GPU-Accelerated Radar Simulation for Large-Scale Machine Learning Dataset Generation and Algorithm Development

Andrew R. Willis, Srini Vasan

Vulcan Ventura

Concord, NC USA

andrew.willis@vulcanventura.com, srini@quantumventura.com

Braden R. Feshami

Quantum Ventura Inc.

San Jose, CA USA

braden@quantumventura.com

ABSTRACT

Radar systems use high-power antennas to illuminate target locations with electromagnetic radiation, e.g., 10GHz radio waves, and illuminated surface backscatter is sensed by the antenna which is then used to generate images of structures. Real radar data is difficult and costly to produce and, for research, lacks a reliable source ground truth. Accurate radar simulation tools provide an important alternative opportunity to develop radar applications at low cost. Important applications of these tools include generating large-scale machine learning (ML) datasets and developing new radar imaging or target recognition algorithms. This work presents a high-fidelity radar simulation framework featuring a Shooting and Bouncing Ray (SBR) electromagnetic solver that leverages NVIDIA RTX GPUs and the OptiX library for hardware-accelerated ray tracing. The simulator enables scalable computation through distributed processing, utilizing high-speed network fabric such as 200 Gbps optical links and RDMA data transfer to achieve unprecedented performance. The proposed framework supports the synthesis of physically accurate RF backscatter waveforms and complex scattering interactions in urban and natural environments. By integrating physics-based modeling with high-performance computing, the simulator provides a powerful tool for radar researchers and system engineers. Furthermore, the simulator facilitates rapid prototyping and validation of advanced radar waveforms, including adaptive and cognitive radar techniques. Through its scalable architecture, the proposed radar simulator bridges the gap between theoretical model development and real-world implementation, significantly enhancing radar system design and evaluation. This work demonstrates how modern GPU-accelerated computing and high-speed networking can transform electromagnetic simulation, paving the way for next-generation radar research.

ABOUT THE AUTHORS

Andrew Willis is one of the top experts in Synthetic Aperture Radar (SAR) simulation and NVIDIA GPUs utilizing the OptiX Ray Tracing engine. He also holds appointments as Associate Professor in the Electrical and Computer Eng. Dept. at UNC Charlotte and as a Research Professor in the Mechanical and Aerospace Dept. at Univ. of Florida. Dr. Willis performs research on SAR, computational vision, machine learning, and other problems favoring 3-dimensional problems that originate from these fields. He has 5 degrees including a PhD in engineering from Brown University, with over 2 decades of experience in SAR simulation.

Srini Vasan is the President & CEO of Quantum Ventura Inc and CTO of QuantumX, the R&D arm of Quantum Ventura Inc. He is an expert in AI/ Advanced Machine Learning, AI/ML Verification & Validation & Quality Assurance, rigorous testing, ML performance measurement and large-scale system development. He has an MBA from MIT Sloan School.

Braden Feshami has a background in theoretical condensed matter physics and has worked in photonic metamaterials, real-time RF scene generation, AI algorithms for RF direction finding, signal detection, and identification, and novel software for simulating 6DOF trajectories of store separation from tactical aircraft. He received his PhD in physics in 2020 from Indiana University with a focus on electronic properties of graphene in the quantum Hall regime.

GPU-Accelerated Radar Simulation for Large-Scale Machine Learning Dataset Generation and Algorithm Development

Andrew R. Willis, Srini Vasan

Vulcan Ventura

Concord, NC USA

Author1@email.com, srini@quantumventura.com

Braden R. Feshami

Quantum Ventura Inc.

San Jose, CA USA

braden@quantumventura.com

INTRODUCTION

Modern radar systems produce vast amounts of complex scattering data, and simulating this data for electrically large environments, where target dimensions span many wavelengths, poses a significant computational challenge due to high frequencies and fine spatial resolutions. This difficulty is compounded when large datasets are required to train machine learning (ML) models for tasks such as automatic target recognition or synthetic aperture radar (SAR) image classification. Traditional methods of acquiring radar data through extensive field measurements or high-fidelity electromagnetic (EM) solvers are time-consuming, costly, and impractical for large-scale, high-resolution scenes. High-fidelity simulation of radio frequency (RF) backscatter is critical for radar system design, autonomous navigation, and RF sensing algorithm development, yet EM solvers are often prohibitively expensive for rapid prototyping. Therefore, a faster simulation framework is needed to efficiently predict realistic RF backscatter characteristics from complex scenes and support scalable model training and system analysis. By reducing computational costs, such a framework would enable more effective development of radar sensing algorithms and facilitate autonomous system validation.

Motivation & Applications

The ability to simulate RF scene returns with high spatial and temporal resolution has applications in Synthetic Aperture Radar (SAR) modeling, radar cross section (RCS) analysis, electronic warfare simulation, and training data generation for machine learning systems. These simulations are crucial in defense and aerospace contexts where airborne or spaceborne radar systems must operate in environments without GPS access or where access is denied. Accurate backscatter simulation also supports hardware-in-the-loop (HWIL) testing and embedded system development, ensuring that radar systems can be validated under realistic operational conditions without relying on extensive field trials.

