Fast and Scalable LiDAR Data Generation for Autonomous Driving Simulation without Raycasting

Irfan Nafiz Shahan[1,*], Al-Mubin Khan Nabil[2,*], and Arpan Kusari[3,†]

[1] Shahjalal University of Science and Technology, Sylhet 3114, Bangladesh
irfan-eee@sust.edu,
[2] Independent Researcher, Bangladesh
almubinnabil@gmail.com,
[3] University of Michigan, Ann Arbor, MI 48109, USA
kusari@umich.edu

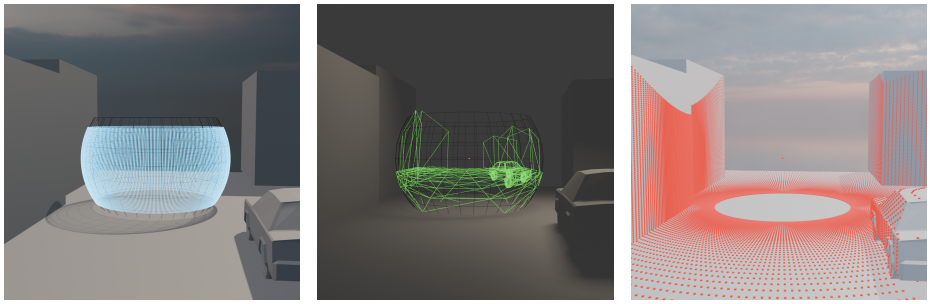
Abstract. Autonomous driving relies on LiDAR simulation for generating realistic 3D point clouds, typically achieved using computationally expensive raycasting. We introduce **GnoSphere360**, a scalable LiDAR data generation method that eliminates raycasting by projecting triangular meshes onto a unit sphere and interpolating point samples with gnomonic projection. This computational approach bypasses traditional ray-mesh intersection, significantly reducing computation time while maintaining compatibility for further downstream physics-based modeling. Benchmarking against NVIDIA OptiX and native CARLA engines, GnoSphere360 achieves up to 35× speedup over CARLA for physical LiDAR models and maintains robust performance against OptiX, while preserving similar LiDAR point cloud output. However, for extremely dense LiDAR configurations, GnoSphere360 achieves over 3.6x speedup against OptiX, showcasing the scalability of our proposed method. Ablation studies demonstrate robustness of GnoSphere360 across varying scenes and sensor models, while preserving data fidelity. Our results show GnoSphere360 is an efficient, accurate, and parallelizable alternative to raycasting and rasterization for autonomous driving simulation. Project website: gnosphere360.github.io

Keywords: LiDAR simulation · Raycasting · Gnomonic projection

1 Introduction

Automated driving systems (ADS) rely on accurate environmental sensing to navigate safely, requiring detection of both static features, like roadways and traffic signals, and dynamic obstacles, such as pedestrians and vehicles. 3D LiDAR (Light Detection and Ranging) has become a pivotal active remote sensing tool for ADS [11], capable of producing highly detailed environmental maps with

* These authors contributed equally to this work.
† Corresponding author



(a) Ray Pre-computation **(b)** Spherical Projection **(c)** LiDAR points generated

Fig. 1: LiDAR data generation for triangular mesh using proposed GnoSphere360 method

sub-centimeter accuracy. This allows ADS to precisely locate objects, recognize road edges, and identify obstacles, with sensor configurations offering varying resolution and fields of view.

Simulation is essential for developing and testing ADS pipelines [9], permitting recreation of complex and rare scenarios to evaluate system robustness. Platforms like CARLA [4] and AirSim [15] offer realistic models with a wide array of static and dynamic assets, enabling researchers to position sensors and extract synthetic data under controlled conditions. These simulation environments help validate perception algorithms by replicating real-world sensor outputs and behaviors.

LiDAR data generation in these simulators commonly relies on raycasting [14], a computational method which traces laser pulses as straight lines to detect intersections with objects in the virtual scene. Raycasting produces accurate 3D point clouds that mimic actual sensor geometry, but its realism comes at a steep computational cost. Calculating millions of ray-object interactions demands significant processing power and memory, presenting challenges for real-time or large-scale simulations.

To address these performance bottlenecks, rasterization-based methods have been adopted [10], leveraging GPUs' native rendering pipelines to simulate LiDAR data with greater speed. By efficiently processing depth maps and parallelizing hit computation, rasterization achieves a substantial reduction in computation time compared to raycasting. However, while rasterization accelerates simulation, it may sacrifice some accuracy. Additionally, neural networks have emerged for LiDAR data generation [18,19], learning sensor behavior from training datasets. Despite their promise, these models can be resource-intensive and may produce anomalies or unrealistic outputs if exposed to novel scenarios, prompting a preference for physics-based methods.

Our contribution critically examines raycasting's role in LiDAR simulation and introduces **GnoSphere360**, a novel approach that eliminates the need for raycasting on triangular meshes. Instead, we project the mesh onto a unit sphere centered on the LiDAR sensor, compute azimuth-elevation coordinates for each

triangle, and cull triangles based on distance, elevation, and visibility. Candidate rays are assigned based on this spherical projection, and using gnomonic projection, we interpolate point samples directly on the mesh—without traditional raycasting. Fig. 1 showcases the output of our method for a sample triangular mesh. Our method is fast, parallelizable, and scalable, representing the first alternative to raycasting, rasterization, and generative models in LiDAR data simulation.

