development of ARARAT (Lewis and Muller, 1993), is such a Monte-Carlo simulator. Similarly, North et al. (2010) extend the 3D radiative transfer model FLIGHT (North, 1996) to model satellite LiDAR waveforms. Monte-Carlo methods represent a simple, robust, and versatile set of techniques to solve multi-dimensional problems by repeatedly sampling from a probability density function describing the system that is investigated (Disney et al., 2000). These stochastic methods are useful for simulating multi-scattering processes, e.g., for modelling canopy reflectance. The main drawback of Monte-Carlo ray tracing methods are high computation times to simulate sufficient photons for the scattering model to converge to an accurate solution (Disney et al., 2000; Gastellu-Etchegorry et al., 2016). The LiDAR extension of the Discrete Anisotropic Radiative Transfer (DART) model attempts to alleviate these restrictions by quickly selecting scattering directions of simulated photons using the so-called Box method and modelling their propagation and interaction using a Ray Carlo method, which combines classical Monte-Carlo and ray tracing methods (Gastellu-Etchegorry et al., 2016).

A comprehensive review of simulators for the generation of point cloud data, including LiDAR simulators, is presented in Schlager et al. (2020) with focus on their applicability to generate data in the context of driver assistance systems and autonomous driving vehicles. In this context, they analyse algorithms with respect to their fidelity, operating principles, considered effects and possible improvements.

HELIOS++ provides a framework for full 3D laser scanning simulation with multiple platforms, i.e., terrestrial (TLS), mobile (MLS), UAV-borne (ULS) and airborne (ALS), and a flexible system to represent scenes, which allows combination of input data from multiple sources and data formats (Fig. 1). The simulation of beam divergence and full waveform recording are supported. Although HELIOS++ may not be as realistic in terms of physical accuracy regarding the energy budget of a single laser shot as, e.g., DART, it provides an easily configurable trade-off between computational efforts and resulting point cloud quality. Users can simulate VLS over a large range of scales, and even combine different scales in one scene. For example, a highly detailed tree model with individual leaves might be placed in a forest scene represented by (transmissive) voxels (Weiser et al., 2021), while using a rasterised digital terrain model as ground surface. This allows to model the influence of the surrounding of a particular object of interest on the derived VLS point clouds. Furthermore, HELIOS++ aims for high usability by providing a comprehensible set of parameters to control the LiDAR simulation, without overwhelming the users with options. Since HELIOS++ can be used from the command line and within Python, workflows integrating HELIOS++ can be easily scripted and automated, as well as linked to external software (e.g., GIS, 3D point cloud processing software, Jupyter Notebooks, and others).

HELIOS++ comes with an extensive documentation of all algorithms that are used in the simulations, including examples and references to the relevant literature describing the implemented methods. Furthermore, the open-source implementation allows any user to i) inspect and ii) alter/adapt the source code of the program. For ease of use, we provide pre-compiled versions that are ready for use on major operating systems (Microsoft Windows 10 and Debian Linux 10.7 Buster), as well as the option to use Python as a scripting language to create, manipulate, and simulate VLS data acquisition with HELIOS++.

3. Implementation of HELIOS++

This section introduces the concepts and interfaces of HELIOS++. We focus on implementation details which are relevant for scientific use and interpretation of the generated VLS data. The important components are the overall architecture and modules (Section 3.1), platforms (Section 3.2), scanners and laser beam deflectors (Section 3.3), the waveform simulation (Section 3.4), interaction between beam and objects (Section 3.5), input (Section 3.6) and output formats (Section 3.7), and the aspect of randomness and repeatability of results (Section 3.8).

3.1. Architecture and modules

The central element in HELIOS++ simulations is a *survey*. A survey contains links to the *scene*, which defines the objects that are scanned, the *platform*, on which the virtual scanner is mounted and moved through the scene, and to the *scanner* itself. Furthermore, a survey contains a number of *legs*, which represent waypoints for the platform. The scene consists of a number of *parts*. Each part represents one input object, for example, a 3D mesh file or a voxel file. Multiple parts may be combined in a scene, and there is no limitation to the combination of different data source types. HELIOS++ uses internationally accepted standard file formats for input and output. These elements are defined through Extensible Markup Language (XML) files, which are referenced using (relative) file paths. XML is a text-based format, which can easily be manipulated using a text editor or an XML editor. Fig. 2 presents the different files along with a subset of the parameters that can be set in the respective files.

A survey may be run through either (i) the command line, where an executable is provided or (ii) the Python bindings `pyhelios`. The Python bindings allow access to the simulation parameters as parsed from the XML files, including changing the parameters programmatically for each simulation run. For example, different scanners can be exchanged automatically and simulated in sequence in a single Python script without changing any other input and settings of the simulation. `pyhelios` furthermore allows accessing the simulation result, i.e., the point cloud and the platform trajectory, and converting them into a NumPy array either at the end of the simulation run or through a callback function that can be executed every *n*-th cast laser ray. In this way, a live preview of the point cloud acquisition can be implemented in Python to give a visual impression of the ongoing simulation. A Python script distributed with HELIOS++ acts as such a visualiser, and is called with the same commands as the standalone executable. A user may thus quickly switch between using the pure C++ implementation without visualisation or the Python bindings with visualisation, as presented in Listing 1.

```
run\helios.exe data\surveys\arbaro_demo.xml --lasOutput --writeWaveform
python pyhelios\helios.py data\surveys\arbaro_demo.xml --lasOutput --writeWaveform
```

Listing 1. Comparison between running HELIOS++ as an executable and through the Python wrapper providing an interactive viewer.

```
import pyhelios
# define callback function.
def callback(output=None):
    # store trajectories in variable.
    trajectories = output.trajectories
    # update plot.
    update_plot(trajectories)

# Build simulation parameters: (surveyPath, assetsPath, outputPath, ...).
sim = pyhelios.Simulation(
    'data/surveys/' + survey_path,
    'assets/',
    'output/',
    0,          # Num Threads - 0 by default
    True,       # LAS v1.4 output - False by default
    False,      # LAS v1.0 output - False by default
    False,      # ZIP output - False by default
    )
# Enable final output.
sim.finalOutput = True
# Set sim frequency.
sim.simFrequency = 10000 # run callback every 10k shots
sim.loadSurvey()
sim.setCallback(callback)
sim.start()
```

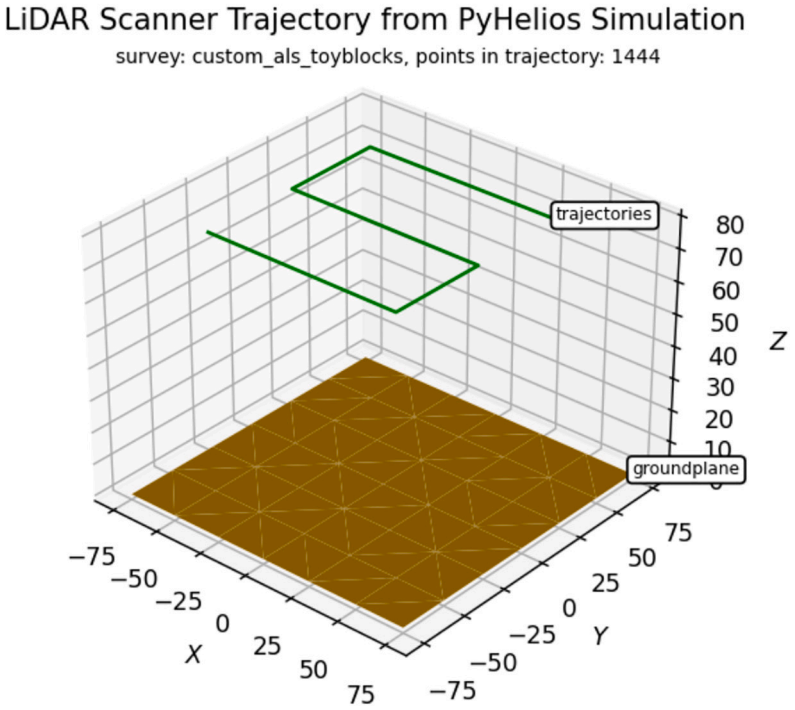

Fig. 3. Screenshot of a Jupyter Notebook showcasing the Python bindings of HELIOS++ by plotting the trajectory of a simulation over flat terrain.

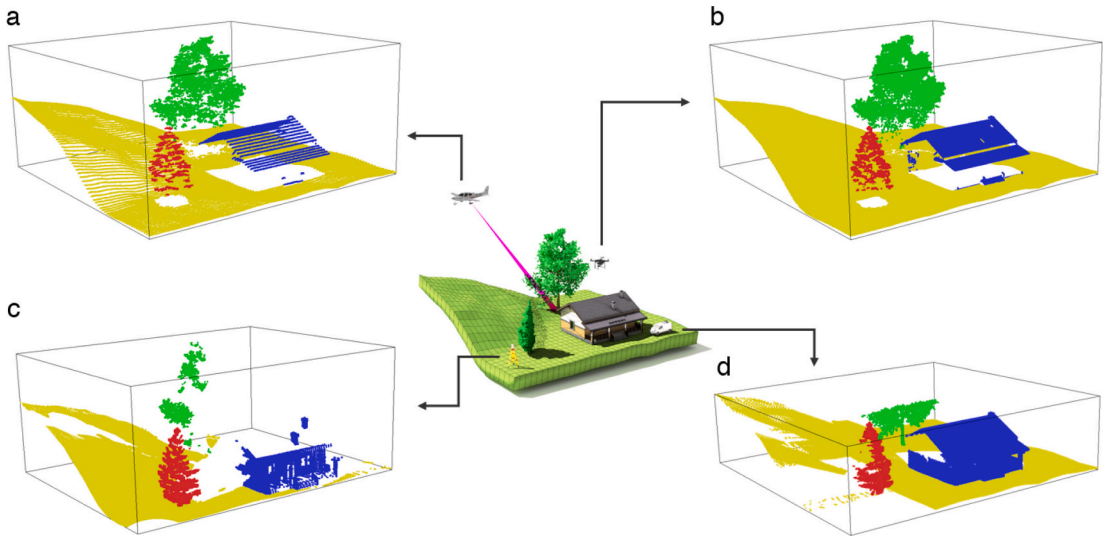

Fig. 4. Point clouds resulting from virtual laser scanning of the scene shown in Fig. 1, using (a) an airplane, (b) a multicopter, (c) a static tripod, and (d) a ground vehicle as platform. As HELIOS++ records which objects are generating which return, the points can be perfectly assigned to the objects, here illustrated by distinct colouring. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

With `pyhelios`, it is also possible to combine HELIOS++ with tools like *Jupyter Notebooks*, allowing for explanations alongside code and figures. We include sample notebooks in the documentation of HELIOS++. One of these samples is shown in Fig. 3.

3.2. Supported laser scanning platforms

Both static and dynamic platforms are supported by HELIOS++, which can resemble an airplane (`LinearPath` platform), a multicopter (`Multicopter` platform), a ground-based vehicle (`GroundVehicle` platform) or a static tripod (`Static` platform). In the case of the `LinearPath` platform, the vehicle is moved with a constant speed from one waypoint (leg) to the next one. The orientation of the platform is always towards the next waypoint. The `Multicopter` platform additionally simulates acceleration and deceleration of the platform. In the turn mode *smooth*, the platform banks to make more smooth turns at the waypoints instead of stopping and turning on the spot. This mode simulates *banked angle turns* available in the flight protocols of major drone companies (e.g., DJI). Custom yaw angles for the beginning and the end of each leg can be provided. The `GroundVehicle` platform is bound to elements of the scene defined as ground in the respective material file (cf. Section 3.6). Furthermore, it considers maximum turn radii and implements three-point-turns to resemble a car or tractor on which a scanner is mounted (i.e., MLS). The different platforms and scene types (Section 3.6) are shown in Fig. 1. As expected, choice of platform influences the resulting point cloud, which is shown in Fig. 4.

