The Design and Implementation of a Ray-tracing Algorithm for Signal-level Pulsed Radar Simulation Using the NVIDIA® OptiX™ Engine

Mogamat Yaaseen Martin[1], Simon L. Winberg[1], Mohammed Yunus Abdul Gaffar[1], and David Macleod[2]
[1] University of Cape Town, Cape Town 7701, South Africa
[2] Centre for High Performance Computing, Cape Town 7700, South Africa
Email: mrtmog014@myuct.ac.za; simon.winberg@uct.ac.za; yunus.abdulgaffar@uct.ac.za; dmacleod@csir.co.za

Abstract —This paper presents the design and development of a ray-traced, signal-level pulsed radar simulation algorithm and a novel ray aggregation technique using the NVIDIA® OptiX™ engine for propagation modelling. This is expected to aid radar engineers and researchers in systems design, testing, and training. The program supports the short- and long-range simulation of various radar scenarios using complex target and signal models with a high degree of realism. This was achieved through raytraced radio propagation and geometry modelling, simulating realistic multipath signal interactions with mesh-based objects as opposed to the more traditional point-model approach. The designed algorithm has also been implemented as an open-source ray-tracing module and was verified against an established radar software package.

Index Terms—Ray tracing, software engineering, radar

I. Introduction

Signal-level radar simulators [1] are used to render the raw return signal that would be received by a radar system, i.e., the output signal after passing through an analogue-to-digital converter. These signals are computed using the system's parameters and the simulation environment. Signal-level simulators have thus been used for various applications throughout the industry, including Synthetic Aperture Radar techniques [2] and specialized systems design and testing [3].

However, most signal-level simulators are developed around point-model approximations for the targets and antennas in the simulation environment [4], [5], ignoring the complexities associated with object shape and material properties. This paper presents an alternative whereby ray tracing is used to compute signal information in multipath scenarios using complex targets and optics-based propagation. The relevant signal samples – and their computed characteristics – are then encoded into a rendered pulse signal in the form of the baseband signal's in-phase and quadrature forms.

Manuscript received March 12, 2022; revised August 17, 2022.
This work was supported through funding by the National Research Foundation (Grant Number 118727) and the Council for Scientific and Industrial Research (CSIR) in South Africa.
Corresponding author email: mrtmog014@myuct.ac.za.
doi:10.12720/jcm.17.9.761-768

A. Ray Tracing

Ray tracing is a widely-used concept in the field of electromagnetics [6] and graphics processing [7], and it is based on the approximation of a wave as a set of rays – each of which is launched from a source and then traced through a simulated environment, reflecting off or refracting through target objects under test [8]. This makes it well suited for modelling radio signals.

One benefit of ray tracing is that every ray is independent of all others, and thus each ray's path can be computed individually. This makes the model highly conducive to parallelism via acceleration structures as used in the NVIDIA® OptiX™ engine [9] – a freely-available, programmable ray-tracing engine that can be deployed and accelerated on supported NVIDIA® Graphics Processing Units (GPUs) for various uses.

The work in [10] detailed the development of an OptiXTM-based simulation tool for medical imaging using millimeter-wave systems. Other works [11]–[19] have also demonstrated the use of ray-tracing algorithms for various radar purposes such as Radar Cross Section (RCS) approximation, electric field modelling, and indoor beam tracing.

B. Scope of Contribution

Most of the aforementioned works are based on *electromagnetic* radar simulators that compute fields at discrete points and emphasize electrical behaviors. However, as far as the authors of this work are aware, there currently exist no 3-D ray-traced *signal-level* radar simulators – where "signal-level" refers to the computation of power, Doppler shifts, and time and phase delays in rendering a received signal.

In this paper, a technique for signal-level pulsed radar simulation (using OptiX™-based ray tracing) is thus designed and implemented. This is expected to be useful to engineers and researchers for training purposes and systems design. While similar work was published in [4], the implementation used point-model target approximations, did not account for phase shifts and refraction effects, and did not render any actual return signals. This work aims to address these deficiencies.

The design and implementation of this algorithm constitutes the main original contribution of this paper

with respect to the established literature, making use of a novel ray aggregation technique in combination with multipath power and Doppler implementations. This was achieved through optics-based models for radio signal propagation and geometric models for defining physical target features, providing an improvement over existing point-model simulators. This work also details the design of the ray-tracing algorithm as well as its verification against a well-established radar simulator – the Phased Array System Toolbox (PAST) in MATLAB® [20].

II. Algorithm Design

It was required that the designed software should operate using a single simulation definition file as the main input – specifying all simulation parameters and properties – and the simulator should then output the raw signal computed at each simulated receiver. This will be in keeping with the signal-level classification and will enable users to post-process the output signal in any way they choose.