Our paper is structured as follows: Section 2 provides some related work, Section 3 details the proposed method, Section 4 provides the results of the proposed method along with the comparison to other approaches and finally, Section 5 provides the conclusion.

2 Related Work

Raycasting has long been a central technique in computer graphics for tracing light rays to generate realistic imagery [6], with advancements like the Möller Trumbore [12] algorithm providing efficient ray/triangle intersection solutions widely used in gaming engines. Rotating LiDAR sensors, equipped with multiple vertically oriented lasers, utilize similar principles, emitting pulses to measure depth. As a consequence, simulating LiDAR devices with raycasting-based methods is straightforward, and many LiDAR simulators leverage computer graphics algorithms and hardware for this purpose. Helios [1] was one of the first general-purpose time-of-flight simulators based on raycasting, while open-source ADS platforms such as CARLA [4] and AirSim [15] rely on raycasting to model rotating LiDAR, generating dense point clouds through rays projected from the sensor to intersect virtual meshes.

Research efforts have focused on accelerating these simulations through both software and hardware optimizations. NVIDIA's OptiX raytracing engine, used within Vires' VTD driving simulator [8], delivers high-speed intersection processing on GPUs, and frameworks like optScan offer modular guidelines for parallel raycasting [7]. To further improve rendering efficiency, rasterization-based approaches have emerged. For example, AADS introduced rasterization via cube-map projection [10], blending real backgrounds with synthetic actors, while other methods render only LiDAR-visible portions of the scene, increasing computational speed and frequency [3, 5].

Recently, deep learning models have tackled LiDAR data generation by learning the physical raycasting process. LiDARGAN and LiDARVAE convert 3D grids into 2D point representations [2], enabling generative adversarial networks and variational autoencoders to produce synthetic LiDAR point grids, which can be unrolled into point clouds during post-processing. While primarily used for realistic data completion, generating LiDAR data from triangular mesh structures in simulation remains relatively unexplored. Learning-based algorithms, such as LiDARGen [19], map traffic layouts from bird's eye views to point cloud sequences; however, these approaches often struggle with out-of-distribution scenarios, limiting their ability to faithfully replicate simulated environments.

3 Methodology

To address the problems of computationally expensive raycast operations for modeling LiDARs in simulations, we propose our solution, **GnoSphere360**. The goal is to come up with an alternative to LiDAR modelling via raycasting that is fast, produces accurate LiDAR points, while being computationally competitive to state-of-the-art raycasting. Our method is able to take the meshes of a simulated world, and LiDAR position to generate point clouds without any ray traversal throughout the simulation environment.

When ray-tracing (RT) is used for traditional rendering applications, a lot of RT features play a significant role in developing a realistic, physically and visually accurate scene. Rendering details such as reflections, shadows, ray bounces, etc. require separate shaders for each ray which traverses a Bounding Volume Hierarchy (BVH) to reduce the number of ray-triangle intersections. Modern massively-parallel silicon (e.g. NVIDIA RT Cores, AMD Ray Accelerators) makes this feasible in real time by running fixed function hardware units, but these accelerators are not universally available and introduce platform dependencies.

LiDAR modeling demands new requirements that allow algorithmic flexibility. Each beam travels in a known, predictable direction arranged on a regular azimuth-elevation grid, and only the nearest intersection per direction is needed. Rather than asking "*Which triangle does this ray intersect?*", we reframe the problem as "*What does the LiDAR observe in a given spherical direction?*" This perspective shift enables us to pre-transform scene geometry into the sensor's local coordinate frame and build acceleration structures that are tailored to the LiDAR's spherical sampling pattern, as described below.

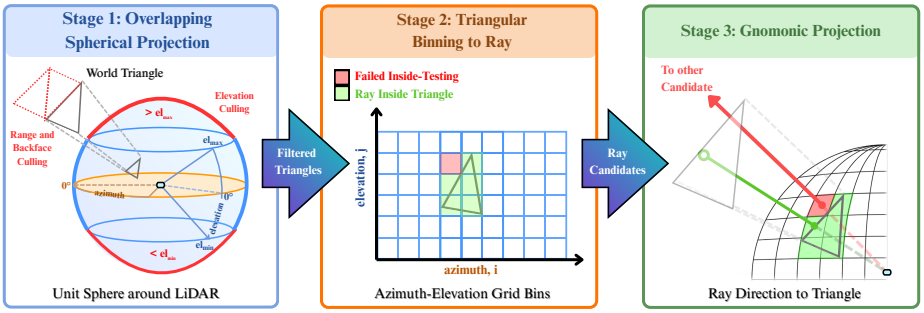


Fig. 2: GnoSphere360 architecture, showcasing main 3 stages of the process.

3.1 GnoSphere360

The pipeline consists of three sequential stages, each designed to minimise memory traffic and arithmetic operations through aggressive precomputation and early culling. Figure 2 illustrates the overall flow.

Stage 1: Overlapping Spherical Projection We can visualize the 3D world projected on a *unit sphere* around the LiDAR source, thus bringing the meshes to the sensor. Since any two triangles at different distances from the sensor can be projected onto the same spherical surface, we allow such projections to happen in an overlapping fashion.

This "mesh-to-unit-sphere" formulation has two immediate advantages. First, a spherical projection preserves collinearity. The projected point $\hat{\mathbf{v}}$ lies on the same ray from the sensor origin as its world-space counterpart $\mathbf{v}$, so no perspective distortion is introduced. Second, the spherical grid directly corresponds to the sensor's azimuth-elevation sampling pattern, making the acceleration structure grid-compatible by construction.