3.3. Supported laser scanners and scan deflectors

The core component of the simulation is the laser scanner, which includes a model for the scan deflector. The choice of the deflector influences the resulting scan pattern (Fig. 5).

HELIOS++ supports polygonal mirror deflection, resulting in parallel scan lines with even point density (Fig. 5a), or a swinging mirror, which results in a zig-zag-pattern with increased point densities at the extrema (Fig. 5b). Fibre-optics, where the beam is fed through fibre-optic cables to point in different directions, result in a similar scan pattern to (a), but without the need of mechanically moving parts (Fig. 5c). It further supports rotating slanted mirrors, resulting in a conical point pattern at a constant scan angle off-nadir, also referred to as Palmer scanner (Fig. 5d), and Risley prism scanners, as recently manufactured by DJI in their Livox series (Fig. 5e). These deflectors create a pattern that samples the surface more densely with longer integration times (Duma and Schitea, 2018).

3.4. Waveform simulation

During real LiDAR acquisitions, a laser beam sent out from the laser scanner has a finite footprint, i.e., a non-zero area that intersects with the scene. For every infinitesimal point in this area, energy is scattered back to the detector where it is recorded as the integral of instantaneous power over the illuminated area. By considering this received power over time, it is possible to extract multiple echoes from one laser pulse, corresponding to multiple targets that were hit by parts of the intersection area, respectively, which can be modelled in HELIOS++. Furthermore, new sensor systems, such as Geiger mode LiDAR and Single Photon LiDAR, have been presented. Their simulation however requires tracing of individual photons and is therefore out of scope for HELIOS++.

In HELIOS++, the non-zero beam divergence is simulated by subrays, that are sampled in a regular pattern around the central ray (Fig. 6). Every subray has its own base power, which is calculated according to Eq. (1), representing a 2D Gaussian power distribution. For details on the derivation of the formulae in this section, the reader is referred to Carlsson et al. (2001). Note, that in contrast to Carlsson et al. (2001), we use β and β_0 as angular helper values, in order to make them better distinguishable from w and w_0.

$$I = I_0 exp\left(-2r^2/w^2\right) \tag{1}$$

Here, I_0 [W] is the peak power, w [m] the beam divergence and r [m] the radial distance from the power maximum, i.e., the ray centre. Note that w changes with the range R because the beam is not perfectly focused. This beam divergence is calculated using Eq. (2) from the beam waist radius w_0 [m], and the helper values $\beta = \frac{\lambda R}{\pi w_0^2}$ and $\beta_0 = \frac{\lambda R_0}{\pi w_0^2}$ with λ [nm] as the wavelength, R [m] the range to the target and R_0 [m] the focusing length of the laser.

$$w = w_0\sqrt{\beta_0^2 + \beta^2} \tag{2}$$

Every subray is individually cast into the scene and intersected with objects. If it hits an object, a return is generated and recorded by the detector. The respective subray does not continue in the scene through

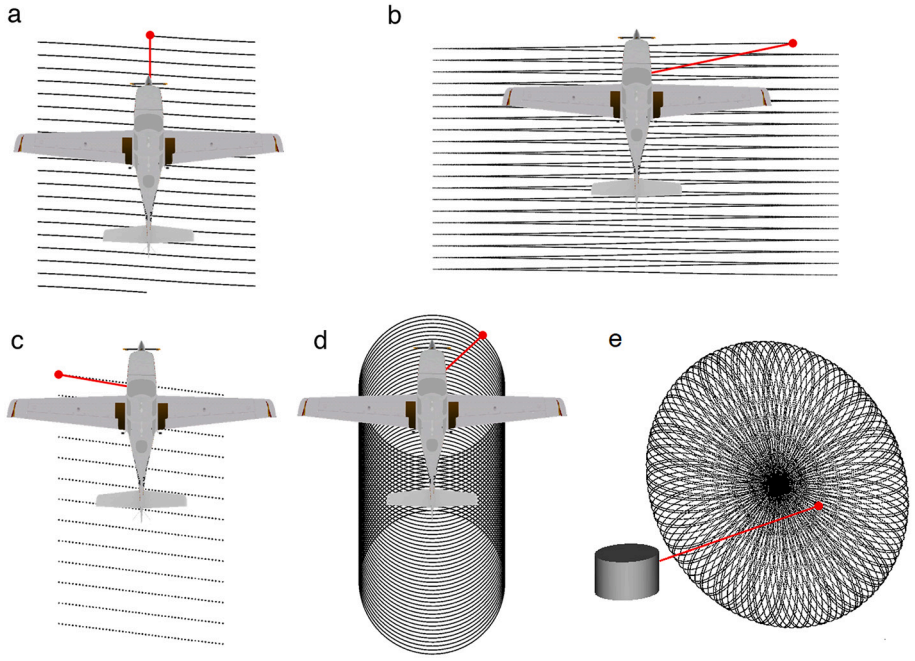

Fig. 5. Different scan patterns depending on the deflector used in the simulation: (a) rotating mirror, (b) oscillating mirror, (c) fibre-optic line scanner, (d) slanted rotating mirror (Palmer scanner), and (e) Risley prisms (i.e., Livox scanner). The patterns result from simulations with HELIOS++ using the respective deflectors, albeit with unrealistic settings of pulse repetition rate and scanning frequency, in order to show the patterns more clearly.

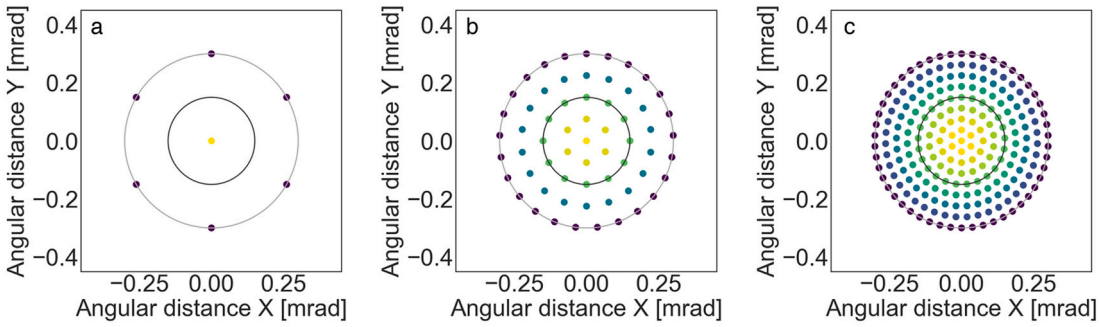

Fig. 6. Subray configurations for beam sample qualities of (a) 2 (7 subrays), (b) 5 (62 subrays), and (c) 9 (223 subrays). The colour of the subrays corresponds to the normalised power at this location within the beam cone: low (purple) to high (yellow). The black circle represents the single beam divergence (at the $1/e^2$ points, here: 0.3 mrad), the gray circle twice the beam divergence. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

transmission or reflection. In the case of the transmissive voxel model (cf. Section 3.6), a subray may either fully traverse a voxel or produce a return, but never both. Therefore, when using transmissive voxels, multiple subrays must be employed to take full advantage of this probabilistic model. The power returned from the object is further dependent on the material specified in the scene definition (Section 3.6).

The number of subrays generated can be set by the user by providing the `beamSampleQuality` parameter in the XML file of the scanner or the survey. The `beamSampleQuality` corresponds to the number of concentric circles where subrays are sampled. For each circle, the number of subrays is defined as $\lfloor 2\pi i \rfloor$ where $i = 1, \ldots,$ `beamSampleQuality` is the circle index. This ensures that the angular distance between adjacent subrays is approximately constant, i.e., each subray represents a solid angle of equal size. A central subray is always added. Fig. 6 shows the subray distribution for three different beam sample qualities. Note that the total area covered by the subrays is always the same, i.e., more subrays do not extend the area that is illuminated, but sample it in more detail. Different choices for subsampling, e.g., random or with a higher concentration of subrays towards the beam centre are imaginable, but have not been implemented so far.

The pulse shape in time is approximated using bins of a regular, user-defined size via the parameter `binWidth_ns`. Each bin's power is calculated according to Eq. (3), where t [ns] is the time and τ [ns] is the pulse length (half width of half maximum) of the scanner divided by 1.75 (as given by Carlsson et al., 2001, Eq. 2.1). I [W] is calculated for each subray according to Eq. (1).

$$P(t) = I\left(\frac{t}{\tau}\right)^2 exp\left(-\frac{t}{\tau}\right) \tag{3}$$

3.5. Ray tracing and beam scattering

Since modern laser scanners can measure millions of points per second, it is crucial to have an efficient implementation for the ray-scene interaction.

In HELIOS++, a scene S can be mathematically described as a set of primitives, so $S = \{P_1, \ldots, P_n\}$. For each primitive P, its boundaries are defined considering an axis-aligned bounding box, its centroid, and the set of vertices $V = \{v_1, \ldots, v_m\}$ composing the primitive. Each primitive

supports rotation, scaling and translation together with a material specification defining its reflectance and specularity. Certain primitives, such as transmissive voxels, also support a look-up table which can be used for vegetation modelling, as explained in Section 3.6.4.

The scene building process consists in generating the set of primitives composing the scene. Multiple input objects are supported (cf. Section 3.6), so it is possible to build a scene considering Wavefront Object files, point clouds as ASCII files specifying (X, Y, Z) coordinates for each point, GeoTIFF files, or a custom voxel file format based on AMAPVox (Vincent et al., 2017). When building a scene, different object types can be considered, as each one is associated to its own scene part. It is also possible to apply aforementioned affine transformations to an entire scene part. This can be used, for instance, to load the same object multiple times in one scene and placing it in different locations through translations. Saving and loading already built scenes is implemented by converting the full scene including its objects to a serialised representation (i.e., a representation that can be written to disk; Ramey, 2004). Objects appearing multiple times in the scene are only serialised once. Any subsequent run (e.g., using a different platform or different scanner settings) can then be carried out by re-loading the already built scene.

Ray intersections are computed through a recursive search performed over a kD-Tree containing all primitives (Bentley, 1975). Let O be the ray origin and $\hat{v}$ the normalised ray direction vector. When recursively searching through the kD-Tree starting at O, $\hat{v}$ is used to consider which node must be visited until a leaf node is reached. Once inside a leaf node, ray intersections with respect to each primitive are computed. The minimum distance intersection t_0 is the time the ray needs to enter the primitive, and t_1 is the time the ray needs to leave it. It is possible to have only t_0 determined, as is the case for triangles, since they are only intersected once per ray.

For the special case of primitives which support multiple ray intersections, such as transmissive voxels, the process is repeated considering consecutive origins. Suppose we have a transmissive voxel intersected by a ray $\{O_1, \hat{v}\}$: If this voxel lets the ray pass through it, then the next ray intersection will be found considering the ray $\{O_2, \hat{v}\}$, with $O_2 = O_1 + (t_1 + \varepsilon)\hat{v}$, where ε is a small decimal number to assure exiting the previously intersected primitive.

If a hit of a subray with any object is recorded, the distance between the sensor and the hit is determined. From the minimum and maximum distances for all hits (i.e., subrays producing an echo) within a single pulse, an array for storing the full waveform is created and initialised with all zeros. For each hit, the backscattered power is evaluated, either according to Phong's bidirectional reflectance distribution function (Tuong-Phong, 1973; Jutzi and Gross, 2009), which takes the incidence angle and material properties (diffuse and specular scattering coefficients, reflectance) into account, or following the power derivation for transmissive objects, as presented in Section 3.6.4, Eq. (7). Note that the incidence angle only influences the backscattered power, but does not scale the recorded waveform in time. This is because elongated echoes are produced by adding multiple subrays, each with a slightly different range.