A. Baseline Simulator

The ray-tracing implementation presented in this paper was achieved through a Ray Tracing Simulator (RTS) algorithm function that makes use of a combination of OptiX™, C and C++ host code, and Compute Unified Device Architecture (CUDA) [21] C++ device code. This was integrated into a modified version of the radar simulator presented in [1] – the Flexible Extensible Radar Simulator (FERS). This presented a strong baseline model that accounted for many typical radar effects, such as pulse simulation, system-target interactions, and antenna gain patterns.

FERS served as the baseline simulator upon which the RTS was built. The original software made use of a netted pulsed radar simulation scheme, and this work has expanded upon the foundation of FERS and integrated OptiX™ into its signal rendering algorithm for more advanced modelling of targets and propagation. FERS made use of a sampling model to read samples of the input pulse (including listening time) and transform them based on radar equations, but the RTS builds upon this using ray tracing to compute multiscatter properties as well. These are then incorporated into the received pulses using the ray-traced quantities for each sample.

B. The OptiX™ Framework

The RTS was integrated into a modified version of FERS as shown in the high-level operational diagram in Fig. 1. This depicts a basic operational overview of the RTS algorithm, with the host code being responsible for initializing variables and setting up the scene within the OptiX™ framework. The device code, which is launched via a kernel, is responsible for executing the ray-tracing algorithm; this includes launching rays from transmitters, propagating and tracing them through the environment, testing them for target intersections, and then aggregating them at receivers.

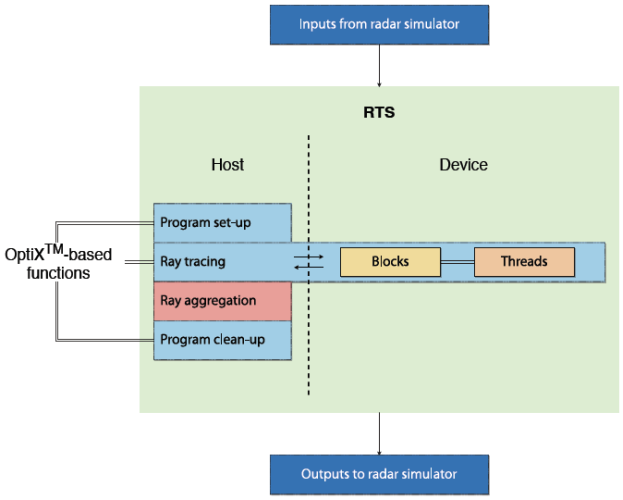

Fig. 1. Block diagram of the OptiXTM-based RTS depicting the implementation of code on the host and device (GPU).

C. Ray Simulation

Every radar simulation consists of at least one transmitter, receiver and target. The most crucial part of adding a ray-tracing algorithm into this configuration is computing the directions of rays, and it is important that the rays be distributed over a specified span to model wave dispersion in free space. This was implemented in the RTS by spherically directing rays around the transmitter's boresight vector through a user-specified beamwidth in both azimuth and elevation as presented in Fig. 2, where each node corresponds to a ray.

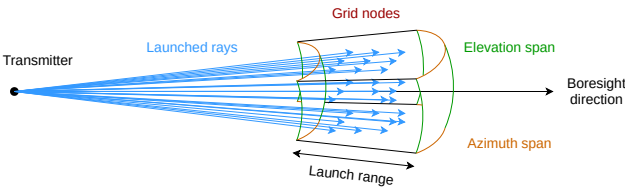

Fig. 2. A point source transmitter launching rays along its boresight vector with directions specified by a grid of 3-D virtual nodes spaced equally along spans in azimuth and elevation.

When a ray collides with a target, a partial reflection results as it bounces off the surface. This can also result in a *refracted* ray at the point of intersection, which then propagates *through* the object with reduced power. New ray directions are computed using Snell's Law [22] whenever a target – modelled as a tessellated mesh of triangles – is intersected by a ray.

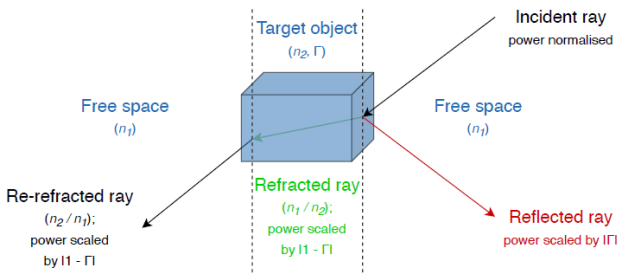

Fig. 3. An intersection occurring between a ray and a target, producing both reflected and refracted rays; a second refracted ray is also produced when the first refracted ray intersects the inside of the object.