A triangle vertex $v = (x, y, z)^\top$ can be expressed in the LiDAR coordinate frame (origin at the sensor) as the vector $\boldsymbol{v}$. Therefore, its unit-sphere projection is $\hat{\mathbf{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$. The full spherical coordinates are found using the formulae $\theta = \arctan(\frac{x}{z}), \phi = \arcsin(\frac{y}{\|\mathbf{v}\|})$, where $\theta \in (-\pi, \pi]$ is the azimuth and $\phi \in [-\pi/2, \pi/2]$ is the elevation.

Range culling. For each triangle with vertices v_0, v_1, v_2 (already in LiDAR space), the vector to first vertex $\boldsymbol{v}_0$ is used to perform a rudimentary range culling. This is an elementary comparison test after our spherical transform points are precomputed. The vector $\boldsymbol{v}_0$ is chosen instead of the centroid of the triangle because for triangles at typical LiDAR range limits of $>$ 100m ranges have centroid vectors which are not very different from a vertex due to their distance.

Backface culling. We compute the geometric face normal,

$$\boldsymbol{n}_g = \frac{(\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0)}{\|(\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0)\|}. \tag{1}$$

We align $\boldsymbol{n}_g$ with the mesh's stored normal to handle winding-order inconsistencies. A triangle is culled when the vector, $\boldsymbol{v}_0$, satisfies $\boldsymbol{n}_g \cdot \boldsymbol{v}_0 \geq 0$, which indicates the face points away from the sensor located at the origin. Backface culling removes approximately half of all triangles before any further processing.

Elevation culling. Triangles whose entire azimuth-elevation bounding box lies outside the LiDAR's configured vertical field of view are discarded at this stage, further reducing the candidate set before binning.

Stage 2: Triangle Binning After transformation, each surviving triangle is assigned to a set of *candidate rays* using an azimuth-elevation bounding box computed from its projected vertices. For a triangle projected onto the unit sphere, we compute its azimuth range $[\theta_{\min}, \theta_{\max}]$.

The bounding box is quantized into ray-grid indices $i = \left\lfloor \frac{\theta - \theta_0}{\Delta\theta} \right\rfloor, j = \left\lfloor \frac{\phi - \phi_0}{\Delta\phi} \right\rfloor$, where $\Delta\theta$ and $\Delta\phi$ are the azimuth and elevation resolutions of the LiDAR model, and (θ_0, ϕ_0) are the minimum angles of the grid.

For each candidate ray (i, j) within the bounding box, we verify that the ray direction $\boldsymbol{r}$ lies inside the spherical triangle using precomputed edge normals $\{\boldsymbol{e}_k\}_{k=0}^{2}$. A ray is confirmed as a candidate if

$$\boldsymbol{e}_0 \cdot \boldsymbol{r} \;\geq\; 0 \;\;\wedge\;\; \boldsymbol{e}_1 \cdot \boldsymbol{r} \;\geq\; 0 \;\;\wedge\;\; \boldsymbol{e}_2 \cdot \boldsymbol{r} \;\geq\; 0, \tag{2}$$

or the analogous all ≤ 0 condition. This sign test is significantly cheaper than barycentric evaluation and is fully enabled by the edge-normal precomputation carried out before the transformation stage.

Stage 3: Gnomonic Ray Projection The final stage resolves the actual intersection distance for each active ray using *gnomonic projection*, which maps any collinear point along a ray direction onto an arbitrary plane in 3D space. This is valid precisely because Stage 1 preserves collinearity.

Let the supporting plane of a triangle be defined by its normal $\mathbf{n}$ and an arbitrary vertex $\boldsymbol{v}_0$ on the plane:

$$\hat{\mathbf{n}} \cdot \boldsymbol{P} + d = 0, \quad d = -\hat{\mathbf{n}} \cdot \boldsymbol{v}_0, \tag{3}$$

where d represents the perpendicular offset of the plane from the 3D origin, O. Geometrically it represents the projection of the vector $\boldsymbol{v}_0$ to the direction of $\hat{\mathbf{n}}$, and hence represents the shortest possible offset to the plane.

Since the LiDAR ray direction $\boldsymbol{S}$ (a point on the unit sphere) is collinear with the intersection point $\boldsymbol{P} = t\boldsymbol{S}$, substituting into Eq. (3) yields the scalar depth

$$\boxed{t = \frac{\hat{\mathbf{n}} \cdot \boldsymbol{v}_0}{\hat{\mathbf{n}} \cdot \boldsymbol{S}}}. \tag{4}$$

All quantities except $\boldsymbol{v}_0$ can be precomputed once per triangle, so each ray-triangle depth evaluation requires only two dot products and one division—a total of 3 subtractions, 3 multiplications, 2 additions, and 1 division. A geometric diagram is included for reference in Figure 3.

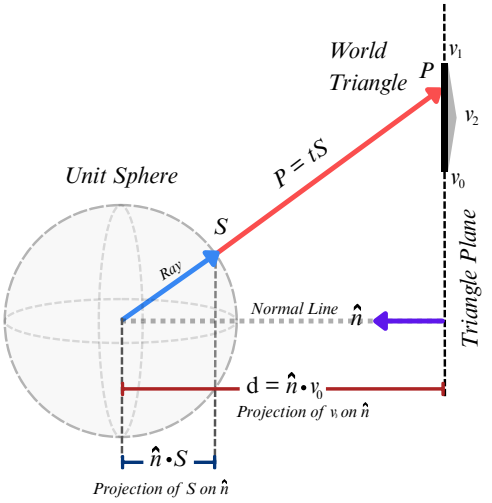


Fig. 3: Gnomonic Projection Isometric Diagram.