The outgoing pulse (Eq. (3)) is then multiplied by this power and added to the corresponding bins of the pre-initialised array, ensuring that the maximum power bin is shifted according to the different ranges of the subrays. Range difference is converted to time difference by the speed of light as a constant ($c = 299{,}792{,}458$ m s^{-1}). To simulate point detection, a local maximum filter is applied at the locations of the hits on the summed waveform to detect peaks. If there are no stronger signals within the filter window, the peak is recorded as an echo and exported as a point. Optionally, a Gaussian may be fitted to the resulting waveform, to get a measure of echo width (i.e., standard deviation of the Gaussian). The position of the point along the range (time) axis, however, is taken from the originally recorded distance, not from the fitted Gaussian, because the outgoing waveform (Eq. (3)) is not Gaussian. This is a quite simple approach to detect echoes from the waveform, which produces realistic point clouds in many applications. Alternatively, users may use the exported full waveform files to implement their own echo detection, e.g., based on Gaussian decomposition or deconvolution (Wagner et al., 2007).

Multi-spectral LiDAR, where laser scanner beams of different wavelengths are employed simultaneously, can be simulated in HELIOS++ by repeating a survey with different settings. The wavelength, set in the scanner configuration, changes the material-dependent reflectance (used in Phong's bidirectional reflectance distribution function).

3.6. Input scene models

HELIOS++ supports data formats that are (de-facto) standards in the field, and can be created and manipulated with free software such as QGIS[3], CloudCompare[4], Blender[5] or AMAPVox[6].

3.6.1. Wavefront objects

In Wavefront Object files (file extension .obj), meshes are represented as lists of triangles (triples of point IDs) and points (triples of coordinates). This allows opaque 3D models of arbitrary complexity. In addition, material properties can be assigned to the objects in so-called material files (file extension .mtl; Wavefront Technologies, 1992).

3.6.2. GeoTIFF models

Raster models created with common GIS tools, such as digital elevation models or digital surface models, can be included. The raster is converted to a triangular mesh on import, interpreting the pixel centres as points. Invalid pixels (i.e., no-data values) are ignored in the triangulation, resulting in holes in the mesh. This data type allows simple representation of topography. As prescribed by the file format, only 2.5D data is supported.

3.6.3. Point clouds

In ASCII XYZ files, point clouds can be used as a raw input to HELIOS++. On import, the point clouds are voxelised using a voxel size defined by the user. This voxel model is a pure occupancy grid, i.e., a single point within a voxel cell suffices to make the voxel appear fully opaque in the simulation. In addition, a normal vector used in the angle-dependent reflectance determination of the subrays can be assigned to the voxels (by nearest neighbour or mean reducing), or calculated from the point cloud on-the-fly. For this calculation, a singular value decomposition (SCD, Golub and Kahan, 1965) is employed. Material properties can be defined in the respective scene part definition of the XML file.

When loading point clouds from ASCII XYZ files, big files might not be entirely containable in memory. For this purpose, a two-stage algorithm is used to digest point clouds of arbitrary size. Voxels which have points inside them are then built as HELIOS++ primitives, whereas empty voxels are discarded. The normal estimation, if required, is performed in batch mode, dividing the workload into smaller parts. Each batch extracts points inside its voxels while ignoring points outside its scope.

3.6.4. Voxel models

Especially for modelling vegetation, we include support for voxel data containing plant area density information, e.g., created using the AMAPVox software (file extension .vox, Vincent et al., 2017). Multiple modes are supported in this case:

[3] https://qgis.org/en/site/
[4] https://www.cloudcompare.org/
[5] https://www.blender.org/
[6] Vincent et al. (2017), http://amap-dev.cirad.fr/projects/amapvox

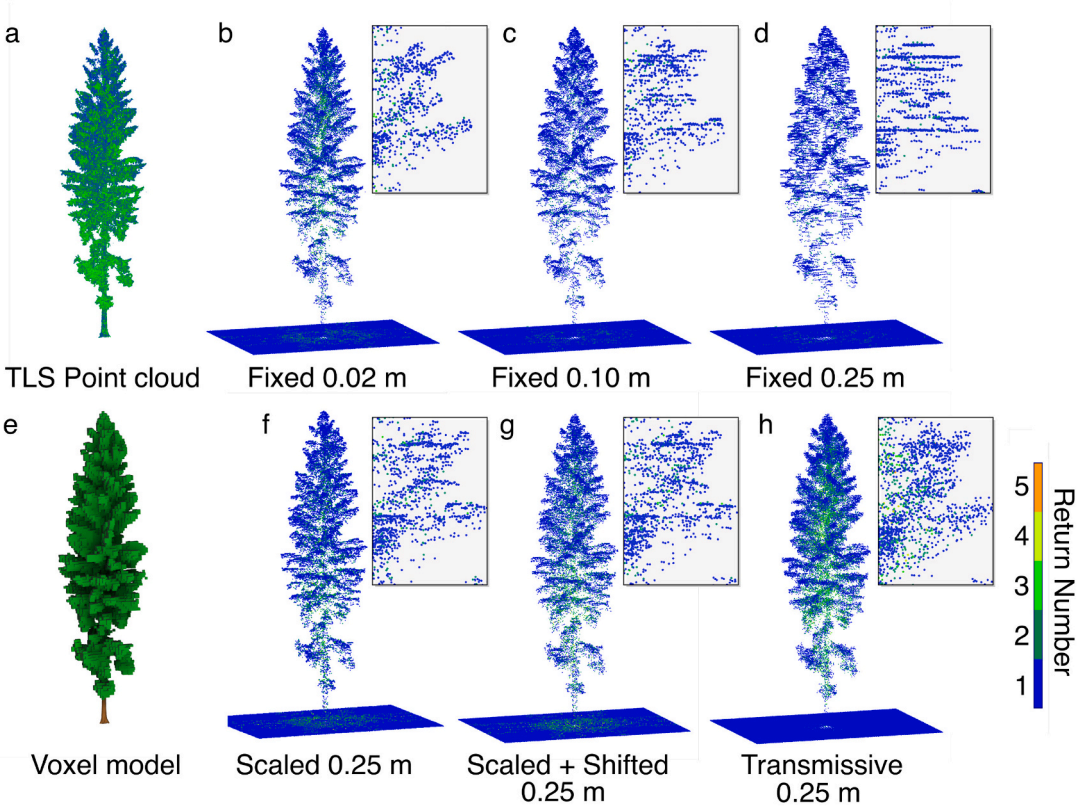

Fig. 7. VLS point clouds of a tree based on different voxel modes, using a UAV as platform. The high-density input point cloud, coloured by signal strength (a) and the voxel model at 0.25 m side length (e) are shown on the left. Point clouds are simulated using input voxel models with different fixed sizes (b–d), PAD-dependent scaled voxels (f and g) and transmissive voxels (h), and are coloured by their return number. The insets show a detail of two branches for the respective models. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- opaque voxels, where the voxels are represented by solid cubes with a fixed side length equal to voxel resolution,
- adaptive scaling of opaque voxel cubes with optional random but reproducible shift to avoid regular patterns in the result, and
- transmissive voxels.

The latter two are explained in more detail in the following. Point clouds resulting from a tree simulation using these different modes are shown in Fig. 7.

The opaque voxel models also support the definition of material properties through a scene part's definition. A comprehensive study on the effects of different levels of detail in modelling forests using opaque and scaled voxels in VLS for the extraction of forestry and point cloud parameters is performed by Weiser et al. (2021).

Adaptive scaling

In the adaptive scaling mode, the side length of a voxel is dependent on the plant area density, related by Eq. (4):

$$a = a_0 \left(\frac{\mathrm{PAD}}{\mathrm{PAD}_{\max}} \right)^{\alpha} \tag{4}$$

Here, a is the side length of the resulting cube, a_0 the base voxel size, PAD the plant area density of each individual voxel, PAD_{max} a maximum value for the plant area density and α a user-defined scaling factor. For $\alpha > 0$, a larger PAD will hence lead to larger voxels, which in turn represent less penetrability in the voxel grid space. With α set to 0.5, the PAD is proportional to the surface area of the voxels.

Transmissive voxels

The transmissive voxels use an extinction approach as presented by North (1996). The extinction coefficient σ is calculated following Eq. (5):

$$\sigma = \frac{\mu_L}{2\pi} \int_0^{2\pi} g_L |\Omega' \cdot \Omega_L| d\Omega_L \tag{5}$$

Here, μ_L is the plant area density (PAD) as defined in the.vox file, g_L is the probability taken from the leaf angle distribution at direction L, and $|\Omega' \cdot \Omega_L|$ is the cosine of the angle between the incident ray Ω' and the direction Ω_L. This leaf angle distribution is supplied via a look-up-table as shown in Table 2, and examples for planophile, erectophile, plagiophile, extremophile, spherical and uniform distributions are provided.

Subsequently, a random number R is drawn from a uniform distribution $\in[0, 1)$. The expectation of the intersection of a subray with a voxel given the extinction coefficient σ is then given as in Eq. (6), where s is the distance of an echo after entering the voxel (i.e., traversed distance).

$$s = \frac{-ln(R)}{\sigma} \tag{6}$$

Table 2
Values from a look-up table (LUT) for the hit probability g_L at a given beam direction L, represented by horizontal and vertical component. The numbers here correspond to an erectophile distribution and are obtained using a numerical integration method on the formulae from North (1996). The probability g_L is normalised over 2π.

Beam horizontal component	Beam vertical component	Hit probability g_L
1.000000	0.000000	0.424413
0.999683	0.025180	0.424682
0.998732	0.050345	0.425489
⋮	⋮	⋮
0.009444	0.999955	0.848789
0.000000	1.000000	0.848822

If the calculated distance to the hit s is larger than the actual path of the beam in the voxel no echo is recorded for this ray in the voxel, and it is assumed to have transmitted through the voxel, potentially scattering in the next voxel or at another object. Otherwise, an echo is created at distance s from where the subray first entered the respective voxel, and the subray is not continued through the scene (North, 1996).

The received power of a subray P_R is calculated according to Eq. (7) where P_E is the emitted power from Eq. (3) (Carlsson et al., 2001), R is the range between ray origin and intersection point, α^2 is the square of the scanner receiver diameter, β^2 is the square of the scanner beam divergence, and λ is the product between atmospheric transmission factor and scanner efficiency. The σ value can be seen as function of ray direction vector $\sigma(\hat{v})$, so it will have a different value depending on ray incidence.

$$P_R \propto \lambda \sigma \frac{P_E \alpha^2}{4\pi R^4 \beta^2} \tag{7}$$

By using detailed voxels operating in transmissive mode together with an appropriate look-up table specification, HELIOS++ is thus capable of simulating laser scanning of vegetation from a precomputed leaf angle distribution, following North (1996). To achieve high-quality output, it is recommended to apply individual leaf angle distributions for each vegetation type within a scene.

Irrespective of the input type, each scene part can be individually scaled, rotated and translated within the scene, allowing the re-use of models to create multiple instances of similar objects. These transformations can also be manipulated using the Python bindings. For rotations, both extrinsic ("global", fixed coordinate axes) and intrinsic ("local", rotating coordinate axes) modes are supported.

HELIOS++ allows arbitrary combinations of all possible object models in a scene. For example, a GeoTIFF can be used as digital terrain model on which a vehicle is moving, combined with transmissive voxel models of trees, mesh models for tree stems and a point cloud of buildings, which is voxelised by HELIOS++ (Fig. 1). Such combination allows highly versatile use of the scene definition, while considering shadowing and overlapping effects between the different types of object models. This is essential for a general-purpose LiDAR simulator, as we do not know the prerequisites of users.

3.7. Output format and options

The simulated point clouds can be written either as LAS file (Version 1.4, point format 6), according to the format definition by the American Society of Photogrammetry and Remote Sensing (ASPRS, 2011), as LAZ file, which is a lossless compression of LAS provided by the LASzip library[7], or as ASCII file, where the point coordinates and attributes are written in columns. The LAS/LAZ formats allow for smaller file sizes and faster read/write access (cf. Section 5), while the ASCII format can be parsed by any program working with text files.

In addition to creating compressed LAS files, HELIOS++ can also compress output ASCII files, if smaller output sizes are required. Uncompressing LAZ or compressed ASCII files is possible by invoking HELIOS++ using the `-unzip` flag.