Any changes in ray directions therefore need to be recorded at the point of intersection – along with any changes to other ray properties. Most notably, the ray experiences a change in power based on the intersected object's reflection coefficient, Γ – a user input to describe a target's material. If refraction occurs, the ray's power that was not reflected is thus transformed into refractive power as depicted in Fig. 3.

A ray will then continue to propagate until its end criterion is met when it (1) exceeds the maximum allowable number of intersections or (2) hits a receiving antenna. The RTS models receive antennas as surface patches of a sphere, defined through both azimuth and elevation ranges as shown in Fig. 4.

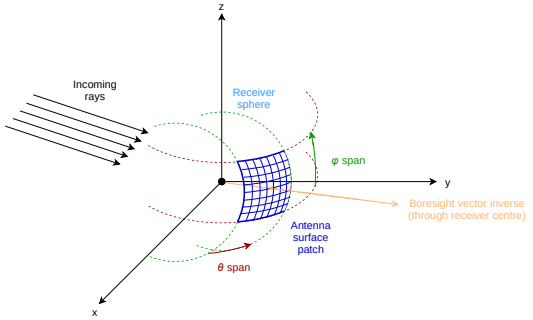

Fig. 4. Illustration of a 3-D receiving antenna modelled in the RTS as a spherical surface patch defined along spans in azimuth and elevation.

In this way, receivers are defined mainly by the radius of the sphere used to model the antenna as a surface patch, the centre coordinates of the sphere, and the angular span of the surface patch in both azimuth θ and elevation ϕ. The RTS will automatically orientate the surface patch so that it faces outward along its boresight direction, and any intersections between a ray and this surface patch are recorded.

III. Development and Implementation

The RTS module has been developed using several files and external dependencies, all of which are compiled together via CMake, NVCC and a C++ compiler. The RTS is independent of any specific baseline simulator and is intended to be a portable signal-level ray tracer that could be integrated into various radar simulators with minimal effort.

A. Kernel Set-up

Before the OptiX™ kernel is launched, input variables are transferred to the RTS on the host. At runtime, OptiX™ will compute, store, and track all relevant information related to each ray. This is achieved using a "Per Ray Data" (PRD) structure that updates whenever a ray intersects a target.

The kernel process begins with the ray-generation program launching on multiple device threads and computing ray directions. It then spawns the rays, initializes their PRDs and the output buffers, and begins ray tracing. Whenever a valid intersection then occurs between a ray and an object, the closest-hit program is invoked to update the ray's PRD and spawn new rays; the miss program is then also called for any rays that "miss" the scene or intersect a receiving antenna.

After the OptiX™ kernel has completed its execution, the captured rays' information is processed on the host as control is passed back from the device; then program clean-up occurs.

B. Multi-path Ray Computations

This subsection details multiple radar-based equations required to compute ray properties – such as received power and phase delays – as part of the implemented model. These quantities are then used to generate the complex-valued samples of the signal rendered at the output of the simulator.

1) Received power: The multistatic radar equation models the received power P_R for a multistatic radar and accounts for a target's RCS (user can select file-based directional RCS or mean isotropic RCS), propagation path loss, and other factors. However, this only holds true for single-scatter cases, where only one target is being considered. For multipath cases comprised of N targets, a generalised equation can be defined for P_R as follows [5]:

$$P_R = \left(\frac{1}{4\pi}\right)^{N+2} \frac{P_T G_T \sigma_1}{R_{T1}^2} \frac{G_R \lambda^2}{R_{NR}^2} \prod_{k=2}^{N} \frac{\sigma_k}{R_{(k-1)(k)}^2} \quad (1)$$

where P_T is the transmitted power, G_T is the transmitter gain in the direction of the first target, G_R is the receiver gain in the direction of the N_{th} target, σ_k is the mean RCS of the k_{th} target, R_{T1} is the range between the transmitter and the first target, R_{NR} is the range between the receiver and the N_{th} target, and λ is the wavelength. The power would also be scaled by Γ if the ray is reflected or $1 - |\Gamma|$ if refracted.

2) Time and phase delays: The time delay τ is given by:

$$\tau = \frac{R_{RL}}{c} \quad (2)$$

where R_{RL} is the total ray length as summed at the point of ray termination at the receiver intersection and c is the speed of light. In narrowband cases, this is also the group delay and is related to the phase delay θ_d through [5]:

$$\theta_d = -(2\pi\tau f_c) \bmod (2\pi) \quad (3)$$

where f_c is the carrier frequency.