Key property. Because the binning stage already guarantees that the ray direction $\boldsymbol{r}$ lies within the spherical triangle (Eq. (2)), and because gnomonic projection preserves the interior of the triangle under this mapping, the intersection point $\boldsymbol{P} = t\boldsymbol{S}$ is *guaranteed to lie inside the triangle in world space.* This eliminates the

barycentric inside-test that dominates the cost of the classic Möller–Trumbore algorithm [12].

Proxy-Occlusion culling. We exploit two additional optimizations that arise from the spherical-projection structure to perform proxy-occlusion culling.

Best-first initialization. Each ray loads its minimum-d candidate from the binning stage and tests it first, establishing a tight upper bound t_{best} on the intersection distance.

Depth-based skipping. For subsequent candidates, we check whether the triangle's plane offset d satisfies $d \geq t_{\text{best}}$. Since $t = -d \,/\, (\hat{\mathbf{n}} \cdot \boldsymbol{S})$ and $-\hat{\mathbf{n}} \cdot \boldsymbol{S} > 0$ for front-facing triangles with $-\hat{\mathbf{n}} \cdot \boldsymbol{S} \leq 1$, we have $t \geq d$. Therefore $d \geq t_{\text{best}}$ implies $t \geq t_{\text{best}}$, enabling early rejection *without* computing the ray-plane intersection or loading the triangle normal.

Compatibility with physics-based parameters. GnoSphere360 replaces only the ray-triangle intersection stage; downstream physical modeling is therefore unaffected. Because triangle-point associations are preserved at each intersection, standard physical attributes such as material-based intensity/reflectance, range-based drop-off, and probabilistic or Gaussian noise remain directly applicable using the same models as conventional raycasting pipelines.

Table 1: Floating-point operations per ray-triangle intersection algorithm. Precomputed quantities are excluded from the per-invocation counts.

Stage / Algorithm	**Scope**	**Sub**	**Mul**	**Add**	**Div**
GnoSphere360					
Backface cull (Stage 1)	per triangle	3	3	2	0
Inside test (Stage 2)	per triangle	0	9	6	0
Gnomonic depth (Stage 3)	per ray	3	3	2	1
Möller–Trumbore					
MT with culling and inside-test	per ray	9	24	8	1
Difference					
Worst-case (no amortization)	per compute	3	9	-2	0

3.2 Computational Complexity

Table 1 compares the per-ray-triangle floating-point operations (FLOPs) of the standard Möller–Trumbore (MT) algorithm against GnoSphere360's gnomonic projection kernel, after all possible precomputations have been applied. Importantly, MT performs backface culling and the barycentric inside-test for each ray, whereas GnoSphere360 distributes these operations across earlier pipeline stages (transformation and binning, respectively) where they can be amortized over all rays that share the same triangle. The table therefore also reports the per-triangle cost of each stage so that the total operation count can be compared fairly.

Table 2: Physical parameters of LiDAR models recreated in simulation. The 256-channel model is a synthetic configuration for ablation testing.

Model	Vert. FOV (°)	H. Res.	Channels	Range (m)	Rays
VLP-16	$-15.0 \sim +15.0$	1800	16	100.0	28,800
VLP-32	$-25.0 \sim +15.0$	3600	32	100.0	115,200
HDL-64E (Baseline)	$-25.0 \sim +2.0$	4500	64	120.0	288,000
Hypothetical-256	$-45.0 \sim +45.0$	18000	256	200.0	4,608,000
Hypothetical-350	$-45.0 \sim +45.0$	18000	350	200.0	6,300,000

3.3 Implementation

The GnoSphere360 pipeline is massively parallelizable, and therefore the pipeline is implemented in CUDA. This serves as a means to compare fairly with the CUDA-based OptiX, and does not constrain the pipeline to only CUDA based hardware architectures. The precomputed data that can be dynamically calculated when necessary such as LiDAR directions from any arbitrary physical LiDAR model, triangle mesh data such as edge normals, plane-d, geometric normals, are all stored within a Structure-of-Arrays (SOA) format. This allows for consecutive threads within a GPU warp to load coalesced memory transactions. Therefore this can be utilized within each stage to reduce memory I/O latency, and increase warp efficiency.

The potential for parallelism can be exploited by mapping the thread granularity to the scope of each stage of the pipeline. The Stages 1 and 2 are both triangle-parallel by assigning a thread per triangle being tested - performing spherical projection, testing for cull conditions and binning. Running the final stage triangle-parallel would cause massive warp divergence, as different triangles can be tested for the same LiDAR direction, and lead to atomic contention as a thread waits to test on the same ray. Instead, we flip to a ray-parallel approach for the gnomonic projection CUDA kernel - where one thread per LiDAR direction resolves the smallest final depth using the triangle candidates assigned by Stage 2. Our experiments show that ray-parallel warp divergence is extremely low as 90% of rays have between 1-5 ray candidates. Ray-parallel also allows threads to exit early due to proxy-occlusion.