A high-speed EM simulator can accelerate both the creation of machine learning datasets and the development of radar signal processing algorithms by generating synthetic radar returns from a variety of virtual scenarios and bypassing the limitations of real-world data collection. This capability enables training deep learning models for aircraft detection and stealth assessment, simulating weather phenomena for meteorological radar, generating data for autonomous vehicle sensors and enhancing surveillance and reconnaissance systems. In defense and security contexts, large-scale simulators support machine learning for radar signature modeling, stealth technology evaluation, and SAR and inverse SAR imaging. Such a simulator serves both civilian and military domains by providing an efficient means to generate high-quality synthetic radar data on demand.

Shortcomings of Existing Approaches

Full wave EM solvers such as finite-difference time-domain (FDTD), finite-element, and method-of-moments (MoM) tools offer high accuracy but scale poorly with scene size and frequency, making them impractical for electrically large, high-frequency scenarios without enormous computing resources. Asymptotic ray-based techniques like shooting and bouncing ray (SBR) improve scalability and are used in industry-standard codes such

as XPatch, Feko RL GO and Ansys HFSS SBR+, but existing implementations often run on CPU clusters, lack real-time capabilities, handle few bounces or materials and neglect phenomena such as diffraction and polarization unless manually tuned. These limitations hinder rapid dataset generation, interactive algorithm development and integration with custom sensors, scene dynamics or modern machine learning pipelines. A modern radar simulator must combine high fidelity, speed and scalability by leveraging cutting-edge hardware, automatically support complex scene variations and seamlessly integrate with machine learning workflows to overcome the performance bottlenecks of prior approaches.

Contributions of This Work

This work introduces a novel RF backscatter simulator that leverages a GPU-accelerated SBR pipeline using the NVIDIA OptiX ray-tracing engine. Our work is based on the development of an open-source SAR simulator that harnesses NVIDIA's OptiX ray-tracing library and RTX GPU hardware to compute phase histories for arbitrary 3D scenes at unprecedented speeds (Willis et al. 2020). The simulator constructs virtual SAR environments from user-defined descriptions of scene geometry, antenna parameters, signal processing settings, and vehicle trajectories. By employing massively parallel GPU computation, it accelerates ray tracing to calculate slant ranges and backscatter intensities for each pulse, overcoming the limitations of CPU-based tools. To address single precision constraints of OptiX, frequency offsets are reformulated to preserve numerical accuracy. Experimental results demonstrate orders-of-magnitude performance gains over MATLAB implementations while maintaining phase-history fidelity, enabling rapid generation of realistic SAR data for RF simulation research

The proposed simulator is designed to generate radar backscatter data from large-scale scenes with unprecedented speed and detail. Its scalable compute architecture, which leverages RDMA over converged ethernet (RoCE) for low latency network communication, has been refined to distribute backscatter response calculations across multiple network nodes equipped with multiple GPUs. As a particular example, our system can easily scale to calculate backscatter responses for 1 million target scatterers located within complex 3D environments at rates of 10,000 times per second. Key contributions include:

1. **Extensive GPU utilization:** We exploit NVIDIA RTX GPUs and their ray-tracing cores via the OptiX library to perform EM raytracing and scattering computations entirely on the GPU. By invoking hardware-accelerated ray intersections and massively parallel field calculations, the simulator achieves order-of-magnitude speedups.
2. **High-fidelity SBR modeling:** Our approach implements the SBR method with multi-bounce reflections and uses physical optics (PO) integration via the Stratton–Chu formula to compute scattered fields. This yields accurate far-field backscatter while maintaining efficiency for electrically large targets. We address the limitations of basic SBR by incorporating corrections for faceted geometry through normal-vector smoothing and laying the groundwork for including edge diffraction in future work.
3. **Large-scale dataset generation:** The simulator can rapidly produce labeled radar returns for complex scenes (terrain, vehicles, etc.), enabling the creation of extensive machine-learning training datasets. We demonstrate its capability by simulating an orbiting radar on an AC-130 gunship collecting data from a scene with detailed terrain and target vehicles, generating realistic SAR or backscatter frames in seconds.
4. **Performance benchmarking and scalability:** We provide a comprehensive performance evaluation across multiple GPU architectures (RTX 3090 Ti, RTX 4090 Ti, and an RTX 4000 Ada generation card). The results highlight near-linear scaling with GPU-compute capability and show that even a single high-end GPU can simulate complex radar scenes in real time. This paves the way for integrating the simulator into real-time algorithm development and HWIL testing for advanced radar systems.

Together, these contributions advance the state-of-the-art in EM simulation by delivering a tool that is both fast and physically accurate, tailored for the dual demands of machine learning data generation and radar research.

BACKGROUND

EM Simulation of Electrically Large Scenes

Simulating EM scattering for large scenes such as terrain with vehicles or urban environments poses challenges because these scenes are electrically large relative to the wavelength. Full-wave methods such as FDTD or MoM

require extremely fine spatial discretization and enormous memory and runtime resources, making them infeasible at high frequencies or over kilometer-scale domains. When target dimensions span dozens or hundreds of wavelengths, full-wave solvers become intractable as the number of unknowns and the computational cost grows faster than linearly with frequency and size. Therefore high-frequency asymptotic methods are employed. The SBR approach models wave propagation and scattering using rays rather than solving field equations everywhere, drastically reducing computation while maintaining accuracy for large scenarios.