3) Doppler shift: For a scenario with N targets, a transmitter and a receiver, the multiscatter Doppler frequency f_d is computed using an iterative summation process [23]. If the transmitter launches a ray with direction $\hat{k}_0$ and the j_{th} target moves with a velocity v_j, the incident ray's direction changes to $\hat{k}_j$ upon intersection. Any intersection thus results in a scattering event producing a Doppler shift – dependent upon the target's velocity along the direction $\hat{k}_j - \hat{k}_{j-1}$. Summing these contributions results in the Doppler frequency:

$$f_d = \frac{1}{\lambda}\left(\sum_{j=1}^{N} \vec{v_j} \cdot (\hat{k_j} - \hat{k_{j-1}})\right) \quad (4)$$

where this represents a modified version of the equation derived in [23] and assumes the convention that a receding target corresponds to a negative Doppler frequency.

4) Ray aggregation: Rays' received powers need to be aggregated to compensate for the increase in power at the receiver based on the number of rays being traced. In essence, this is achieved by scaling each ray's voltage by a factor of $1/N_{pathi}$, where N_{pathi} is the number of rays that followed the same intersection path as ray i. However, this can also be extended to the other ray measurements by averaging all rays that follow the same path.

This is achieved in the RTS by averaging the measured quantities of all rays that intersect the same targets (and receiver) in the same order. Tracking these paths is computationally complex, however, and as such, this was implemented across two CUDA kernels as illustrated in Fig. 5.

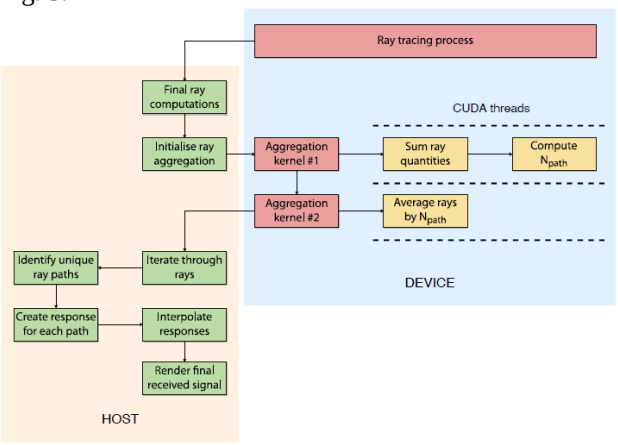

Fig. 5. Flowchart showing the ray aggregation process used in the RTS.

Specifically, two CUDA-based aggregation kernels are used to iterate through all rays and identify common paths. The first kernel is responsible for computing the value of N_{pathi} for each ray i as well as summing the power, Doppler, and time and phase delays; a second kernel is thereafter used to synchronize all threads and scale the summations by $1/N_{pathi}$. The drawback to this approach is that it requires updating the path order for each ray across thousands of threads, leading to an extensive process with significant overhead – but this is necessary to ensure signal accuracy.

C. System Operation

As a whole, the simulator operates as presented in Fig. 6. In particular, the RTS is invoked by the baseline simulator and the scene is initialized; ray tracing follows, and the output ray properties are recorded for each receiver based on the captured rays. Finally, ray aggregation is used to generate the signal based on the transmit pulse and ray responses – which are used to interpolate samples along the output signal.

Various results were compared between PAST and the RTS – two of which are presented in this section. Ultimately PAST lacks some features present in the RTS, such as ray tracing and fractional delay filtering, but it has notably been used in other published works before [24] — and the software was previously used for verification purposes [25].

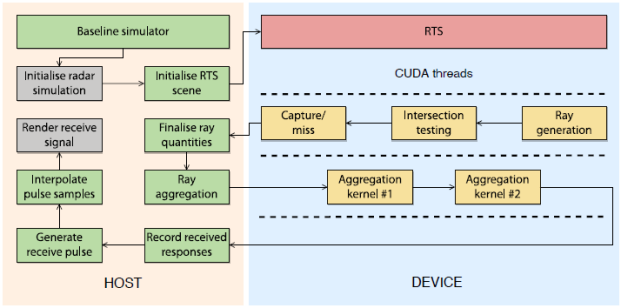

Fig. 6. Flowchart depicting the logic and order of operations in producing a received signal using the RTS.

IV. Simulator Verification

A. System Configuration

The RTS was tested with NVIDIA® OptiX™ 5.1.1 and CUDA 10.0 with Driver Version 410.48 – these were chosen as they were the latest versions compatible with the available hardware. All testing was conducted on a high-performance GPU node running CentOS 7 with 132 GB of memory, an Intel® Xeon® E5-2695 v3, and a NVIDIA® Tesla V100.

B. Experiment #1: Multi-ray Tracing

The simulation parameters used to validate the RTS are presented in Table I for a monostatic radar (positioned at the origin) transmitting 16 pulses to one target.