We evaluate GnoSphere360 against two ray-tracing LiDAR alternatives: the native LiDAR simulator bundled with CARLA and NVIDIA OptiX [13]. CARLA LiDAR sensor serves as the CPU-computed lower bound on ray-tracing performance. OptiX is a hardware-accelerated ray-tracing framework that represents the current state-of-the-art but requires dedicated RT-core hardware tied to the NVIDIA platform. OptiX also uses instance acceleration structure (IAS). IAS refits the bounding volume hierarchy (BVH) generated by OptiX from the previous frame instead of building the BVH for the entire scene again, thus massively improving efficiency. All benchmarks are conducted on an RTX 3070, with AMD Ryzen 5 2600x, 16 gigabytes of DDR4 RAM.

4 Results

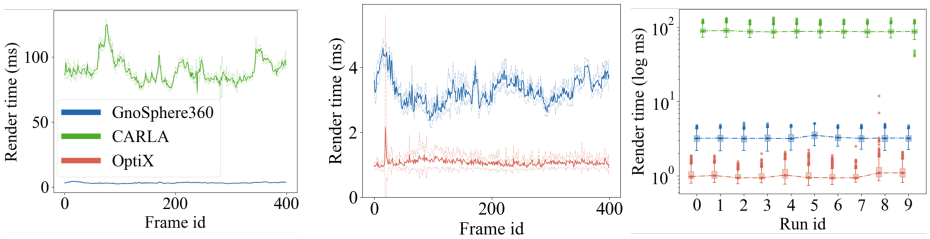
For the baseline LiDAR model, we simulate the 64 channel HDL-64E rotating LiDAR sensor at a resolution of 288K rays. Details for each LiDAR model used can be found in Table 2.

4.1 Benchmarking Setup

Existing simulation software does a lot of background computations and thus can be characterized as black-boxes. This leads to restricted low-level control which limits our ability to do a fair one-to-one benchmark. Therefore, we have built a custom evaluation platform in C++ that uses OpenGL for rendering and debugging. OptiX Wrapper Library (OWL) [17] is used to perform GPU based ray-tracing that uses RT-core acceleration, and CUDA for GPU based GnoSphere360 implementation. The advantage of such a setup involves complete deterministic control, therefore we can perform all benchmarks in no-rendering mode for valid comparison.

A deterministic run was collected for 400 LiDAR frames with CARLA v0.9.15 based on Unreal Engine 4.26 (UE4) for benchmarking. The LiDAR sensor location data were collected for the 400 frames, using a fixed pseudorandom seed. We then exported the CARLA town map from UE4, imported it into our evaluation framework and replayed the run for OptiX and GnoSphere360 using the LiDAR sensor location data. This ensured that we have repeatable scenes of 400 frames for CARLA-based towns both in CARLA Simulator, and also within our evaluation framework for GnoSphere360 and OptiX. To increase validity further, the rendering and physics-simulation time from CARLA simulator was also subtracted from each frame. LiDAR point cloud data when collected, were always collected separate from the timing benchmarks to avoid including I/O read-write latency. Static CARLA scenes were used for the benchmark runs to represent the best-case performance of OptiX using IAS refitting, as opposed to scenes containing dynamic changes such as traffic[1]. GnoSphere360 performs computations independently for every frame, resulting in consistent performance regardless of scene dynamics. It is worth noting, since GnoSphere360 does not use recaching, OptiX is at a greater advantage, and there is room for improvement.

We ran the simulation for 11 independent runs to capture run-to-run variance with one warm-up run which is ignored in the results. The baseline comparisons are performed on CARLA Town03, a medium-complexity urban environment, which is also ported to our benchmark program. Cross-scene generalization over other towns is discussed in Section 4.5.



(a) GnoSphere360 vs CARLA **(b)** GnoSphere360 vs OptiX **(c)** Run to run consistency

Fig. 4: Frame time comparisons across different pipelines for baseline LiDAR (a) and (b) show the average frame time for each of the 400 frames across 10 runs for GnoSphere360 vs CARLA and OptiX respectively; (c) shows the consistency of each pipeline as box plots across runs in log time.

Table 3: Frame time statistics for each renderer on the HDL-64E baseline configuration.

		Variance (ms^2)	
Renderer	Mean (ms)	Across Frames	Across Runs
CARLA	89.6170	76.6338	1.1224
GnoSphere360	2.5420	0.0701	**0.0003**
OptiX	**1.2270**	**0.0162**	0.0110

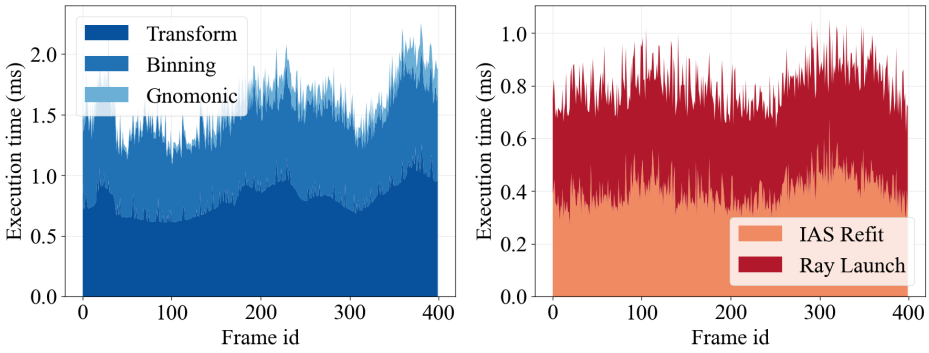
4.2 Comparisons with OptiX Raytracing and CARLA