In the simulation of full waveforms (see Section 3.4), the echo width is optionally estimated for every recorded return. Since this estimation uses a non-linear least squares method and is hence expensive to calculate, the estimation has to be switched on explicitly by the user. In typical use cases, this increases the runtime of the simulation by around 30%.

If the full waveform is to be exported, a separate ASCII file will be written. This file contains information about every beam that generated at least one hit consisting in the beam origin and the beam direction, and the sampled returned waveform. The bin size as well as the maximum length of this sampled waveform can be defined by the user (Section 3.4). To connect the point cloud output with the full waveform output, the point cloud contains an attribute `fullwaveIndex`, which corresponds to the `fullwaveIndex` in the ASCII waveform output. Multiple points, if originating from the same beam, have the same `fullwaveIndex`. The output created by HELIOS++ can be easily converted to standard waveform formats such as PulseWaves and LAS 1.4 WDP[8].

Furthermore, HELIOS++ exports the trajectory of the platform. The time interval between successive platform positions can be defined by the user in the survey XML file. In addition to the time and the position of the platform, the attitude angles of roll, pitch and yaw are exported. Time is in the same format as the *GPSTime* field in the point cloud data and corresponds to simulated seconds since midnight of the previous Sunday (GPS time), relative to the time of starting the simulation.

3.8. Randomness and repeatability

The trajectory of the platform, the scanner definitions and the ray-scene interactions with the exception of transmissive voxels are deterministic. To allow for more realistic point clouds, random noise sources may be introduced at various points of the simulation. A single distance measurement is attributed with a random ranging error, drawn from a normal distribution with parameters defined for each scanner. Additionally, random platform noise can be added to simulate trajectory estimation or scan position localisation errors. By default, the system time is used as random seed, which results in different outcomes for every simulation run. However, the ability to produce identical results in repeated simulation runs is a major advantage of VLS over real data acquisitions and may be aspired for certain use cases. To allow for repeatable survey results while using randomness, there are additional options to define a custom seed for pseudo-randomness generation. Still, when using multithreading, each thread accesses the randomness generator in a non-deterministic order, resulting in different outputs for every simulation run. This can be avoided by running HELIOS++ in single-threaded mode and with a fixed seed, which will generate the exact same result in repeated runs, even for transmissive voxels, as the random number R drawn from the uniform distribution (cf. Eq. (6)) is also created based on the seed.

4. Scientific use cases and fitness for purpose of general-purpose VLS

In order to determine scientific value and use cases of a general-purpose VLS method, we subsequently present different applications of laser scanning simulation. These publications define use cases where VLS, notably the HELIOS VLS concept, has been successfully utilised for scientific research. We include studies using HELIOS (Bechtold and Höfle, 2016), because HELIOS and HELIOS++ share the same core concepts of modularity, subray sampling, and configurability.

The analysis is conducted on all publications that cite the original publication of Bechtold and Höfle (2016), and which actively use HELIOS in their research according to the published article. The following questions are posed to analyse the contents of each publication:

- What is the scientific objective of the simulation?
- How many simulations are carried out on which platform (ALS, TLS, ULS, MLS)?
- Which type of model is employed to compose the scene for the simulation?

[7] https://laszip.org/, Isenburg (2013)

[8] As part of the HELIOS++ distribution, we provide a script to perform this conversion: https://github.com/3dgeo-heidelberg/helios/blob/main/pyhelios_demo/txt2las_wdp.py

- Which parameters are extracted from the point cloud, if applicable?
- What is the resulting VLS point cloud compared to?

The synthesis of the literature review allows grouping the publications and the purposes of laser scanning simulation into four main categories: (1) optimising or analysing different scan settings and acquisition modes, (2) comparing parameters extracted from simulated data to parameter values obtained from the 3D models, (3) generating training data for supervised machine learning, and (4) testing and developing novel or future algorithms and sensors.

4.1. Data acquisition planning and scan setting effects

Data acquisition planning at its core is an optimisation problem. The goal is to acquire data that is fit for the purpose of the analysis with minimal effort. This can be a minimum number of scan positions, flight lines, etc., that is sufficient for the specific requirement to the data, such as coverage of the scene regarding occlusion or point density. Flight planning tools and visibility analysis-based methods can be used to create potential acquisition plans. However, data coverage only has limited significance for the usability of the data. For example, even if point density and occlusions are considered, a sensor might not be sufficiently accurate to capture a phenomenon of interest. Additionally, long ranges may lead to large footprints, resulting in a reduced resolution in the point cloud.

With virtual laser scanning, assuming that at least a coarse model of the area of interest is available, a 3D point cloud can be generated and tested for its usability directly in the planned application. There is no need to define proxy metrics like target point density, required accuracy and overlap as required by simple survey planning tools. This way, users can ensure that the acquired data will meet all requirements such as coverage, adequate representation of geometry, and resolution before actually going to the field to collect the data, by running their analyses on the simulated point clouds and interpreting the results. Similarly, the effects of different scan settings on parameters extracted from the simulated point clouds can be studied with HELIOS++, as single variables can easily be manipulated in a way that is isolated from all other influences, including environmental influences, which are very difficult to control in repeated real-world acquisitions. For example, the effects of flying height and maximum scan angle on the resulting ground point density and resolution (i.e., illuminated area per beam) may be analysed. Similarly, TLS scans of different resolution can be simulated and the effect of resolution on the results of data analysis (e.g., the quality of extraction of tree stems) can be quantified.

While the quality of the simulation in terms of being physically realistic highly depends on the input models, even a coarse model can be useful to estimate occlusion and resulting point densities. Such analyses, carried out prior to real data acquisitions, can save valuable time in the field. Using a Monte-Carlo approach, multiple simulations using different parameters are carried out to create the optimal acquisition plan (in terms of number of positions, time, etc.).

Existing publications in this category include Backes et al. (2020), who use a digital surface model created from photogrammetry to simulate acquisition of an alpine valley by TLS and ULS. They estimate the minimum detectable change to optimise scan positions and trajectories for change analysis. Similar analyses, but with focus on resulting point density and completeness of data acquisition, are carried out based on a DEM provided by public agencies by Lin and Wang (2019). They are able to show that when scaling the scene model, the pulse frequency needs to be adapted in order to keep the point density constant (in scaled units). This validates the simulation's representation of spatial scales.

A validation of scan position planning based on viewshed analysis is carried out with respect to achieved accuracy, point density and completeness by Previtali et al. (2019). Their use cases are acquisitions of complex archaeological sites, where complete coverage is often difficult to achieve due to occlusions. In two examples, they optimise the positions of 97 and 16 scan positions, respectively, to scan a basilica and part of an ancient food storage.

The effect of scan settings on point cloud-based parameters is analysed for forestry settings by Hämmerle et al. (2017), who extract understory tree heights from TLS and ULS data. They use 3D models created with the Arbaro tree generator[9], on which they densely sample points to create a reference point cloud. They find a favourable trade-off between acquisition effort and accuracy of results for a number of three TLS scan positions around a tree object of interest. Similarly, Li et al. (2020) create 3D tree models using OnyxTREE[10] and evaluate the influence of scan parameters on extracted values of diameter at breast height (DBH), tree height, stem curve, and crown volume. They succeed in reducing the root-mean-squared error on these values by iteratively adapting scan parameters.

4.2. Algorithm and method evaluation: validation and calibration

In numerous point cloud analysis methods, the objective is to extract certain parameters describing the objects of interest. For example, in forestry applications, ALS and TLS point clouds are commonly used to derive the diameter at breast height of trees, crown radii, tree heights, and tree species (Giannetti et al., 2018). To calibrate the extraction algorithms as well as to validate the results, in situ measurements are required, which are laborious and costly to acquire, and not free of error.

An alternative to this in situ data acquisition can be provided by HELIOS++, if the parameters to be extracted from the point cloud can be derived from the given 3D objects within the scene. Scenes can even be created by randomly selecting scene objects from a collection. For example, a method may be designed to measure DBH from a point cloud. In VLS, the true values of DBH can be derived from the stem models, or the virtual tree models themselves may be generated according to given values (or distributions) of DBH. In addition to being error-free, the domain of the parameters (e.g., the range of DBH values) used in the simulation can be defined by the user. A simulated example can therefore be picked to have exactly the properties needed in a specific application (say, use case-specific DBH values between 20 cm and 25 cm), whereas real objects with the required properties may be hard to find.

In addition to parameter extraction, parameter values derived from the 3D input models can be used to evaluate the performance of classification algorithms. Currently, classification methods are difficult to compare, as authors use different evaluation approaches, mainly due to the lack of semantic reference data. Since in VLS the interaction between the laser beam and the scene can be attributed to the exact mesh face that is hit during ray casting, and therefore to a specific object, the reference data is free of attribution error even in highly complex and multi-echo scenarios. This has two compelling advantages: First, reference data can be created automatically, drastically cutting costs and providing the possibility of generating massive amounts. And second, it removes labelling errors, as manual labelling will always include some degree of human error. On the other hand, when using VLS data, it has to be ensured that the employed models (scene objects, but also fullwave simulation, resolution, etc.) are not limiting in the evaluation. The degree of detail and accuracy of the input scene model directly influences the VLS output and therefore must be balanced with respect to study objectives.

Considerable research using the HELIOS VLS concept has been undertaken in multiple forestry applications to validate or calibrate extracted parameters. Liu et al. (2019a) and Liu et al. (2019b) estimate leaf angles on trees, where reference data is practically impossible to obtain, because in real settings, leaves are continuously moving due to wind. VLS allows validating their approach of leaf angle estimation and

[9] http://arbaro.sourceforge.net/ (Weber and Penn, 1995)
[10] http://www.onyxtree.com/

subsequently applying it to real data. Wang et al. (2020) validate their calculation of photon recollision probability over spatial locations using VLS data. Zhu et al. (2020) use two different methods to assess Leaf Area Index (LAI) and compare these methods with a reference obtained from the object models, allowing them a comparison with the true LAI of the input objects. For tree segmentation, reference data is also difficult to obtain, as tree crowns often intersect one another. By using HELIOS, Wang (2020) and Xiao et al. (2019) model the attribution of singular points to individual trees for training and validation of their segmentation methods. Wang (2020) achieves 2.9% and 19.8% root-mean-square error (RMSE) for tree height and crown diameter estimates, respectively.

In a non-forestry application, road curves are reconstructed and the reconstruction is compared with the model parameters (Zhang et al., 2019), achieving a relative error of 0.6% in circle radii estimation using VLS data. Requirements for Building Information Modelling (BIM) are evaluated by Rebolj et al. (2017), who generate around 100 point clouds to ascertain the influence of parameter values. They define accuracy criteria for the successful identification of building elements in the scans. Bechtold et al. (2016) test a segmentation tool for rock outcrops by using HELIOS as a simulator. They show the value of simulated test data for method development by easily generating point clouds with different occlusions and scan settings, resulting in a multitude of point densities and point patterns.

Weiser et al. (2021) analyse different opaque voxel models at different scales and their effect on common tree metrics, showing that a scaled voxel model requires much less complexity (i.e., allows for a larger voxel size) than binary voxels at a fixed scale. Schäfer et al. (2019) use the voxel-support of HELIOS++ for similar analyses with ALS and ULS data. Their findings show that the required model complexity very much depends on the type of LiDAR that is simulated (e.g., ALS, TLS, or ULS). These results suggest that the model choice should be validated, ideally using real data of similar properties, to ensure that the effects of modelling are secondary to the ones being investigated.

4.3. Method training

With the advent of neural networks as supervised machine learning method in geospatial domains, the need for training data has grown almost indefinitely. Whereas raster-based approaches can make use of existing pre-trained networks by domain transfer (Pires de Lima and Marfurt, 2019), no such networks exist for point-based deep learning such as PointNet or PointNet++ (Qi et al., 2017).