Table I: Simulation Parameters Used in the Multi-ray Tracing Experiment to Validate the RTS

arameter	Value	Units
Number of rays	10^6	—
Number of pulses	16	—
Carrier frequency	1	GHz
Transmit power	10	kW
Transmitter boresight (θ, φ)	(90, 90)	Deg
Receiver span (θ, φ)	(90, 90)	Deg
Pulse width	2	μs
Pulse repetition interval	200	μs
Mean RCS	15	cm^2
Reflection coefficient	0.95	—
Refractive index	3.0	—

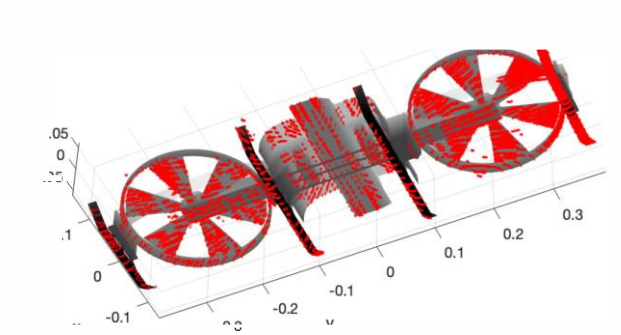

Fig. 7. Drone target mesh and the ray-target intersection coordinates (marked in red) for Experiment #1's first transmit pulse in the RTS.

The target in this simulation was defined by a custom mesh modelling a drone [26] (shown later in Fig. 7) that was centred at (x, y, z) = (0, 0, 100) with a velocity of +20 m/s on the z-axis (away from the radar). The radar was placed at the origin and set to face directly towards the target.

The objective of this was to run near-identical simulations in the RTS and PAST, and thereafter to compare the results. The set-ups of these simulations are illustrated in Fig. 8, showing their implementations in the RTS as well as PAST.

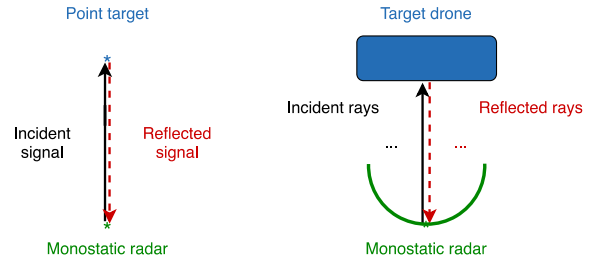

Fig. 8. Side view of a multi-ray validation test scenario in PAST (left) and the RTS (right); diagram is not to scale.

The main discrepancies between the simulations are that: (1) PAST uses one "ray" mapped directly from a radar to a point target and back again, whereas the RTS can use millions of rays that vary in direction; (2) PAST, like most simulators, makes use of point model approximations for targets, whereas the RTS uses tessellated meshes comprised of triangular primitives [10]. The latter allows the RTS to distinguish between varying material types and accurately account for an object's volume.

With this, the RTS implementation can be verified by comparing both simulators' received signals as depicted in Fig. 9. This shows both the power and phase for the fourth received pulse from each simulator. Note that the gains in PAST were manually set so as to similarly model the RTS's antenna gain pattern and boresight.

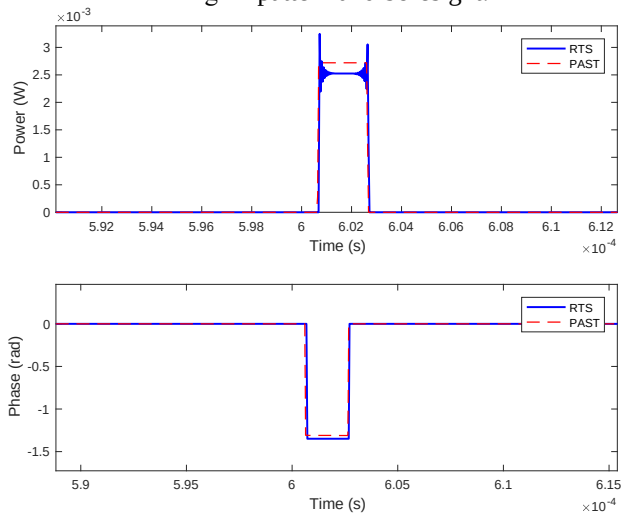

Fig. 9. Power and phase components of the 4th received pulses from both the RTS and PAST for a simple multi-ray test simulation.

As shown, the signals agree quite closely. This demonstrates how both rendered signals match up with a high degree of accuracy, showing that the RTS is able to nearly replicate PAST results while also using more realistic target and propagation models and accounting for refraction, target and receiver shape, and object reflectivity. The root mean square error (RMSE) was also averaged across the in-phase and quadrature components and was found to be 1.96×10^{-4}.