Baseline LiDAR Comparison We provide per frame comparison over the 10 runs for the different LiDAR generation pipelines for the HDL-64E baseline configuration in Fig. 4. In comparing GnoSphere360 with CARLA in a relatively simpler LiDAR model, we find our proposed approach to be faster by about ~ 35 times, while being much more consistent in performance over frames (Fig. 4a). In contrast, GnoSphere360 has a relatively similar performance to OptiX without any RT-core hardware acceleration (Fig. 4b) with a frame time difference on average of about $\sim 1.3ms$. Fig. 4c shows the run-to-run consistency further showcasing the validity of these findings over multiple runs. The frame time is in log-scale, showing an orders of magnitude difference between CARLA and GnoSphere360 and the close similarity between OptiX and GnoSphere360. OptiX frame time has greater stability over an entire run, while it can be seen to have greater variance between separate runs as quantified by the Table 3, which tabulates the summary statistics of the different generation pipelines. GnoSphere360 can be said to have greater run-to-run consistency overall.

[1] Dylan Lacewell, "Fast Ray Tracing of Dynamic Scenes Using NVIDIA OptiX 9 and NVIDIA RTX Mega Geometry," NVIDIA Technical Blog, April 24, 2025. Available at `https://developer.nvidia.com/blog/fast-ray-tracing-of-dynamic-scenes-using-nvidia-optix-9-and-nvidia-rtx-mega-geometry/`. Accessed: 2026-06-30.

Table 4: Frame time statistics for GnoSphere360 and OptiX on hypothetical stress-test LiDAR configurations. (See Table 2)

LiDAR Model	Renderer	Mean (ms)	Variance (ms^2)	Speedup
Hypothetical-256 (4.6M rays)	OptiX	13.54	7.59	
	GnoSphere360	**5.60**	**1.06**	**x2.42**
Hypothetical-350 (6.3M rays)	OptiX	62.98	157.84	
	GnoSphere360	**17.27**	**39.40**	**x3.65**



(a) GnoSphere360 execution timing per stage **(b)** OptiX execution time per stage

Fig. 5: Comparison of GnoSphere360 and OptiX in terms of GPU timing breakdown.

Hypothetical LiDAR Stress-Test To stress-test the performance of our upper bound OptiX with GnoSphere360, we utilized two hypothetical LiDAR models detailed in Table 2, that generate about 4 million and 6 million rays per frame. A detailed overview of the comparisons is shown in Table 4. What is noteworthy is the performance of GnoSphere360 against OptiX in the extreme examples - OptiX struggles to keep up with the number of rays. In contrast, GnoSphere360 implementation is resilient over a greater number of rays, performing about 2.42 times faster than OptiX at 4 million rays. As the number of rays is increased to 6 million this discrepancy rises to GnoSphere360 being about 3.65 times faster than OptiX. We believe OptiX degrades with ray-scaling due to - (a) register-heavy traversal kernel at large ray counts, and (b) BVH traversal for millions of rays all distributed in different angles around the LiDAR source leads to cache misses. [16] This data point highlights the potential of GnoSphere360 pipeline as a means for other high-resolution, high-throughput applications similar to a 360°LiDAR.

4.3 Hardware Timing Comparisons between OptiX and GnoSphere360

Fig. 5 provides the GPU timing breakdown of GnoSphere360 and OptiX. It is noteworthy, that GnoSphere360 ***does not reuse previous frames*** for the next

frame (unlike OptiX's IAS refit) and performs all computations per frame. Despite this, GnoSphere360 holds up competitively against the RT-core accelerated ray-tracing framework, and surpassing it at high ray-count LiDAR configurations. A prior-frame caching strategy for Stages 1 and 2 represents the most direct path to further performance gains.

Looking at the GPU time within OptiX, we can observe an equal distribution of time on IAS refit and ray launch. Ray launch includes the total time taken for the ray directions to be generated, the BVH in the respective directions to be traversed and the ray-triangle intersections to be performed. Comparatively we can see almost similar times for first two stages of the GnoSphere360 pipeline, whereas the final gnomonic projection stage contributes negligibly by comparison.

4.4 Similarity of LiDAR Point Generation

We evaluate the geometric similarity between LiDAR point clouds generated by OptiX ray-tracing and GnoSphere360 using nearest-neighbor (NN) distance. For each point in the GnoSphere360 output, we find the nearest neighbor from the OptiX output point cloud and compute their distance. This gives an estimate of how closely GnoSphere360 resembles the same spatial distribution of points produced by hardware-accelerated raytracing.

Upon initial evaluation, we found inverted mesh normals and planar one-sided meshes (fences, leaves, etc.), which were artifacts caused by exporting towns from CARLA. These mesh errors were corrected manually and two-sided triangles were used for one-sided meshes. A gated function was used to remove erroneous projections.