Simulated data, though only replicating parts of reality, can be used by neural networks to learn basic descriptors which describe point cloud neighbourhoods, e.g., planes, corners, or edges. From these descriptors, higher-level features are derived, which are subsequently used in classification or regression (Winiwarter et al., 2019).

Once a network has learned to represent data in form of these features, it can be adjusted to real data by adding relatively small amounts of training data, in approaches shown to work for the image domain (Danielczuk et al., 2019). Further research is needed on this approach especially for point cloud data, but HELIOS++ allows easy and fast generation of training data, which does not suffer from labelling errors.

As an example application of this purpose, Martínez Sánchez et al. (2019) use HELIOS-simulated data to train and evaluate a semantic classification of an urban scene created from OpenStreetMap[11] models. The use of VLS allows a quantification of their classifier's total error, which amounts to 0.5% on simulated point clouds.

4.4. Sensing experimentation

The fourth category summarises publications concerned with the development of novel sensors and methods, such as Park et al. (2020), who present a new Time-of-Flight sensor and compare its results to a HELIOS simulation. In general, the parameters of the virtual sensors can be tuned to resemble a non-existent sensor, the performance of which can then be simulated without the need of actually building a prototype. Especially when looking for potential improvements of current sensors, this enables the identification of weak links or bottlenecks for certain use cases.

More in-depth experimentation is also conceivable, where, e.g., a novel deflector model shall be simulated. Due to the open-source license, a developer may take the HELIOS++ framework and would only have to implement a new deflection method, while the scene and all other components can be used as is to run a simulation, testing the usability of the novel deflector model. An example is the Risley prism, commonly used in DJI's Livox sensors, which has already been included in HELIOS++ (cf. Fig. 5e).

Especially when considering the short lifecycle of current hardware, simulation may be the only way to ensure fitness for use of the sensor. Similar concepts are used in radar and optical remote sensing, as especially tools working with data acquired by satellites can be developed using the simulated data, and are then ready to use as soon as the first real data is delivered. The most prominent example is DART, which has recently been used for the simulation of Trishna satellite data, to be launched in 2025 (Gastellu-Etchegorry et al., 2020).

5. Performance comparison of HELIOS++, HELIOS and DART

As the direct successor of HELIOS (Bechtold and Höfle, 2016), it is interesting to compare the performance of HELIOS++ with the original implementation in Java. HELIOS++ was not just a translation or port of the code, but also comes with multiple computational improvements over its predecessor. One of these improvements concerns the ability for the user to better leverage accuracy at the cost of runtime, or vice versa, by setting parameters accordingly. For some tasks, a rough, thereby faster simulation may suffice, whereas for other tasks very detailed simulation is required, but smaller sample sizes can be used or long processing times are acceptable.

Another improvement concerns the binning mechanism for the full-waveform simulation, which is used for maxima detection even if the waveform is not written to an output file. In HELIOS++, we use the parameters `binSize_ns` and `maxFullwaveRange_ns` for this binning, where the bin size is used for both the outgoing pulse and the returned waveform. To limit the impact on performance of very low-incidence rays with a high sampling quality, the user can provide a maximum length of the recorded waveform, beyond which any further echoes are discarded. We present the improved computational performance in Section 5.1.

In Section 5.2, a second comparison is carried out between HELIOS++ and DART, the state-of-the-art Monte Carlo ray tracing model (Gastellu-Etchegorry et al., 2016). Here, we investigate the resulting waveform for a single beam with a large footprint. As a demo scene, we use a model of a citrus orchard, which is part of the RAMI V radiative transfer model intercomparison run by the European Commission[12] (Pinty et al., 2001; Widlowski et al., 2015).

5.1. HELIOS++ and HELIOS - runtime and memory consumption

To compare the performances of HELIOS++ (Version 1.0.9) and HELIOS (Version 2018-09-24), we carry out a number of simulations using different parameter settings, and record the runtime as well as peak memory usage. We present three different scenes at different complexity levels, and make use of the option in HELIOS++ to write different file types and to skip the echo width determination, if not

[11] https://www.openstreetmap.org/

[12] See https://rami-benchmark.jrc.ec.europa.eu/_www/

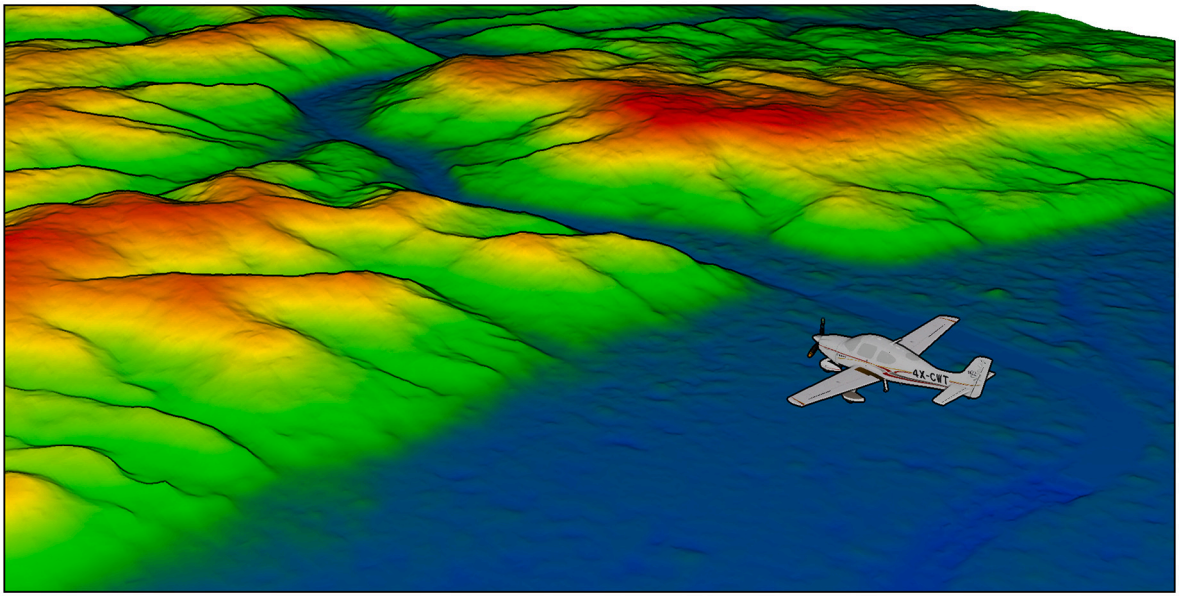

Fig. 8. Example survey for airborne laser scanning (ALS), using a digital terrain model as object and a standard ALS instrument as scanner. The 3″-SRTM model by the USGS is used as input raster.

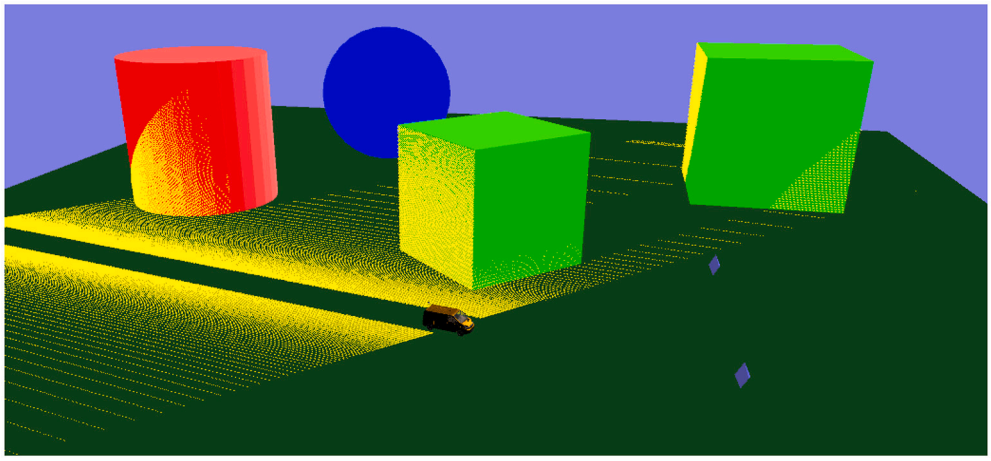

Fig. 9. Example survey for mobile laser scanning (MLS), with a car driving between objects of simple geometry. The yellow points show the acquired point cloud of scanned parts. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

required.

The first example scenario represents an ALS survey over terrain, which is created by loading a digital terrain model in GeoTIFF format as described in Section 3.6. Since the GeoTIFF loader in the Java version was faulty and this version is no longer maintained, we created a mesh in Wavefront Object format as input for the Java version. We simulate two flight lines at an altitude of 1500 m asl over terrain with an extent of 26 km × 17.8 km. As scanner we use a *Leica ALS50-II*, and a Cessna SR-22 as platform. The scene is depicted in Fig. 8.

The second scenario represents a mobile laser scan of an urban area, where a car is driving through downtown buildings modelled as prisms, cylinders and pyramids (Fig. 9). The trajectory of the car is 208 m long and its average speed is 20 m s^{-1}. The scanner is an oblique-mounted *RIEGL VUX-1UAV*.

In the third and final scenario, we present a TLS survey of a vegetation scene with a large potential for multi-echoes, as it represents two trees generated with the Arbaro Tree Simulator (Weber and Penn, 1995), scanned using a *RIEGL VZ-400* from two static positions. This simulation has a larger number of geometric primitives. A visual is depicted in Fig. 10.

Each simulation scenario is carried out three times on an Intel-i9-7900 @ 3.3 GHz with 64 GB of RAM, and I/O on an SSD connected via SATA. The average runtime and memory consumption values are given in Table 3.

In the case of largest improvement over HELIOS, runtimes of HELIOS++ are lower by 99% (MLS) while using only 2% of the previously required memory. Especially for large scenes, the issue of memory footprint has been a limiting factor for the usability of HELIOS. In the case of the ALS scene, the memory consumption can be reduced by 72%, from more than 6 GB to just under 2 GB, with a runtime reduction of 78%. For the TLS scene using the complex tree shapes, the reduced processing overhead leads to a reduction of 94% in memory usage and 91% in runtime. From these results we deduce a significant improvement both in runtime and memory footprint when comparing any configuration of HELIOS++ to the previous Java version.

Since not only the output options and scene complexity influence the runtime and memory consumption, we provide a list of parameters and their effects on simulation speed in Table 4.

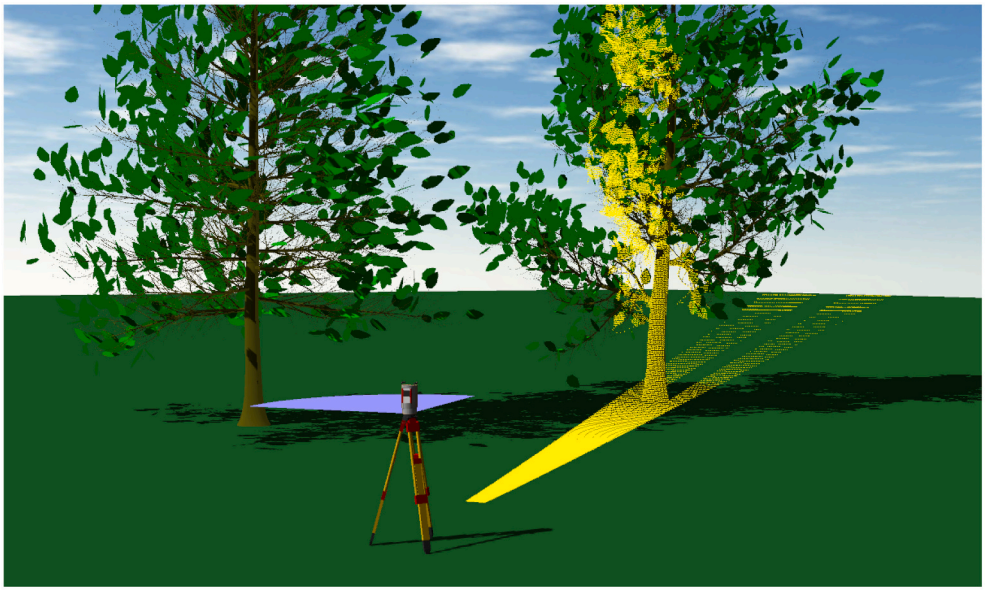

Fig. 10. Example survey for terrestrial laser scanning (TLS), scanning two complex tree models created using the Arbaro Tree Simulator. The yellow points show the acquired point cloud of scanned parts. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Performance comparison of HELIOS and HELIOS++ with different options. We use default parameters for waveform modelling (`beamSampleQuality = 3` and `binSize_ns = 0.25`; `numBins = 100` and `numFullwaveBins = 200` for HELIOS++ and HELIOS, respectively). Runtimes are averaged over three runs (± standard deviation), memory footprint is the highest value (maximum) during the full run.