However, there are notable peaks present at the edges of each pulse in the RTS signal that PAST does not replicate. These are attributed to the fractional-delay filtering technique [5] applied in the RTS, accounting for the round-trip time of a pulse not being constant in reality – as is often assumed by other radar simulators. Additionally there are slight power differences shown in Fig. 9 as a result of refraction and the reflection coefficient used – which PAST does not account for – as well as phase differences due to varying ray delays.

Further results are depicted in Fig. 7, which shows the ray hit-point coordinates along the underside of the drone mesh. This demonstrates the spread of the rays in the intersections of the rays with the target mesh, and thus also the spread of the received rays. In particular, this emphasizes the improved realism in target and propagation simulation relative to PAST and similar radar simulators, which make use of only one "ray" that is mapped directly to point models for both the target and receiver.

C. Experiment #2: Multi-target Simulation

This experiment considers the simulation of two targets in a monostatic scenario. In this way, the effects of simulating multiple targets can be tested and a range-Doppler map can be used to verify the returns. Most of the same parameters from Table I were reused, but for simplicity, refraction was disabled, 10^3 rays were used, the reflection coefficient was set to 1, and the noise temperature was set to 20 K (to demonstrate the RTS's noise capability).

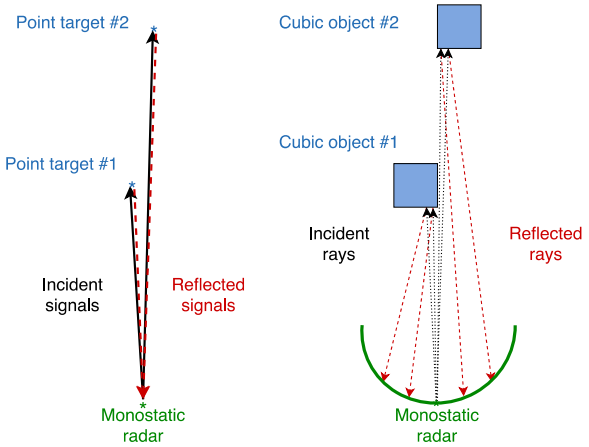

Fig. 10. Side view of a basic multi-target test scenario in PAST (left) and the RTS (right); diagram is not to scale.

Additionally, the drone from Experiment #1 was replaced with a pair of cuboid meshes; one was positioned at 5 km on the z-axis (with a velocity of 100

m/s *away* from the system), and the other was placed at 10 km on the *z*-axis (with a velocity of 50 m/s *towards* the system). The two targets were also slightly separated along the *y*-axis so that neither target shadowed the other. The geometry of this scenario is depicted in Fig. 10.

After simulating the scenario in both simulators, the rendered receive signal voltages are shown (for the twelfth transmit pulse) in their in-phase and quadrature forms in Fig. 11. As demonstrated, the results show strong agreement with an average RMSE of 3.70×10^{-9}.

The range-Doppler maps in Fig. 12 also show how the range and Doppler velocity components agree with the input parameters for both targets across both simulators – with only minor variations due to noise and the inherent simulator design differences.

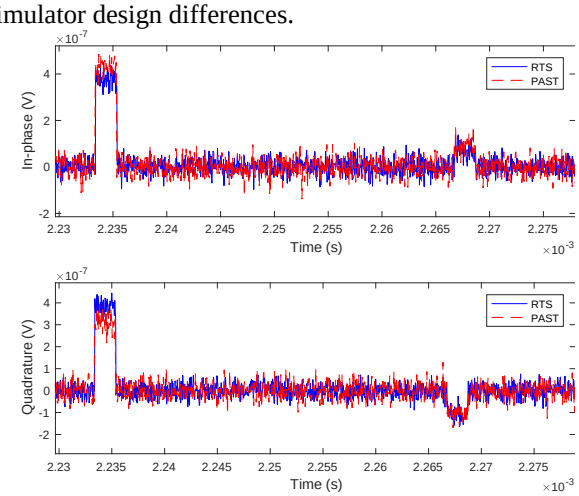

Fig. 11. In-phase and quadrature components of the received responses to the 12th transmit pulse for two targets in the RTS and PAST.