SBR Asymptotic Solvers, Their Benefits, and Shortcomings

SBR-based methods model EM wave propagation by tracing rays through a geometric scene, making them well suited for high-frequency analysis and efficient scaling with scene complexity, but conventional SBR lacks wave interference and diffraction modeling and typically simplifies scattering physics through geometric approximations. Originating from geometrical optics and physical optics, SBR was developed for RCS calculations of complex shapes in the late 1980s. The simulation begins by launching a dense grid of rays from the transmitter or incident wavefront, and as each ray strikes object surfaces it can reflect or refract and continue bouncing through the scene. At reflection points, geometric optics determine the ray paths and physical optics approximate induced surface currents, which represent how the surface re-radiates energy.

The scattered field is then obtained by integrating all equivalent currents using formulations such as the Stratton–Chu integral or Kirchhoff's diffraction formula. This two-step process enables accurate far-field radar return predictions with far less computational effort than full-wave solvers. SBR is most accurate in the high-frequency limit, where wavelengths are very small relative to object geometry and assumptions of local flatness hold, though it neglects diffraction and other wave effects not captured by simple specular reflections. Extensions like the Uniform Theory of Diffraction or Physical Theory of Diffraction can be incorporated to account for energy scattered from edges and corners. Despite limitations, SBR solvers have proven highly effective for radar signature prediction of large targets and form the foundation of the simulator in this work.

Born Approximation & Stratton-Chu Integral Equation

The Born approximation is a linearization technique that assumes the field inside a scatterer equals the incident field, making single scattering and negligible self-interaction assumptions. This simplifies the scattering problem into a linear one by integrating induced polarization or current without feedback, which is computationally efficient and widely used in radar imaging such as SAR or tomography. However, it fails for strong multiple scattering or high contrast interfaces, requiring a higher order iterations or full wave solutions. In our simulator, which deals with metallic strongly scattering objects, the Born approximation is not used, since SBR and physical optics inherently allow multiple reflections. SBR can be viewed as a multi-bounce extension beyond the first Born approximation, formulated via ray tracing rather than volume integrals.

The Stratton–Chu integral provides a full wave surface integral formulation that computes EM fields in space based on tangential fields on a closed surface, enabling accurate synthesis of backscattered waveforms including constructive and destructive interference. This makes it well suited for coupling with SBR ray tracing to approximate field integrals from bouncing rays. Essentially, Stratton–Chu expresses scattered fields in terms of integrals over surface currents derived from Green's identities, and in the far field it simplifies to a diffraction integral where each surface element contributes a phase-weighted secondary wave. In our simulator, after ray tracing we obtain a collection of hit points on object surfaces, each with a local current induced by the incident wave, and we compute backscatter by summing contributions from all equivalent current sources.

RELATED WORK

Commercial full wave tools such as CST Microwave Studio, HFSS, and FEKO offer high accuracy, but suffer from exponential growth in resource requirements, and even with domain decomposition and adaptive meshing they remain unsuitable for rapid scene-level backscatter simulations, especially in dynamic or time critical contexts. Before the widespread adoption of asymptotic methods, researchers relied on full-wave EM solvers to simulate radar scattering. Methods such as MoM and FDTD can exactly solve Maxwell's equations for a given target and remain the gold standard for small or moderate-sized problems where high fidelity is paramount. However, their computational complexity and memory usage grow at least on the order of N squared or N cubed for N unknowns,

making electrically large targets beyond reach and parametric studies or large dataset generation infeasible, since a single run might take many hours or days. This motivated the development of faster high-frequency methods and the adoption of SBR approaches.

SBR Solvers Reported in the Literature

Notable SBR solvers include XPatch, Feko's RL GO module, and Ansys Electronic Desktop's SBR+. XPatch, developed in the 1990s for the U.S. Department of Defense, predicts far‑field and near‑field radar signatures of three‑dimensional targets, including RCS, range profiles, SAR images, and three‑dimensional scattering center models. Feko's RL GO uses ray launching with geometrical optics and physical optics for scattering, allowing RCS analysis of large platforms beyond the scope of its method‑of‑moments solver, and can be hybridized with MoM for parts of a model. Ansys HFSS SBR+, originally Delcross Savant, employs SBR for antenna placement and RCS modeling on electrically large platforms. These tools implement efficient geometric ray tracing with multipath support and some near‑to‑far‑field transforms, but they rely predominantly on CPU resources, limit user extensibility, and often require proprietary model formats or interfaces that restrict research workflows. Academic work has optimized SBR algorithms using advanced data structures (for example k-d trees and bounding volume hierarchies) and parallel processing to handle millions of rays, demonstrating that SBR offers an excellent trade‑off between accuracy and efficiency for high‑frequency scattering problems.

GPU-Accelerated Approaches

Only in recent years have SBR solvers begun to fully leverage GPU acceleration. Early implementations were CPU-bound, but the parallel nature of ray tracing and independent calculations for each ray bounce make the problem well-suited to GPUs. Meng et al. (2011) implemented a CUDA-based SBR and reported over 150× speedup compared to a CPU for computing field profiles. More recently, Xu et al. (2021) introduced an OptiX-based SBR with normal vector corrections for curved surfaces, achieving up to 60× faster simulation than a traditional CPU ray tracing approach. Ansys HFSS SBR+ has also incorporated GPU support in its latest versions, claiming up to 100× acceleration using NVIDIA hardware. These developments underscore a trend: by utilizing GPUs (specifically ray tracing cores on NVIDIA RTX cards), one can substantially reduce simulation times for radar scattering. Our simulator builds on this trend, focusing on large-scale data generation and integration with machine learning workflows, and is designed to be flexible and research-oriented, allowing users to set up new scenes, adjust parameters, and stream out simulation data for ML pipelines.