	HELIOS (Java) Version 2018-09-24	HELIOS++ Version 1.0.9[a]		
Echo width	✓	✓		
Waveform output	✓	✓	✓	
File format	XYZ	XYZ	XYZ	LAS
Scene 1 (ALS, GeoTIFF)	4,712,1± 613.8 s 6,570 MB	2,819,5± 43.0 s 1,838 MB	2,721,4± 0.5 s 1,846 MB	1,048.4± 10.4 s 1,827 MB
Scene 2 (MLS, geometric primitives)	68.0± 2.3 s 560 MB	0.5± 0.0 s 11 MB	0.5± 0.0 s 11 MB	0.7± 0.3 s 12.3 MB
Scene 3 (TLS, tree models)	362.6± 6.7 s 5,360 MB	55.5± 0.0 s 320 MB	45.2± 0.0 s 320 MB	34.4± 0.3 s 320 MB

[a] https://github.com/3dgeo-heidelberg/helios/releases/tag/v1.0.9.

5.2. HELIOS++ and DART – waveform comparison

In this comparison, we simulate a single LiDAR beam with a footprint diameter of 50 m over a model of the *Wellington Citrus Orchard*, consisting of 1115 trees created from 10 different models. We report the `lidar_co_sgl` metric, which in our case is identical to the `lidar_tot` metric, as multiple scattering of single photons is not supported in HELIOS++. For DART (Version 5.7.5 / 2019-08-23 v1140), both metrics in the form of waveforms are reported.

A view on the scene in the Python-based HELIOS++ viewer is shown in Fig. 11. The DART 3D viewer was unable to visualise the full scene due to memory issues (>256 GB RAM).

We evaluate the resulting waveform qualitatively by plotting selected simulations (with different beam sample quality settings, i.e., number of subrays) and quantitatively by RMSE compared to DART. In Fig. 12a, the waveforms of HELIOS++ with beam sample qualities of 1, 2, 5, and 50 are shown, corresponding to 1, 7, 62, and 7673 subrays, respectively. The waveform with beam sample quality 1 indicates that the central ray hits only the ground at 2000 m range. With more subrays, a signal from the canopy is recorded. Note that the DART signal shows a ground signal slightly below 2000 m. 2000 m is the distance between the sensor and the planar terrain in the scene, as defined by the RAMI operators[13]. The HELIOS++ ground returns show at 2000.20 m, which is due to the non-central rays travelling a longer path.

The weighted mean of these slanted paths can be evaluated using a two-dimensional Gaussian with standard deviation $\sigma = 50$ m at a flight height of 2000 m (Eq. (8)). A numerical evaluation results in values around 2000.21 m, which corresponds to the result given by

Table 4
Influences of different parameters on simulation runtime in HELIOS++. For more details, the reader is pointed to the official documentation: https://github.com/3dgeo-heidelberg/helios/wiki.

Parameter	Location	Simulation duration increasing with	Severity
Scene geometry	Scene	↑ complexity	> linear
Number of primitives	Scene	↑ number	> linear
Number of legs	Survey	↑ number	linear
Length of path	Survey	↑ length	linear
Platform speed	Survey	↓ speed	linear
Max. number of returns	Scanner	↑ number	< linear
Max. fullwave range (ns)	Scanner	↑ range	linear
Number of subrays (beamSampleQuality)	Survey, Scanner	↑ quality	> linear
Pulse frequency	Survey, Scanner	↑ frequency	linear
(Max) scan angle(s)	Survey, Scanner	↑ angle range	linear
Fullwave bin size	Survey, Scanner	↑ bin size	linear
Scan frequency	Survey, Scanner	(no influence)	
Trajectory interval	Survey	(no influence)	
Fullwave window size	Survey, Scanner	(no influence)	
Pulse length	Scanner	(no influence)	

[13] See https://rami-benchmark.jrc.ec.europa.eu/_www/measurements/phase_meas.php?strPhase=meas&strVar=lidar_co_sgl, last accessed 2021-08-18.

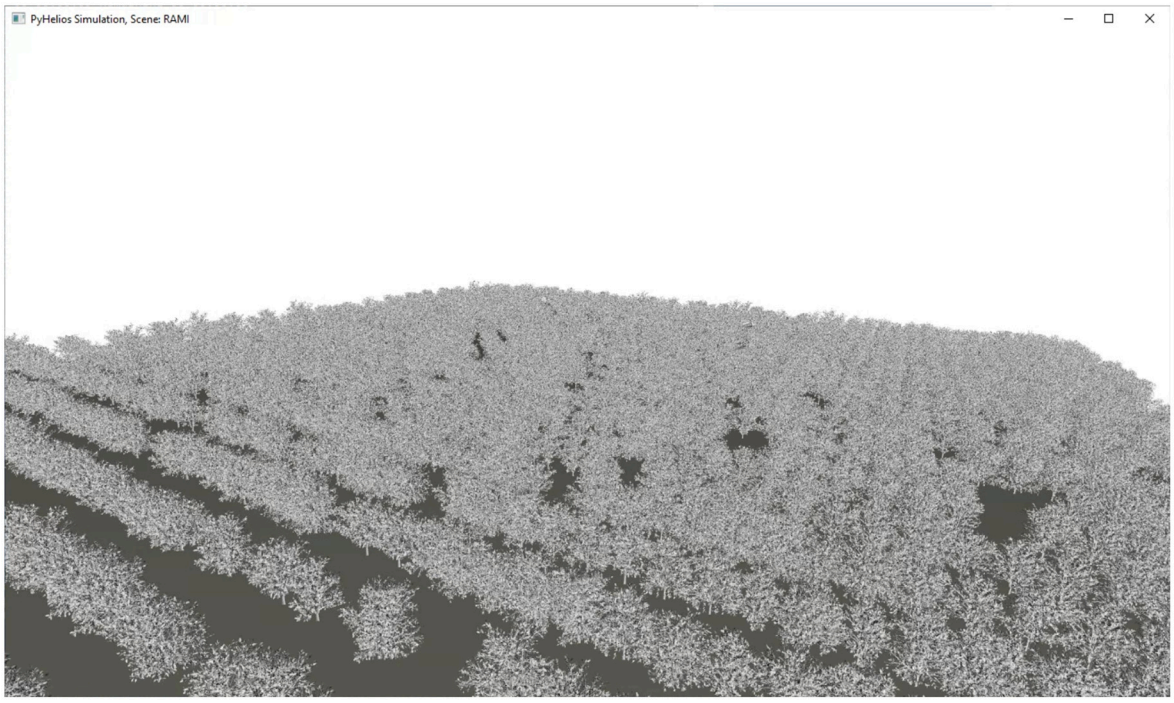

Fig. 11. 3D view of the Wellington Citrus Orchard (RAMI V scene). The view was created with pyhelios employing the *Open3D* library (Zhou et al., 2018).

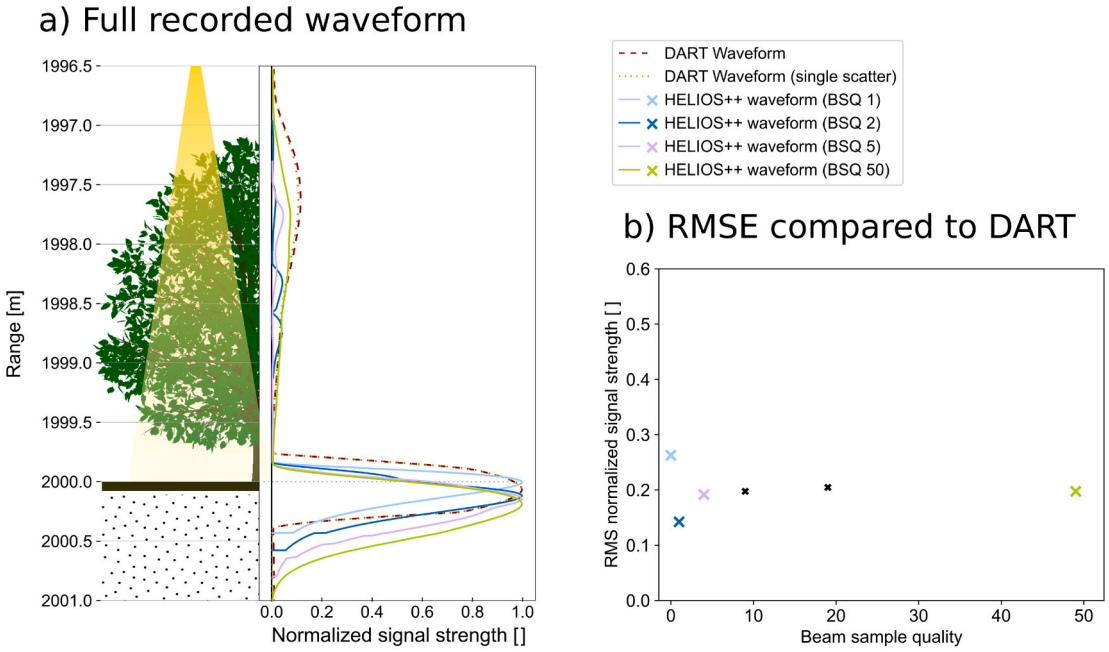

Fig. 12. Comparison of waveforms between HELIOS++ and DART, and dependence on beam sample quality (BSQ). a) Waveforms and schematic tree model. Note that the simulation includes 10 different tree models, and the waveforms (except for the one with BSQ 1) are the response from a circular area with 50 m diameter. The waveforms are all normalised to [0,1] before plotting. b) Root-Mean-Square errors of the waveform bins for different discretisation levels. BSQs of 1, 2, 5, 10, 20, and 50 correspond to 1, 7, 62, 279, 1185, and 7673 subrays cast into the scene.

HELIOS++. HELIOS++ does not support simulation of parallel subrays.

$$\overline{d} = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{x=-50}^{50} \int_{y=-50}^{50} \sqrt{2000^2 + x^2 + y^2}\ exp\left(-\frac{x^2 + y^2}{2\sigma^2} \right) \mathrm{dy}\ \mathrm{dx}$$
$$\approx 2000.213m \tag{8}$$

The quantitative comparison shows that an increasing number of subrays generally results in less discrepancy between DART and HELIOS++ (Fig. 12b). However, it suggests that a certain error remains even with a very large number of subrays. Also, certain numbers of subrays (here: beam sample quality 2, dark blue) may be favourable due to sampling effects, which disappear e.g., upon rotation of the scene. This is partly due to the fact of non-parallel rays (as explained above), and in part due to non-multiple scattering of radiation in HELIOS++. The latter effect, however, is very small for the sample scene, as can be seen by comparing the dotted line and the dashed line in Fig. 12. The

RMSE value between the normalised signal strengths (per bin) when comparing the two normalised DART approaches is 0.005 (dimensionless units). Furthermore, the (outgoing) pulse waveform generated by DART is symmetric with respect to the maximum, whereas the waveform used in HELIOS++ is not, leading to non-zero RMSE values.