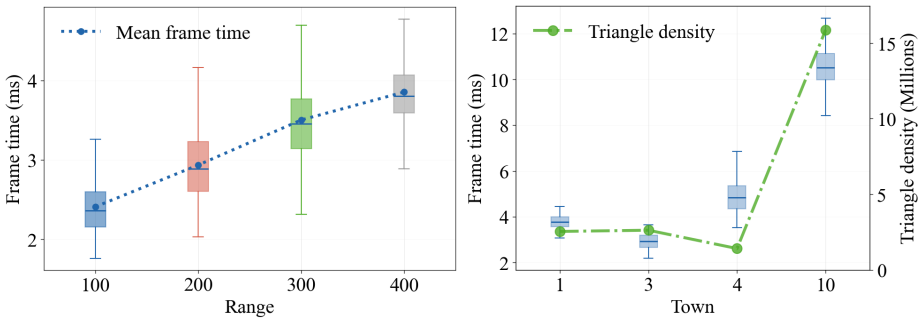
Table 5: Distribution of nearest-neighbour distances between point clouds generated by OptiX and GnoSphere360.

NN Distance (m)	Count	Percentage (%)
$0.00 - 0.20$	9,589,134	99.967964
$0.20 - 0.40$	630	0.006568
$0.40 - 0.60$	739	0.007704
$0.60 - 0.80$	400	0.004170
$0.80 - 1.20$	546	0.005692
$1.20 - 1.60$	408	0.004253
$1.60 - 2.00$	350	0.003649
Total	9,592,207	100.000000

We report the distribution of NN distances across all frames of the point cloud with the baseline configurations in Table 5. The results show that 99.97% of the generated points reside within 0.20m of the OptiX counterpart. Therefore there is a strong geometric alignment between the two methods.

It is important to note that GnoSphere360 and OptiX rays are initialized in identical spherical directions in a deterministic setting, hence the NN distance represents per-ray depth disagreement. Therefore the results reflect numerical precision of GnoSphere360, and confirm that geometrically faithful point clouds are produced. Since GnoSphere360 computes ray-triangle depth exactly rather than approximating it, this accuracy is not traded off against speed.

4.5 Ablation study



(a) LiDAR Range limit comparison **(b)** Evaluation of different towns

Fig. 6: Ablation study on different towns and range values for the LiDAR sensor (a) shows LiDAR range limit comparison for triangle projection in GnoSphere360 with frame time statistics; (b) shows evaluation of frame time in different towns along with triangle density in GnoSphere360

We perform ablation experiments for different hyperparameters of GnoSphere360 such as maximum range, scenes, and LiDAR models. These give us an insight into how the performance of GnoSphere360 changes across varying scenarios and across varying stages of the pipeline. Lowering the maximum range of the LiDAR leads to greater triangles being culled during spherical projection in stage 1, and should therefore be reflected on the frame timings as the propagating stages all benefit from culling. Ablating across scenes according to CARLA-based towns (in our case towns 01, 03, 04, and 10HD) provide us an understanding of how the distribution of triangles (density) within the scene impacts the stages 1 and 2, which are both triangle-parallel. Finally, performing our runs on various real-world LiDAR models (and a hypothetical extreme) provides valuable insights on how the ray-parallel third stage behaves as ray-count is increased.

Range Ablation We vary the maximum range allowed before the triangles are culled and monitor the mean performance over time. The graph in Figure 6a shows a linear relationship between increase in range to increase in per frame time. This is expected, as increasing the maximum range of the LiDAR

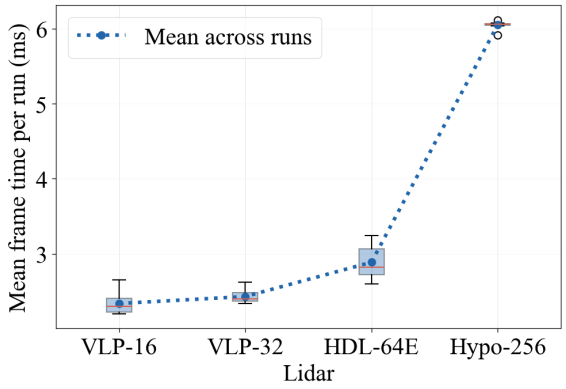


Fig. 7: Frame times for each different real-world physical LiDAR models, including one hypothetical model of 256-channels listed in Table 2.

increases the bounding volume of triangles in the region around the LiDAR position. Increase in range corresponds to cubic increase of bounding volume, but as most triangles are concentrated close to ground level the relationship is closer to quadratic. Despite this, having a linear relationship is noteworthy which likely arises from the multiple culling operations.

Cross-scene Generalization We vary the LiDAR data generation in four different simulated towns in CARLA with the baseline HDL-64E LiDAR. Figure 6b showcases the frame time statistics of a run for each town and correlates that against the triangle density (number of triangles in unit area). We find that the mean frame time of LiDAR data generation of a scene correlates directly with the triangle density within the range of the LiDAR. Town10HD is a small, highly urban town with lots of man-made structures with a much greater number of triangles (about 21M) compared to the others. While Town03 has a larger total number of triangles, the town is much sparser in comparison - therefore reducing the number of triangles being processed within the range of the LiDAR model.

LiDAR Sensor Model Ablation Physical LiDAR models can easily be implemented in GnoSphere360 due to our attention to LiDAR-specific topology. We have recreated all the listed real-world LiDAR sensors in Table 2, as well as hypothetical 256-channel and 350-channel LiDARs, each with 4 times the number of horizontal channels. The hypothetical LiDARs are used to stress test each method for an extremely large number of rays. Fig. 7 shows the result of running each listed LiDAR through the Town03 setup for 10 runs. Our results show that, as expected, frame time increases with the number of rays, as more rays have to be processed, and the spherical grid becomes more dense. For the hypothetical model with 256 channels, GnoSphere360 runs at a 6 ms mean frame time over the 10 runs.