Willis et al. (2020) describe an approach for solving Maxwell's equations using an SBR technique, leveraging a combination of geometric optics and physical optics to quickly simulate the 3D electromagnetic fields resulting from RF wave propagation in contexts where the simulation volume size is electrically large. This technology performs simulation using NVIDIA's OptiX library to accelerate the computationally expensive ray tracing calculations required by this approach. The computational performance of this approach provides orders of magnitude speed increases over CPU simulation and even more when using RTX graphics cards for simulation which can further accelerate the GPU calculations by a factor of 10 using NVIDIA's ray tracing specific RTX hardware. The new structure, processing framework and calculations required by the OptiX ray tracing libraries are described to solve radar simulation problems at unprecedented speeds due to the use of massively parallel GPU computation devices.

METHODOLOGY

Overview of Simulator Stages

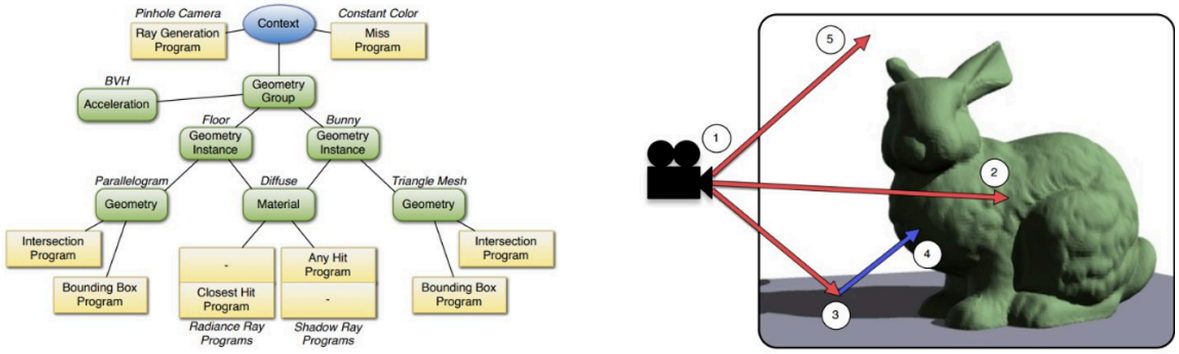
The simulator operates in two primary stages, identification of discrete scattering events using GPU accelerated ray tracing, and backscatter synthesis by numerically integrating the Stratton Chu integral. In **Stage 1**, the simulator launches a dense set of rays from the radar transmitter (or radar receiver for monostatic cases). Rays propagate through a scene composed of complex meshes and interact with objects according to material properties and scattering models. A spherical ray launch distributes rays over a hemisphere or follows an antenna beam pattern to cover potential scatterers. Each ray surface intersection yields the point, surface properties, and energy contribution (direction, phase, amplitude and polarization data). Reflected or transmitted rays continue until a termination criterion is met, resulting in a sparse set of scattering centers representing induced surface currents. In **Stage 2**, the simulator computes the radar return by assigning equivalent currents to each scatterer, computing radiated fields

toward the radar, and summing complex amplitudes using GPU-accelerated vector math. This integration supports arbitrary radar configurations, pulse shapes & antenna patterns, accounts for coherent interference, and can be repeated for each look angle or time step to generate sequences of radar returns or SAR images.

Ray Tracing Using NVIDIA OptiX

We employ NVIDIA's OptiX ray tracing engine to trace millions of rays through the scene on RTX GPUs, leveraging hardware acceleration for rapid intersection tests. Our simulator loads terrain and target models as triangular meshes and builds a bounding volume hierarchy to enable fast ray-geometry queries. An OptiX ray generation program launches rays from the radar while a miss program handles rays that leave the scene. Upon each intersection, a 'closest hit' program computes the local scattering contribution, determines the reflection coefficient based on material, angle of incidence, and polarization, and spawns secondary rays with attenuated power.

We record each hit's position, surface normal, and incident field amplitude and phase, then compute the equivalent current at that point on the fly in the shader. Hits are stored in a GPU buffer. We include special handling for second and third-bounce reflections to capture contributions from corner reflectors and dihedral structures. To improve accuracy on curved surfaces, we apply a normal vector correction by interpolating normals from smoothing groups. By harnessing thousands of GPU threads, the ray tracing stage performs millions of intersection tests per second and scales linearly as more rays or bounces are added. We also parallelize over frequency points and viewing angles by launching separate ray batches, fully exploiting GPU concurrency.



(a) Algorithm (b) Illustrated example

Figure 1. OptiX ray tracing of an example scene with a pinhole camera, two objects and shadows (a) algorithm (b) illustrated example.

Figure 1 shows how rays are traced into a 3D scene by the OptiX engine. It depicts the five categories of rays that OptiX uses to allow massively parallel ray tracing, OptiX allows users to design small programs that can be associated with each ray category which are invoked as required to generate a ray tracing result. Provided the example in Figure 1, the OptiX engine operates in the following fashion:

1. A ***ray generation*** program is a starting point for execution. This program creates and traces rays into the 3D scene. Each ray initiates a Bounding Volume Hierarchy (BVH) traversal to determine how the ray interacts with geometries of the context.
2. A '***closest hit'*** program will be called for those rays that intersect scene geometries. OptiX invokes the program and passes it the 3D intersection location and the local surface geometry which can be used to modify the ray tracing result or emit new rays.
3. An '***any hit'*** program will be called when a ray intersects the scene multiple times which is useful for the computation of surface shadows and surfaces having transmissive material properties, e.g. glass simulation.
4. A ***miss*** program will be called for those rays that do not intersect any scene geometries.
5. An ***exception*** program will be called when unexpected ray-tracing circumstances are detected by the OptiX engine.