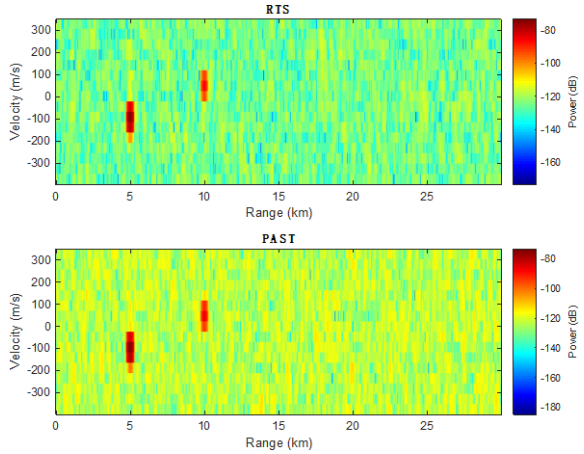

Fig. 12. Comparison between the range-Doppler maps for the RTS and PAST for a multi-target test simulation.

This section has thus demonstrated how the RTS: (1) reflects and refracts signals based on optics principles, (2) models targets using physical parameters such as geometric and material properties, (3) models the physical properties of receiving antennas, and (4) accounts for system noise in the output signal. Together, these allow the RTS to realistically model signal propagation and geometries with a small degree of error relative to PAST, showing strong agreement with the theory despite the discrepancies between the systems.

These simulations confirm that ray tracing is a viable mechanism for signal-level radar simulation and shows promising results for improving the complexity of modelled scenarios. However, it remains important to note the discrepancies between the RTS and a typical radar simulator; specifically, it is not always possible to replicate the same results in both programs. With that said, all tested scenarios (some of which were not listed here) showed close agreement between the simulators.

V. Conclusion

This paper presented a novel technique for ray aggregation in multipath power and Doppler computation, and the development of a ray-tracing program module for signal-level radar simulation. It highlighted the design and implementation of the Ray Tracing Simulator, which enables ray tracing in radar experiments and allows for complex simulations involving realistic models for targets and signal propagation with multiscatter capabilities.

The results have evaluated the implementation of the simulator against established theory and the PAST simulation package. Through comparison, the ray-tracing implementation was validated and the post-processed signals showed that the simulators' results are in close agreement despite their inherent differences. Future improvements could be made to the software by accounting for wave polarization, fluctuating RCS, and ray diffraction to increase modelling accuracy.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

M. Y. Martin conducted the research and experiments shown in this work, developed the software used to generate the results, and wrote the bulk of the paper; S. L. Winberg, M. Y. Abdul Gaffar, and D. Macleod served as supervisors on the project, contributed ideas and suggestions towards the work, and provided revisions and edits to the paper. All authors had approved the final version.

Acknowledgment

The authors wish to thank the Council for Scientific and Industrial Research (CSIR) in South Africa for providing funding towards this project as well as access to their computing hardware for use in the experiments conducted in this paper. This work was also supported through funding by the National Research Foundation (Grant Number 118727).

References

[1] M. Brooker and M. Inggs, "A signal level simulator for multistatic and netted radar systems," *IEEE Transactions*