HELIOS++ is aimed at simulating full laser scanning surveys, with less focus on singular beams than, e.g., DART. This also shows in the simulation times for a single shot, as in the example of the RAMI scene. Here, running HELIOS++ on an AMD Ryzen Threadripper (32 cores, 3.7 GHz) takes just under 1 h with a peak memory usage of 157.6 GB at a beam sample quality of 1. The DART simulation of the same scene takes 4.4 min at 27.3 GB peak memory usage (using default parameters, 100,000 photons and 1 m^3 voxel size). Subsequent runs of DART for each additional beam take approximately 1.57 s (0.64 beams/s), whereas HELIOS++ is able to trace around 200,000 beams/s after 3,300 s (55 min) of loading the scene. When running HELIOS++ with a beam sample quality of 317 (314,541 subrays per shot), DART and HELIOS++ would perform at approximately the same speed (when measured in pulses per second and ignoring the overhead of scene loading).

In addition to waveform validation, the experiment showed HELIOS++'s ability to work with large amounts of data. The over 1,000 highly detailed tree models in the scene make up more than 100 million primitives. However, especially with dynamic objects such as trees, we argue that such a level of detail is not required for simulation, as in reality, effects such as wind predominate any efforts of more detailed modelling.

6. Conclusions

With HELIOS++, we present an open-source laser scanning simulation framework that enables highly performant virtual laser scanning (VLS). In its C++ implementation and with the possibility to use the `pyhelios` package in Python to manipulate simulation parameters, we opt for an efficient and easy-to-use software. While physical accuracy and realism may be superseded by complementary simulation software, HELIOS++ provides a flexible solution to balance computational requirements (runtime, memory footprint) and quality of results (physical realism, accuracy), while being straightforward to configure for users and to apply for specific scientific purposes.

Different studies using the HELIOS VLS concept demonstrate the fitness-for-use of virtually acquired laser scanning data for analyses in different categories of purpose. The main categories are (a) planning of flight patterns (ULS, ALS) or scan positions (TLS), (b) generation of ground-truth data for validation of algorithms that extract parameters from point clouds (i.a., for forestry), (c) generation of training data for supervised machine learning, and (d) testing and developing novel or future sensors and algorithms.

The novel object model of HELIOS++, the transmissive voxel model, allows a stochastic simulation of penetrable objects, such as vegetation canopies, without the need for a highly detailed 3D mesh model. Generally, HELIOS++ performs much faster than its predecessor HELIOS on common scenes, and allows the user to omit intensive computations if not required (e.g., the calculation of the echo width for returned pulses). Although the exact waveforms obtained with state-of-the-art ray-tracing methods such as DART differ slightly from the HELIOS++ waveform, we show the validity of the multi-beam subray approach for simulating complex and large scenes ($>$ 100 million primitives).

To summarise, HELIOS++ is a versatile and extensively documented scientific software for laser scanning simulations, which provides a tool to generate VLS data, complementing data obtained by real-world laser scanning. The framework invites users to experiment and develop new ideas, while enabling to rely on established algorithms from the literature.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in the frame of the project SYSSIFOSS (project number: 411263134/2019-2022); and the Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Education and Research) in the frame of the project LOKI (funding code: 03G0890A).

The authors wish to acknowledge the contribution of Patrick Herbers (Ruhr-Universität Bochum) for the improvement the triangle-ray intersection in C++, which helped to significantly reduce the runtime of HELIOS++.

Figs. 1, 4, 5, and 8 show an airplane model CC-BY Emmanuel Beranger. Figs. 1 and 4 show a house model by free3d.com user `gerhald3d`, and a drone model by cgtrader.com user `CGaxr`. Figs. 11 and 12 show tree models created by Jan Stuckens, Ben Somers and colleagues (from the Katholieke Universiteit Leuven in Belgium)[14].

References

Agrawal, N., Agrawal, N.K., Lohani, B., 2004. Development of a simulator for airborne altimetric LiDAR. In: Proceedings of Map India. http://home.iitk.ac.in/blohani/Limulator/publication/mapIndia.pdf.

ASPRS, 2011. LAS Specification. https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf.

Backes, D., Smigaj, M., Schimka, M., Zahs, V., Grznárová, A., Scaioni, M., 2020. River morphology monitoring of a small-scale alpine riverbed using drone photogrammetry and LiDAR. In: ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLIII-B2-2020, pp. 1017–1024. https://doi.org/10.5194/isprs-archives-XLIII-B2-2020-1017-2020.

Bechtold, S., Höfle, B., 2016. HELIOS: A multi-purpose LiDAR simulation framework for research, planning and training of laser scanning operations with airborne, ground-based mobile and stationary platforms. ISPRS Annal. Photogramm. Remote Sens. Spatial Inf. Sci. III-3, 161–168. https://doi.org/10.5194/isprs-annals-III-3-161-2016.

Bechtold, S., Hämmerle, M., Höfle, B., 2016. Simulated full-waveform laser scanning of outcrops for development of point cloud analysis algorithms and survey planning: an application for the HELIOS LiDAR simulation framework. In: Proceedings of the 2nd Virtual Geoscience Conference, Bergen, Norway, pp. 57–58. http://lvisa.geog.uni-heidelberg.de/papers/2016/Bechtold_et_al_2016.pdf.

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. CACM 18, 509–517. https://doi.org/10.1145/361002.361007.

Calders, K., Lewis, P., Disney, M., Verbesselt, J., Herold, M., 2013. Investigating assumptions of crown archetypes for modelling LiDAR returns. Remote Sens. Environ. 134, 39–49. https://doi.org/10.1016/j.rse.2013.02.018.

Carlsson, T., Steinvall, O., Letalick, D., 2001. Signature Simulation and Signal Analysis for 3-D Laser Radar.

Danielczuk, M., Matl, M., Gupta, S., Li, A., Lee, A., Mahler, J., Goldberg, K., 2019. Segmenting unknown 3D objects from real depth images using mask R-CNN trained on synthetic data. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 7283–7290. https://doi.org/10.1109/ICRA.2019.8793744.

Dayal, S., Goel, S., Lohani, B., Mittal, N., Mishra, R.K., 2021. Comprehensive airborne laser scanning (ALS) simulation. J. Indian Soc. Remote Sens. 49, 1603–1622. https://doi.org/10.1007/s12524-021-01334-5.

Disney, M., Lewis, P., North, P., 2000. Monte carlo ray tracing in optical canopy reflectance modelling. Remote Sen. Rev. 18, 163–196. https://doi.org/10.1080/02757250009532389.

Disney, M.I., Kalogirou, V., Lewis, P., Prieto-Blanco, A., Hancock, S., Pfeifer, M., 2010. Simulating the impact of discrete-return LiDAR system and survey characteristics over young conifer and broadleaf forests. Remote Sens. Environ. 114, 1546–1560. https://doi.org/10.1016/j.rse.2010.02.009.

Disney, M.I., Lewis, P.E., Bouvet, M., Prieto-Blanco, A., Hancock, S., 2009. Quantifying surface Reflectivity for Spaceborne LiDAR via Two Independent Methods. IEEE Tran. Geosci. Remote Sens. 47, 3262–3271. https://doi.org/10.1109/TGRS.2009.2019268.

Duma, V.F., Schitea, A., 2018. Laser scanners with rotational risley prisms: exact scan patterns. Proc. Rom. Acad. Ser. A 19 (1), 53–60.

[14] https://rami-benchmark.jrc.ec.europa.eu/_www/phase/phase_exp.php?strTag=level3&strNext=filter_testcases&strPhase=RAMI5&strTagValue=ACT_HET14_WCO_UND, last accessed 2021-08-18

Gastellu-Etchegorry, J.P., Wang, Y., Regaieg, O., Yin, T., Malenovsky, Z., Zhen, Z., Yang, X., Tao, Z., Landier, L., Al Bitar, A., et al., 2020. Why to model remote sensing measurements in 3D? recent advances in dart: atmosphere, topography, large landscape, chlorophyll fluorescence and satellite image inversion. In: 2020 5th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). IEEE, pp. 1–6. https://doi.org/10.1109/ATSIP49331.2020.9231884.

Gastellu-Etchegorry, J.P., Yin, T., Lauret, N., Cajgfinger, T., Gregoire, T., Grau, E., Feret, J.B., Lopes, M., Guilleux, J., Dedieu, G., Malenovský, Z., Cook, B., Morton, D., Rubio, J., Durrieu, S., Cazanave, G., Martin, E., Ristorcelli, T., 2015. Discrete anisotropic radiative transfer (DART 5) for modeling airborne and satellite spectroradiometer and LIDAR acquisitions of natural and urban landscapes. Remote Sens. 7, 1667–1701. https://doi.org/10.3390/rs70201667.

Gastellu-Etchegorry, J.P., Yin, T., Lauret, N., Grau, E., Rubio, J., Cook, B.D., Morton, D. C., Sun, G., 2016. Simulation of satellite, airborne and terrestrial LiDAR with DART (I): waveform simulation with quasi-monte carlo ray tracing. Remote Sens. Environ. 184, 418–435. https://doi.org/10.1016/j.rse.2016.07.010.

Giannetti, F., Puletti, N., Quatrini, V., Travaglini, D., Bottalico, F., Corona, P., Chirici, G., 2018. Integrating terrestrial and airborne laser scanning for the assessment of single-tree attributes in mediterranean forest stands. Europ. J. Remote Sens. 51, 795–807. https://doi.org/10.1080/22797254.2018.1482733.

Golub, G., Kahan, W., 1965. Calculating the singular values and pseudo-inverse of a matrix. J. Soc. Indus. Appl. Math. Series B Numer. Anal. 2, 205–224. https://doi.org/10.1137/0702016.

Goodwin, N.R., Coops, N.C., Culvenor, D.S., 2007. Development of a simulation model to predict LiDAR interception in forested environments. Remote Sens. Environ. 111, 481–492. https://doi.org/10.1016/j.rse.2007.04.001.

Hämmerle, M., Lukac, N., Chen, K.C., Koma, Z., Wang, C.K., Anders, K., Höfle, B., 2017. Simulating various terrestrial and UAV LiDAR scanning configurations for understory forest structure modelling. ISPRS annals of photogrammetry. Remote Sens. Spatial Inf. Sci. IV-2/W4, 59–65. https://doi.org/10.5194/isprs-annals-IV-2-W4-59-2017.

Hodge, R.A., 2010. Using simulated terrestrial laser scanning to analyse errors in high-resolution scan data of irregular surfaces. ISPRS J. Photogram. Rem. Sens. 65, 227–240. https://doi.org/10.1016/j.isprsjprs.2010.01.001.

Holmgren, J., Nilsson, M., Olsson, H., 2003. Simulating the effects of LiDAR scanning angle for estimation of mean tree height and canopy closure. Can. J. Remote Sens. 29, 623–632. https://doi.org/10.5589/m03-030.

Isenburg, M., 2013. LASzip: lossless compression of LiDAR data. Photogramm. Engg. Remote Sens. 79, 209–217. https://www.cs.unc.edu/isenburg/lastools/download/laszip.pdf.

Jutzi, B., Gross, H., 2009. Normalization of lidar intensity data based on range and surface incidence angle. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci 38, 213–218.

Kim, S., Lee, I., Lee, M., 2012. LiDAR waveform Simulation over complex targets. ISPRS - international archives of the photogrammetry. Remote Sens. Spatial Inf. Sci. XXXIX-B7, 517–522. https://doi.org/10.5194/isprsarchives-XXXIX-B7-517-2012.

Kim, S., Min, S., Kim, G., Lee, I., Jun, C., 2009. Data simulation of an airborne lidar system. In: Turner, M.D., Kamerman, G.W. (Eds.), Laser Radar Technology and Applications XIV, 73230C. SPIE, Orlando, Florida, United States.

Kukko, A., Hyyppä, J., 2009. Small-footprint laser scanning simulator for system validation, error assessment, and algorithm development. Photogramm. Engg. Remote Sens. 75, 1177–1189. https://doi.org/10.14358/PERS.75.10.1177.

Lewis, P., 1999. Three-dimensional plant modelling for remote sensing simulation studies using the botanical plant modelling system. Agronomie 19, 185–210. https://doi.org/10.1051/agro:19990302.