The OptiX library integrates these programs to a structure purposefully designed to minimize dependency between traced rays which, by design, produce high-performance ray tracing results for potentially complex phenomena.

Performance gains are reaped via massively parallel computation of programs that run asynchronously and within ultra-fast GPU local memory to perform local atomic contributions to the final result.

Backscatter Integration from Stratton-Chu Equation

The implementation of SAR simulation adopts the NVIDIA OptiX library to compute the solution to the geometric optics problem. The complete simulation problem is broken into (5) main parts:

1. Load the virtual 3D world into an OptiX context and send it to the GPU compute device(s).
2. Initialize models of the vehicle trajectory, pulse signal waveform and radiation characteristics of the SAR antenna.
3. Simulate the emission of SAR EM pulse and sensed backscatter by the receiving antenna.
4. Advance the simulation to the next timestep adjusting moving object positions, velocities and accelerations.
5. Return to step 3 until the backscatter from all pulses has been calculated; then exit.

The scene contains all the models used to generate a simulated environment. This consists of all the objects and their basic material properties. The scene also contains a description of the radar system which characterizes the antenna and radar platform. An observation view is also established to simultaneously monitor the radar and scene simulations. There are a total of (8) different YAML objects for the simulator: the ”mesh” and ”primitive” YAML objects specify object geometry and ”material” YAML objects specify reflectance properties. The ”light” and ”camera” illuminate and visualize the virtual 3D scene. Finally the “antenna”, “SAR system”, and “trajectory” YAML objects specify the antenna, signal processing system and vehicle trajectory and aperture parameters respectively. SAR antenna is specified by describing the operating bandwidth and the mode, i.e. spotlight and stripmap.

Once the ray tracing stage finishes for a given radar view, we obtain a large set of scattering contributions that must be summed to compute the overall backscatter. This process numerically evaluates the Stratton–Chu surface integral. On the GPU, we perform a parallel reduction over all the hit points, where each hit contributes a complex phasor proportional to the local scattering amplitude, multiplied by an exponential term that accounts for phase delay and divided by distance to model spreading loss. By dividing hits among thread blocks, partial sums are accumulated on the GPU, avoiding costly data transfers to the CPU.

The final complex value represents the received electric field, from which the RCS is derived in the far field. We also support generating full-range profiles and SAR images by simulating pulses across frequency sweeps and applying an inverse fast Fourier transform. Multiple frequencies are processed in parallel on the GPU, and coherent processing can occur on either a GPU or CPU. Because rays and field calculations remain entirely on the GPU, data transfer overhead is minimal. The resulting scattered field or power is then sent to the CPU or directly fed into an ML pipeline if training is performed concurrently. In summary, our method couples fast GPU ray tracing with parallel field integration to implement SBR and physical optics efficiently on modern hardware.

RESULTS

Validation of SBR Accuracy

The simulator’s accuracy was validated by comparing its RCS outputs to reference solutions from XPatch, Feko RL GO, Ansys HFSS SBR+, and high-resolution MoM from Altair Feko. Test cases included canonical geometries such as spheres (Mie series reference), flat plates (analytical PO solution), trihedral and dihedral reflectors, and an extended two-sphere scene mimicking a multi-target scenario. For the dihedral corner reflector, the monostatic RCS versus aspect agreed with benchmarks within 0.5 dB across all angles, capturing the strong double bounce peak. For spheres and flat plates errors remained within a few percent while constructive interference lobes and nulls were reproduced accurately.

We conducted an exhaustive analysis to validate the accuracy of our SBR solver, comparing computed RCS magnitudes and phases with those computed under identical situations using Ansys HFSS SBR+ and several other solvers. We compared our SBR solver with Ansys’ for a variety of complex objects, including three-plane geometries such as the trihedrals shown in the top panel of Figure 2. Also shown here is the RCS phase computed

for these trihedral geometries under 1-bounce and 3-bounce instances. We note that, to accurately capture RCS for trihedral geometries, the ray tracing calculation must consider up to 3 bounces. In the middle and bottom panels of Figure 2, we show the RCS phase computed at a fixed elevation over a full 360° azimuthal sweep. Here, the red and blue curves indicate RCS computed by Ansys and our SBR solver, respectively. When we account for the 3-bounce instances, perfect agreement is realized between these two EM solvers, validating the accuracy of our software against leading commercial software.