on Aerospace and Electronic Systems, vol. 47, no. 1, pp. 178–186, 2011.
[2] G. Franceschetti, M. Migliaccio, D. Riccio, and G. Schirinzi, "SARAS: A Synthetic Aperture Radar (SAR) raw signal simulator," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 30, no. 1, pp. 110–123, 1992.
[3] M. Y. Martin, S. L. Winberg, M. Y. Abdul Gaffar, and D. Macleod, "The design and development of a GPU-accelerated radar simulator for space debris Monitoring," in *Proc. 5th High Performance Computing and Cluster Technologies Conference*, HPCCT 2021, p. 27–40.
[4] D. Gubelli, O. Krasnov, and O. Yarovyi, "Ray-tracing simulator for radar signals propagation in radar networks," in *Proc. European Radar Conference*, 2013, pp. 73–76.
[5] M. Brooker, "The design and implementation of a simulator for multistatic radar systems," PhD thesis, University of Cape Town, 2008.
[6] C. A. Balanis, *Advanced Engineering Electromagnetics*, John Wiley & Sons, 2012.
[7] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann, 2016.
[8] A. Zubaroglu, "GPU accelerated radio wave propagation modeling using ray tracing," Master's thesis, Middle East Technical University, 2014.
[9] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, *et al.*, "OptiX™: A general purpose ray tracing engine," in *ACM Transactions on Graphics (TOG)*, vol. 29, p. 66, ACM, 2010.
[10] K. Williams, L. Tirado, Z. Chen, B. Gonzalez-Valdes, *et al.*, "Ray tracing for simulation of millimeter-wave whole body imaging systems," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 12, pp. 5913–5918, 2015.
[11] Z. Yun and M. F. Iskander, "Ray tracing for radio propagation modeling: Principles and applications," *IEEE Access*, vol. 3, pp. 1089–1100, 2015.
[12] G. Xu, C. Dong, T. Zhao, H. Yin, and X. Chen, "Acceleration of shooting and bouncing ray method based on OptiX and normal vectors correction," *Plos One*, vol. 16, no. 6, p. e0253743, 2021.
[13] D. He, B. Ai, K. Guan, L. Wang, *et al.*, "The design and applications of high-performance ray-tracing simulation platform for 5G and beyond wireless communications: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 10–27, 2018.
[14] Y. Tao, H. Lin, and H. Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Transactions on Antennas and Propagation*, vol. 58, no. 2, pp. 494–502, 2009.
[15] R. Felbecker, L. Raschkowski, W. Keusgen, and M. Peter, "Electromagnetic wave propagation in the millimeter wave band using the NVIDIA® OptiX™ GPU ray tracing engine," in *Proc. 6th European Conference on Antennas and Propagation (EUCAP)*, 2012, pp. 488–492.
[16] Y. Liu, D. Shi, A. Li, and P. Zeng, "Radio Wave Propagation Prediction Based on the NVIDIA OptiX GPU Ray Tracing Engine," in *Proc. IEEE 6th International Symposium on Electromagnetic Compatibility (ISEMC)*, 2019, pp. 1–3.
[17] J. S. Lu, E. M. Vitucci, V. Degli-Esposti, F. Fuschini, *et al.*, "A discrete environment-driven GPU-based ray launching algorithm," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 2, pp. 1180–1192, 2018.
[18] J. Tan, Z. Su, and Y. Long, "A full 3-D GPU-based beam-tracing method for complex indoor environments propagation modeling," *IEEE transactions on antennas and propagation*, vol. 63, no. 6, pp. 2705– 2718, 2015.
[19] S. Auer, R. Bamler, and P. Reinartz, "RaySAR-3D SAR simulator: Now open source," in *Proc. IEEE International Geoscience and Remote Sensing Symposium*, 2016, pp. 6730–6733.
[20] N. Tucker, "Phased array design toolbox V2.3 for MATLAB," Technical Report, 2009.
[21] D. Kirk., "NVIDIA® CUDA software and GPU parallel computing architecture," *ISMM*, vol. 7, pp. 103–104, 2007.
[22] R. W. Scharstein, "Visualization and interpretation for the electromagnetic derivation of Snell's laws," *IEEE Transactions on Education*, vol. 41, no. 4, pp. 286–292, 1998.
[23] A. Battaglia and S. Tanelli, "DOMUS: DOppler multiple-scattering simulator," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 1, pp. 442–450, 2010.
[24] M. Roggero, J. Zhao, and G. Zucchelli, "Design of FMCW radars for active safety applications," in *Proc. Embedded World Conference*, 2015.
[25] A. A. Qasim and A. H. Sallomi, "Design and analysis of phased array system by MATLAB toolbox," *Al-Kitab Journal for Pure Sciences*, vol. 4, no. 1, 2020.
[26] D. Haupt. (March 2017). Controllable drone – blender game engine 3D model. [Online]. Available: https://free3d.com/3d-model/controllabledrone-blender-game-engine-67381.html

Mogamat Yaaseen Martin received the B.Sc. Eng. degree in Electrical Engineering from the University of Cape Town (UCT) in 2018, and he is currently pursuing his Ph.D. in the same field, with funding from the Council for Scientific and Industrial Research in South Africa. His research interests include signal processing, space science, and high-performance computing.

Simon Lucas Winberg is a lecturer in the Department of Electrical Engineering at UCT. He worked in the embedded systems industry in both South Africa and in the United States prior to embarking on his Ph.D. studies and subsequently becoming an academic staff member at UCT. His research areas are hybrid computer systems, high-performance computing, and FPGA-based reconfigurable computers for application to radar systems and radio astronomy.

Mohammed Yunus Abdul Gaffar received the B.Sc. Eng. and M.Sc. Eng. degrees in electronic engineering from the University of Natal in 2002 and 2003 respectively, and subsequently received his Ph.D. from the University of Cape Town. He was with the Council for Scientific and Industrial Research from 2003 to 2016, where he was active in the fields of Inverse Synthetic Aperture Radar and radar detection of ground moving objects. In 2016, he joined the University of Cape Town as a Senior Lecturer. His research interests are in the fields of short-range radar systems and radar signal processing.

David Macleod received the B.Sc. Eng. and M.Sc. Eng. Degrees from the University of Cape Town in 2008 and 2011 respectively. He leads of a team of engineers and technologists at the Centre for High Performance Computing (a subsidiary of the Council for Scientific and Industrial Research) and conducts research, development and evaluation for high-performance and cloud computing technologies. His research interests include high-performance and cloud computing as well as cyber-infrastructure.