5 Conclusion

We present a method to generate LiDAR data points in ADS simulations using physics-based principles by projecting each triangle onto a unit sphere, binning the triangle based on azimuth and elevation and finally, using Gnomonic projection to find the final point coordinates. The method, **GnoSphere360**, is shown to have extremely robust performance against NVIDIA OptiX raytracing framework and is about 35 times faster than the native CARLA LiDAR simulator for baseline HDL-64E LiDAR with 288K rays. If the number of rays is increased to 4.6M rays, GnoSphere360 offers a speedup of about 2.5 times over OptiX, which jumps to a speedup of 3.65 times when the rays are increased to 6.3M. Regarding similarity of LiDAR point clouds, we find that almost 99.97% of the output point cloud from GnoSphere360 lies within 0.2 m of OptiX point cloud.

We also add a detailed ablation study for the maximum range of the LiDAR data generation, time-based consistency of each frame generation in the different simulation towns of CARLA and finally, the effect of different LiDAR models on the generation of data. We find that there is almost a linear relationship between range and frame time. We also compute a mean time-frame for generating LiDAR data using HDL-64E parameters for four different simulated towns in CARLA and compare them against the triangle density. We find that there is a direct correspondence against the triangle density with Town10HD taking double the time of other towns due to larger than triple triangle density. Finally, we change the model of LiDAR used for data generation and show that GnoSphere360 can run different physical LiDAR models at similar speeds while generating data for an extremely dense hypothetical LiDAR model at double the mean frame time.

Our future work would be to further test the potential of gnomonic projection as a depth estimator and integrate it into the CARLA framework. Furthermore, we envision implementation and empirical validation of physics-based extensions such as material-based intensity/reflectance, drop-off, and Gaussian noise, which the preserved triangle-point associations in GnoSphere360 already support architecturally.

References

1. Bechtold, S., Höfle, B.: 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 Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences **3**, 161–168 (2016)
2. Caccia, L., Van Hoof, H., Courville, A., Pineau, J.: Deep generative modeling of lidar data. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 5034–5040. IEEE (2019)
3. Denis, L., Royen, R., Bolsée, Q., Vercheval, N., Pižurica, A., Munteanu, A.: Gpu rasterization-based 3d lidar simulation for deep learning. Sensors **23**(19), 8130 (2023)
4. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. In: Conference on robot learning. pp. 1–16. PMLR (2017)

5. Fang, J., Zhou, D., Yan, F., Zhao, T., Zhang, F., Ma, Y., Wang, L., Yang, R.: Augmented lidar simulator for autonomous driving. IEEE Robotics and Automation Letters **5**(2), 1931–1938 (2020)
6. Goldman, R.: An integrated introduction to computer graphics and geometric modeling. CRC Press (2009)
7. Gusmão, G.F., Barbosa, C.R.H., Raposo, A.B.: Development and validation of lidar sensor simulators based on parallel raycasting. Sensors **20**(24), 7186 (2020)
8. Hanke, T., Schaermann, A., Geiger, M., Weiler, K., Hirsenkorn, N., Rauch, A., Schneider, S.A., Biebl, E.: Generation and validation of virtual point cloud data for automated driving systems. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). pp. 1–6. IEEE (2017)
9. Kusari, A.: State of modeling and simulation in autonomous vehicles. In: Autonomous Vehicles and Systems, pp. 303–334. River Publishers (2024)
10. Li, W., Pan, C., Zhang, R., Ren, J., Ma, Y., Fang, J., Yan, F., Geng, Q., Huang, X., Gong, H., et al.: Aads: Augmented autonomous driving simulation using data-driven algorithms. Science robotics **4**(28), eaaw0863 (2019)
11. Li, Y., Ibanez-Guzman, J.: Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. IEEE Signal Processing Magazine **37**(4), 50–61 (2020)
12. Möller, T., Trumbore, B.: Fast, minimum storage ray-triangle intersection. Journal of Graphics Tools **2**(1), 21–28 (1997)
13. Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., et al.: Optix: a general purpose ray tracing engine. Acm transactions on graphics (tog) **29**(4), 1–13 (2010)
14. Rosique, F., Navarro, P.J., Fernández, C., Padilla, A.: A systematic review of perception system and simulators for autonomous vehicles research. Sensors **19**(3), 648 (2019)
15. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and service robotics: Results of the 11th international conference. pp. 621–635. Springer (2017)
16. Tozlu, Y.S., Zhou, H.: Cooprt: Accelerating bvh traversal for ray tracing via cooperative threads. In: Proceedings of the 52nd Annual International Symposium on Computer Architecture. p. 166–179. ISCA '25, Association for Computing Machinery, New York, NY, USA (2025). `https://doi.org/10.1145/3695053.3731118`, `https://doi.org/10.1145/3695053.3731118`
17. Wald, I., et al.: Owl: The optix wrappers library. GitHub repository (2020), `https://github.com/NVIDIA/OWL`, accessed: 2026-06-30
18. Zyrianov, V., Che, H., Liu, Z., Wang, S.: Lidardm: Generative lidar simulation in a generated world. In: 2025 IEEE International Conference on Robotics and Automation (ICRA). pp. 6055–6062. IEEE (2025)
19. Zyrianov, V., Zhu, X., Wang, S.: Learning to generate realistic lidar point clouds. In: European Conference on Computer Vision. pp. 17–35. Springer (2022)