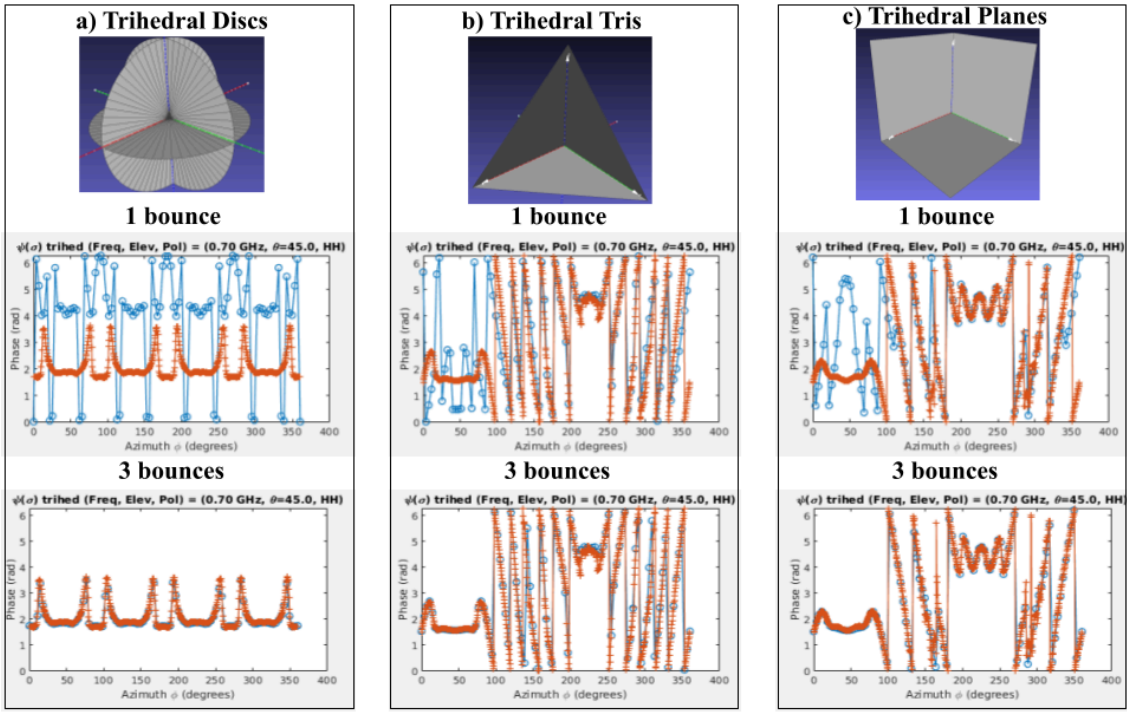


Figure 2. Multi-bounce validation results. RCS phase only for trihedral discs, tris, and planes. Our SBR solver is shown in blue, and HFSS SBR+ is shown in red/orange.

Large-Scale Scene Simulation and Performance

We constructed a complex scene with a high‑resolution terrain mesh and a detailed vehicle model, then simulated an orbiting AC‑130 platform equipped with an X‑band monostatic radar. As the aircraft circled at a few kilometers range, rays were launched downward at a grazing angle, tracing through roughly ten million rays per look and interacting multiple times with terrain and vehicle surfaces. The ray tracing engine processed over three million triangles and recorded about twelve thousand valid multi‑bounce scattering points per pulse. On an NVIDIA RTX 3090 Ti, each look required around 1.3 seconds; on an RTX 4090 Ti, about 0.8 seconds; and on an RTX 4000 Ada, approximately 1.0 second.

These timings include both ray tracing and field integration for roughly one hundred thousand hit points per view. Output backscatter data were processed into high‑resolution SAR images, clearly revealing the vehicle against terrain clutter. Memory usage remained around eight to ten gigabytes, well below available GPU capacity. Near‑linear scaling was observed when doubling ray density. Generating a full 360° dataset with one thousand look angles takes under fifteen minutes on a single RTX 4090 Ti, demonstrating that GPU acceleration enables large‑scale radar simulations and rapid prototyping of signal processing algorithms.

As an example of our current capabilities, Figure 3 illustrates a snapshot of our simulation environment (region around San Carlos Airport near San Francisco) as visualized in Gazebo. The simulation includes a square patch antenna mounted to an AC130 (indicated by green arrow) orbiting a mobile scud launcher (indicated by red arrow) in a circular spotlight SAR scenario. The box contained within the red rectangle displays the antenna's field of view. The plot on the bottom left are computed backscatter magnitudes resulting from reflected RF energy in the scene as

a function of range bin (top left is backscatter response after the data is sent and unpacked by an emulated radar system – e.g. RFSoC FPGA).

In this snapshot, a building comes into view of the antenna, and there is a large response at smaller range bins due to the presence of the building, while the smaller response at farther range bins represents the response from the scud launcher. We have since refined our 3D model generation methods which improve simulation geometric accuracy.