Lewis, P., Muller, J.P., 1993. The advanced radiometric ray tracer: ararat for plant canopy reflectance simulation. Proc. 29th Conf. Int. Soc. Photogramm. Remote Sens. 29. In: http://www.isprs.org/proceedings/XXIX/congress/part7/26_XXIX-part7.pdf.

Li, L., Mu, X., Soma, M., Wan, P., Qi, J., Hu, R., Zhang, W., Tong, Y., Yan, G., 2020. An iterative-mode scan design of terrestrial laser scanning in forests for minimizing occlusion effects. IEEE Tran. Geosci. Remote Sens. 1–20. https://doi.org/10.1109/TGRS.2020.3018643.

Pires de Lima, R., Marfurt, K., 2019. Convolutional neural network for remote-sensing scene classification: transfer learning analysis. Remote Sens. 12, 86. https://doi.org/10.3390/rs12010086.

Lin, C.H., Wang, C.K., 2019. Point density simulation for ALS survey. In: Proceedings of the 11th International Conference on Mobile Mapping Technology (MMT2019), pp. 157–160. https://www.geog.uni-heidelberg.de/md/chemgeo/geog/gis/mmt2019-lin_and_wang_compr.pdf.

Liu, J., Skidmore, A.K., Wang, T., Zhu, X., Premier, J., Heurich, M., Beudert, B., Jones, S., 2019a. Variation of leaf angle distribution quantified by terrestrial LiDAR in natural European beech forest. ISPRS J. Photogram. Rem. Sens. 148, 208–220. https://doi.org/10.1016/j.isprsjprs.2019.01.005.

Liu, J., Wang, T., Skidmore, A.K., Jones, S., Heurich, M., Beudert, B., Premier, J., 2019b. Comparison of terrestrial LiDAR and digital hemispherical photography for estimating leaf angle distribution in European broadleaf beech forests. ISPRS J. Photogram. Rem. Sens. 158, 76–89. https://doi.org/10.1016/j.isprsjprs.2019.09.015.

Lohani, B., Mishra, R.K., 2007. Generating LiDAR data in laboratory: LiDAR simulator. In: ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007, pp. 264–269. In: https://www.isprs.org/proceedings/XXXVI/3-W52/final_papers/Lohani_2007.pdf.

Lovell, J.L., Jupp, D., Newnham, G.J., Coops, N.C., Culvenor, D.S., 2005. Simulation study for finding optimal LiDAR acquisition parameters for forest height retrieval. Forest Eco. Manag. 214, 398–412. https://doi.org/10.1016/j.foreco.2004.07.077.

Martínez Sánchez, J., Váquez Alvarez, A., López Vilarino, D., Fernández Rivera, F., Cabaleiro Domínguez, J.C., Fernández Pena, T., 2019. Fast ground filtering of airborne LiDAR data based on iterative scan-line spline interpolation. Remote Sens. 11, 2256. https://doi.org/10.3390/rs11192256.

Morsdorf, F., Frey, O., Koetz, B., Meier, E., 2007. Ray tracing for modeling of small footprint airborne laser scanning returns. international archives of the photogrammetry. Int. Arch. Photogr. Remote Sens. Spatial Inf. Sci. XXXVI (3/W52), 249–299. https://doi.org/10.3929/ethz-b-000107380.

North, P., 1996. Three-dimensional forest light interaction model using a Monte Carlo method. IEEE Tran. Geosci. Remote Sens. 34, 946–956. https://doi.org/10.1109/36.508411.

North, P.R.J., Rosette, J.A.B., Suárez, J.C., Los, S.O., 2010. A Monte Carlo radiative transfer model of satellite waveform LiDAR. Int. J. Remote Sens. 31, 1343–1358. https://doi.org/10.1080/01431160903380664.

Park, M., Baek, Y., Dinare, M., Lee, D., Park, K.H., Ahn, J., Kim, D., Medina, J., Choi, W. J., Kim, S., et al., 2020. Hetero-integration enables fast switching time-of-flight sensors for light detection and ranging. Sci. Rep. 10, 2764. https://doi.org/10.1038/s41598-020-59677-x.

Pinty, B., Gobron, N., Widlowski, J.L., Gerstl, S.A.W., Verstraete, M.M., Antunes, M., Bacour, C., Gascon, F., Gastellu, J.P., Goel, N., Jacquemoud, S., North, P., Qin, W., Thompson, R., 2001. Radiation transfer model intercomparison (rami) exercise. J. Geophys. Res.: Atmos. 106, 11937–11956. https://doi.org/10.1029/2000JD900493.

Previtali, M., Díaz-Vilariño, L., Scaioni, M., Frías Nores, E., 2019. Evaluation of the expected data quality in laser scanning surveying of archaeological sites. In: 4th International Conference on Metrology for Archaeology and Cultural Heritage, Florence, Italy, pp. 19–24. https://re.public.polimi.it/retrieve/handle/11311/1124569/476842/Previtali%20et%20al%202019%20MetroArchaeo.pdf.

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017. PointNet++: deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems. NIPS'17, Long Beach, California, USA, pp. 5099–5108. In: https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Ramey, R., 2004. C++ BOOST Serialization. URL: https://www.boost.org/doc/libs/1_72_0/libs/serialization/doc/index.html.

Ranson, K.J., Sun, G., 2000. Modeling LiDAR returns from forest Canopies. IEEE Tran. Geosci. Remote Sens. 38, 2617–2626. https://doi.org/10.1109/36.885208.

Rebolj, D., Pucko, Z., Babic, N.C., Bizjak, M., Mongus, D., 2017. Point cloud quality requirements for Scan-vs-BIM based automated construction progress monitoring. Auto. Construct. 84, 323–334. https://doi.org/10.1016/j.autcon.2017.09.021.

Schäfer, J., Faßnacht, F., Höfle, B., Weiser, H., 2019. Das SYSSIFOSS-Projekt: synthetische 3D-fernerkundungsdaten für verbesserte waldinventurmodelle. In: 2. Symposium Zur Angewandten Satellitenerdbeoachtung, Cologne, Germany. URL: https://www.dialogplattform-erdbeobachtung.de/downloads/cms/Stele3/EO-Symposium_Jannika.Schaefer.pdf.

Schlager, B., Muckenhuber, S., Schmidt, S., Holzer, H., Rott, R., Maier, F.M., Saad, K., Kirchengast, M., Stettinger, G., Watzenig, D., Ruebsam, J., 2020. State-of-the-art sensor models for virtual testing of advanced driver assistance systems/autonomous driving functions. SAE Int. J. Connect. Automat. Veh. 3, 233–261. https://doi.org/10.4271/12-03-03-0018.

Tulldahl, H.M., Steinvall, K.O., 1999. Analytical waveform generation from small objects in LiDAR bathymetry. Appl. Opt. 38, 1021–1039. https://doi.org/10.1364/ao.38.001021.

Tuong-Phong, B., 1973. Illumination for Computer-Generated Images. Technical Report 129, UTEC-CSC-73, Computer Science. https://doi.org/10.1145/360825.360839.

Vincent, G., Antin, C., Laurans, M., Heurtebize, J., Durrieu, S., Lavalley, C., Dauzat, J., 2017. Mapping plant area index of tropical evergreen forest by airborne laser scanning. a cross-validation study using LAI2200 optical sensor. Remote Sens. Environ. 198, 254–266. https://doi.org/10.1016/j.rse.2017.05.034.

Wagner, W., Roncat, A., Melzer, T., Ullrich, A., 2007. Waveform analysis techniques in airborne laser scanning. Int. Arch. Photogramm. Remote Sens. 36, 413–418. In: https://www.isprs.org/proceedings/XXXVI/3-W52/final_papers/Wagner_2007_keynote.pdf.

Wang, D., 2020. Unsupervised semantic and instance segmentation of forest point clouds. ISPRS J. Photogram. Rem. Sens. 165, 86–97. https://doi.org/10.1016/j.isprsjprs.2020.04.020.

Wang, D., Schraik, D., Hovi, A., Rautiainen, M., 2020. Direct estimation of photon recollision probability using terrestrial laser scanning. Remote Sens. Environ. 247, 111932. https://doi.org/10.1016/j.rse.2020.111932.

Wang, Y., Xie, D., Yan, G., Zhang, W., Mu, X., 2013. Analysis on the inversion accuracy of LAI based on simulated point clouds of terrestrial LiDAR of tree by ray tracing algorithm. In: IGARSS 2013-2013 IEEE International Geoscience and Remote Sensing Symposium. IEEE, Piscataway, pp. 532–535. https://doi.org/10.1109/IGARSS.2013.6721210.

Wavefront Technologies, 1992. Appendix B1. Object Files (.obj), Advanced Visualizer Manual.. http://fegemo.github.io/cefet-cg/attachments/obj-spec.pdf.

Weber, J., Penn, J., 1995. Creation and rendering of realistic trees. In: Mair, S.G., Cook, R. (Eds.), Computer Graphics Proceedings. Association for Computing Machinery, New York, NY., pp. 119–128. https://doi.org/10.1145/218380.218427

Weiser, H., Winiwarter, L., Anders, K., Fassnacht, F.E., Höfle, B., 2021. Opaque voxel-based tree models for virtual laser scanning in forestry applications. Remote Sens. Environ. 265, 112641. https://doi.org/10.1016/j.rse.2021.112641.

Widlowski, J.L., Mio, C., Disney, M., Adams, J., Andredakis, I., Atzberger, C., Brennan, J., Busetto, L., Chelle, M., Ceccherini, G., Colombo, R., Coté, J.F., Eenmäe, A., Essery, R., Gastellu-Etchegorry, J.P., Gobron, N., Grau, E., Haverd, V., Homolová, L., Huang, H., Hunt, L., Kobayashi, H., Koetz, B., Kuusk, A., Kuusk, J., Lang, M., Lewis, P.E., Lovell, J.L., Malenovský, Z., Meroni, M., Morsdorf, F., Mõttus, M., Ni-Meister, W., Pinty, B., Rautiainen, M., Schlerf, M., Somers, B., Stuckens, J., Verstraete, M.M., Yang, W., Zhao, F., Zenone, T., 2015. The fourth phase of the

radiative transfer model intercomparison (RAMI) exercise: actual canopy scenarios and conformity testing. Remote Sens. Environ. 169, 418–437. https://doi.org/10.1016/j.rse.2015.08.016.

Winiwarter, L., Mandlburger, G., Schmohl, S., Pfeifer, N., 2019. Classification of ALS point clouds using end-to-end deep learning. PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science 87, 75–90. https://doi.org/10.1007/s41064-019-00073-0.

Winiwarter, L., Esmorís Pena, A.M., Weiser, H., Anders, K., Sanchez, J.M., Searle, M., Höfle, B., 2021. 3dgeo-Heidelberg/helios: Version 1.0.0. Zenodo. https://doi.org/10.5281/zenodo.4452871.

Xiao, W., Zaforemska, A., Smigaj, M., Wang, Y., Gaulton, R., 2019. Mean shift segmentation assessment for individual forest tree delineation from airborne LiDAR data. Remote Sens. 11, 1263. https://doi.org/10.3390/rs11111263.

Zhang, Z., Li, J., Guo, Y., Yang, C., Wang, C., 2019. 3D Highway curve reconstruction from mobile laser scanning point clouds. IEEE Trans. Intell. Transp. 21 (11), 4762–4772. https://doi.org/10.1109/TITS.2019.2946259.

Zhou, Q.Y., Park, J., Koltun, V., 2018. Open3D: A Modern Library for 3D Data Processing. arXiv:1801.09847.

Zhu, X., Liu, J., Skidmore, A.K., Premier, J., Heurich, M., 2020. A voxel matching method for effective leaf area index estimation in temperate deciduous forests from leaf-on and leaf-off airborne LiDAR data. Remote Sens. Environ. 240 https://doi.org/10.1016/j.rse.2020.111696.