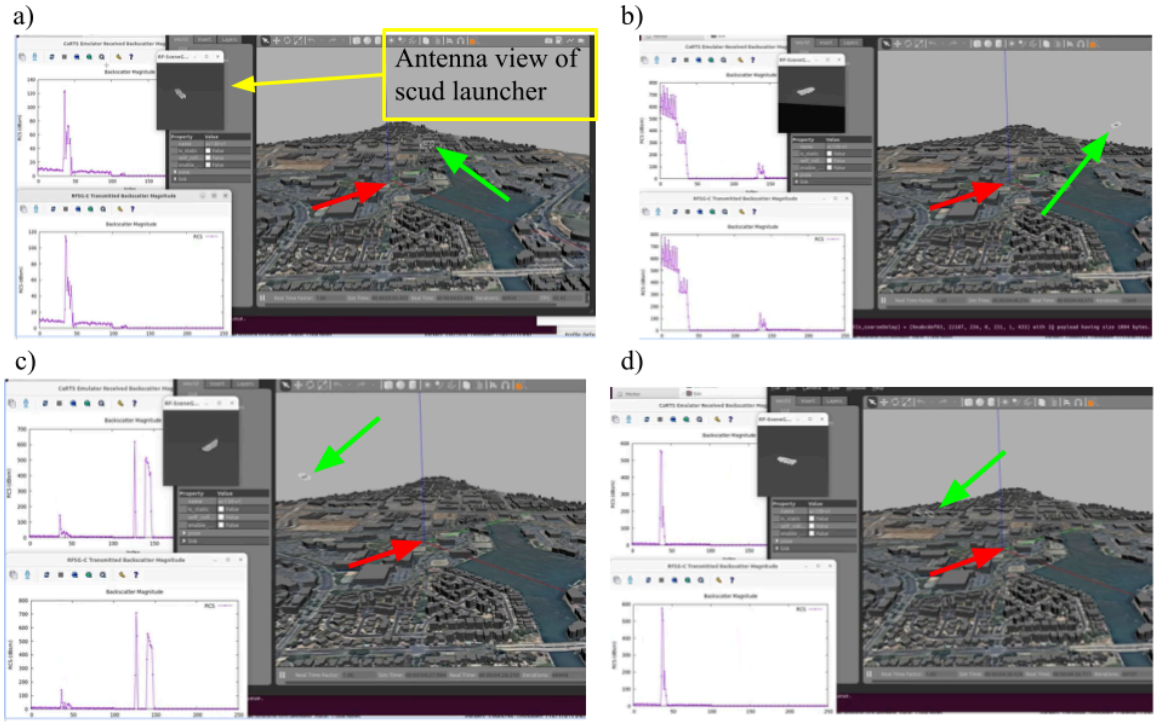


Figure 3 shows n4 snapshots from the end-to-end simulation environment within the emulated HWIL setup. The green arrow indicates the position of the antenna (C130 airplane). The red arrow indicates the position of the scud launcher.

Table 1 shows a latency experiment that measures latency for two algorithms: (1) backscatter computation and minimum-distance computation. Each algorithm is run for scenes with sparse (250k) and dense (1M) scatterers. Results are compiled across three different NVIDIA RTX GPUs using two operational modes: (1) with rendered RGB visualizations and (2) compute-only (no RGB). Table 1 latencies are reported in microseconds for each algorithm, highlighting the impact of scene resolution and rendering overhead on overall latency. Despite its lower hardware specs, the RTX 4000 Ada demonstrates surprisingly competitive performance in both modes. This confirms that our benchmarking framework and software updates (CUDA 12.8, OptiX 9.0, updated drivers) produce reliable, reproducible results across different hardware generations.

As shown in Table 1, the single-slot RTX 4000 Ada completes full resolution backscatter in 4 ms, compared with 2.89 ms on the triple-slot RTX 4090 Ti (Table 12.1). Its minimum-distance calculations are similarly close to the 4090 Ti, even though the 4090 Ti boasts roughly three times as many CUDA cores. This performance density underscores our emphasis on compute density over raw per-GPU power: three RTX 4000 Adas can fit where one RTX 4090 Ti would, potentially matching or exceeding its absolute throughput within the same chassis footprint.

When normalizing performance against hardware capabilities, a single RTX 4000 Ada achieves about 76% of the RTX 4090 Ti's compute-only throughput, even though the 4000 Ada has only 48 RT cores while the RTX 4090 Ti has 142 (Table 1). RT cores are the primary metric that determines latency for ray tracing calculations. Extrapolating linearly, three 4000 Adas could deliver roughly 228% of a single 4090 Ti's performance, translating worst-case framerates of 258 Hz for 1 M scatterers versus 113 Hz on the 4090 Ti; the worst-case framerate for the RTX 4090 Ti during the on-site demo was 113 Hz for 1M scatterers. If we consider average-case performance (≈250 Hz for the

4090 Ti), three Adas might even approach 570 Hz, more than doubling the minimum guaranteed rate. This analysis suggests that simulation systems consisting of densely packed RTX 4000 Ada generation cards promise to outperform simulations consisting of RTX 4090 Ti cards due to their size difference where RTX 4000 solutions have a higher resulting ray tracing compute density..

Table 1 shows compute latency (all values in µs) for backscatter and minimum‑distance calculations on 250 k and 1 M scatterers, with and without rendered RGB, across selected RTX GPUs. Compute times corresponding to no RGB with 1M scatterers for both the 4090 Ti and 4000 Ada are highlighted in red for easy comparison.

Latency Study	RGB		No RGB	
# of scatterers	**Backscatter**	**Min. Distance**	**Backscatter**	**Min. Distance**
NVIDIA RTX 4090 Ti				
1,000,000	15,400	926	2,890	804
250,000	15,200	420	1,083	546
NVIDIA RTX 4000 Ada (new)				
1,000,000	15,800	850	4,000	850
250,000	16,070	411	1,270	625
NVIDIA RTX 3090 Ti				
1,000,000	14,700	1,916	4,400	1,776
250,000	15,200	420	1,356	600
NVIDIA GeForce RTX 4060 Laptop GPU				
1,000,000	9,827	1,721	8,005	1,988
250,000	3,520	646	4,400	1,184

Distributed Compute and “Lookahead” Capabilities

Simulation software was first optimized to fully utilize individual RTX GPU cards with current OptiX and CUDA libraries (9.0.0 and 12.2, respectively). From there, the software stack is packaged into a lightweight container and deployed en masse to a network of worker nodes across an RDMA over converged ethernet (RoCE) fabric. At boot, the driver layer initializes the link layer and spawns threads for completion queues, work-request queues, and an asynchronous message dispatcher. Once connected to the central controller, each worker announces its available GPU resources and network stats back to the controller over RDMA.

The controller program maintains two key roles: real-time job scheduling and a “lookahead” prediction module. With all kinematic state variables of the radar-equipped aircraft known, the controller interpolates future poses along the flight path. These predicted states are encapsulated in a “Job Message” and sent via RDMA to workers whose GPUs are forecasted to idle at the corresponding simulation times. Upon receipt of a config message (carrying a JSON system setup and OBJ scene file), a worker thread binds the data to the SBR solver and begins pre-computing the backscatter response.

By pre-staging responses at future waypoints along the vehicle’s trajectory, the latency of the overall backscatter during the simulation is reduced: as the aircraft arrives at the predicted location, the backscatter data is already computed and immediately available. This three-layer RDMA architecture, link, driver, and application, ensures ultra-low overhead transfers and tightly synchronized dispatch, allowing massive, distributed GPU clusters to simulate complex 3D scenes in real-time. The network architecture easily scales to provide the user with the desired latency. Moreover, the architecture is agnostic to the type of available GPU hardware; all GPUs in the network need not be the same. During the initialization phase, latency statistics are collected for each network GPU to construct an efficient scheduling routine that distributes the backscatter calculation.

RDMA’s ability to bypass the OS kernel and leverage zero-copy transfers drives end-to-end latencies down into the single-digit microsecond range, often 5-10x lower than traditional TCP/IP sockets. By mapping remote memory directly into the local address space, RDMA eliminates extra buffer copies and interrupts, delivering highly deterministic delivery times with minimal jitter (this is essential toward constructing an efficient scheduling routine).

Its hardware-offloaded flow control and congestion management also ensure stable, line-rate throughput even under heavy loads, making it ideal for coordinating fine-grained GPU workloads across hundreds of nodes without the unpredictable delays one might expect over Ethernet or TCP-based fabrics.

CONCLUSION

This work introduces a GPU-accelerated RF scene backscatter simulator that merges the geometric scalability of the SBR method with the physical accuracy of the Stratton–Chu integral, all implemented on NVIDIA's OptiX ray tracing engine. By leveraging RTX GPU hardware for both ray tracing and field integration, the simulator achieves one to two orders of magnitude speedups over traditional solvers while maintaining high fidelity. It can process electrically large, complex scenes—including terrain, multiple targets, and multi-bounce interactions—and generate realistic radar returns suitable for tasks such as automatic target recognition, sensor fusion, and SAR imaging.

Validation against canonical targets and comparisons with state-of-the-art full-wave and asymptotic tools confirm that GPU acceleration does not compromise accuracy. The framework produces high-quality synthetic datasets for machine learning and serves as a testbed for radar algorithm development, significantly reducing computation time compared to CPU-bound solutions. Future work will extend the simulator to dynamic scenes with moving targets, support multi-sensor platforms, and incorporate hybrid learning-based enhancements for scattering prediction in unresolved geometries, further enhancing its flexibility and research utility.

Benefits & Capabilities

The GPU-accelerated SBR simulator offers a unique combination of performance and flexibility. It enables researchers to rapidly explore scenarios; for example, evaluating how a stealth coating on a vehicle changes its radar signature, or how different flight trajectories of a radar platform affect imaging. Because of the speed, one can iterate over many designs or conditions in a short time. Moreover, the system is inherently scalable: it can be deployed on multi-GPU clusters to further increase throughput, or scaled down to run on a single gaming laptop GPU for smaller scenes. The use of OptiX and RTX technology ensures that it will continue to benefit from hardware advancements in GPU ray tracing. The simulator's integration of ray tracing and field integration on the GPU also positions it well for real-time applications in the future, such as HWIL simulations where a radar system's software might be tested with simulated live returns.

Future Enhancements

Extensions include adding diffraction modeling via PTD or UTD to improve accuracy for sharp edges or grazing angles, and supporting polarimetric effects with materials beyond perfect conductors (such as dielectrics and radar absorbing materials), possibly by hybridizing with volumetric methods for penetrable media. Computationally, adaptive ray refinement could focus rays on regions that contribute most to backscatter, enhancing fidelity without a linear cost increase. Integrating neural networks to learn surrogate scattering models or compress output data offers a promising intersection of ML and physics-based simulation. We also foresee a pipeline for automatic dataset labeling and augmentation, where 3D model libraries generate labeled radar images at scale for deep learning. The GPU-accelerated simulator thus provides a powerful platform for research and development in radar sensing, lowering the barrier to high-quality synthetic data and enabling data-driven approaches in radar signal processing and automatic target recognition.

REFERENCES

1. Meng, Huan-Ting. (2011). Acceleration of asymptotic computational electromagnetics physical optics — shooting and bouncing ray (PO-SBR) method using CUDA. Masters thesis, University of Illinois at Urbana-Champaign.
2. A. R. Willis, M. S. Hossain and J. Godwin. (2020). Hardware-accelerated SAR simulation with NVIDIA-RTX technology. In Algorithms for Synthetic Aperture Radar Imagery XXVII (Vol. 11393, p. 113930O). International Society for Optics and Photonics.
3. Xu G, Dong C, Zhao T, Yin H, Chen X. (2021). Acceleration of shooting and bouncing ray method based on OptiX and normal vectors correction. PLoS ONE 16(6): e0253743. https://doi.org/10.1371/journal.pone.0253743