

Hardware-assisted Range Image Generation for LiDAR Point Clouds

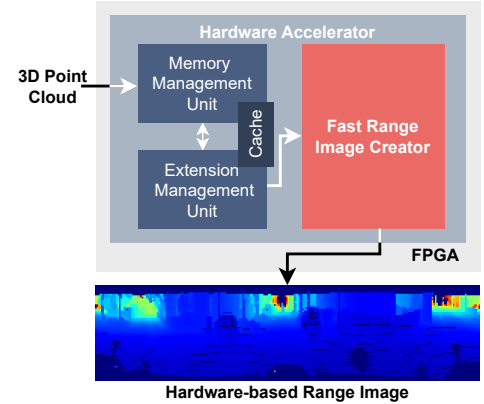
Fábio Oliveira¹, Vitor Ferreira¹, Sandro Pinto¹, and Tiago Gomes¹

¹ALGORITMI Research Center/LASI, University of Minho, 4800-058 Guimarães, Portugal

Manuscript received day month year; revised day month year; accepted day month year. Date of publication day month year; Date of current version day month year.

Abstract—As the need for efficient 3D data processing increases, particularly in autonomous driving applications, fast and resource-efficient methods for structuring and representing LiDAR data, such as range images (RIs), have become crucial. This letter presents a hardware-accelerated algorithm for generating RIs from 3D LiDAR point clouds, tailored for embedded platforms featuring Field-Programmable Gate Arrays (FPGAs). The proposed solution enables real-time point cloud processing, achieving up to a 605% performance improvement over the PCL desktop implementation, and up to 470% when deployed in FPGA, while maintaining minimal hardware resource utilization.

Index Terms—Range Image, Light Detection and Ranging (LiDAR) sensors, field-programmable gate array (FPGA).



I. INTRODUCTION

As technology continues to advance, the need for efficient 3D representations is becoming increasingly important across various industries, particularly in the automotive sector. Autonomous driving applications rely on accurate 3D representations of the surrounding environment, making Light Detection and Ranging (LiDAR) a key sensor for vehicle navigation and environmental perception. While LiDAR systems play a crucial role in tasks such as data segmentation and object detection, efficiently managing and processing point cloud (PC) data still poses significant challenges [1]. One common approach to address this challenge is converting 3D PC data into a structured 2D format, such as a range image (RI) [2]. This transformation enables more efficient data representation, handling, and storage, while reducing computational overheads without compromising the local geometry details of the environment.

Several methods exist for projecting 3D PC data into an RI, including mapping each LiDAR laser channel [3]–[5] or using the elevation angle [6]–[8] to assign points to corresponding image rows. The latter approach results in less quantization error by avoiding intra-frame distortion caused by LiDAR ego-motion [9]. However, generating an RI from high-resolution 3D PC data requires significant computational resources, including powerful processing systems and large memory utilization. These requirements hamper the utilization of this technique in real-time embedded systems for automotive applications, which are often constrained by limited hardware resources [1], [10].

This letter presents HARIG, a hardware-accelerated solution for generating Range Image (RI) from 3D LiDAR point clouds that leverages the acceleration capabilities of embedded systems with field-programmable gate array (FPGA) technology. To the best of our knowledge, this is the first work to present a fully pipelined, hardware-accelerated RI generator specifically tailored for automotive LiDAR sensors. The approach integrates tightly with embedded FPGA platforms and is optimized for real-time deployment in constrained environments. Compared to the Point Cloud Library (PCL) [11], an open-source library of algorithms for processing 2D/3D point cloud data, HARIG achieves a performance improvement of nearly 605% in generating an RI on a desktop platform, and up to 470% when generating the same RI in hardware using point clouds from a Velodyne VLS-128 sensor.

II. RANGE IMAGE TECHNIQUES

An RI is a 2D projection of a 3D scene, where each pixel value corresponds to the distance (or range) from the sensor's origin to a point in 3D space. Unlike traditional images, which encode color or intensity, an RI encodes the distance to an object at each pixel location. This transformation simplifies the complex and the unstructured data generated by LiDAR sensors, making it easier to process while preserving the spatial relationships between points in the 3D space. This approach can be used for different LiDAR applications, such as ground segmentation and point cloud data compression [10], [12].

A LiDAR sensor usually provides raw spherical data, which includes the azimuth θ (horizontal angle), the elevation ϕ (vertical angle), the distance ρ (range), and the intensity (reflection strength) of each detected point. This data is then used to create the RI, where

Corresponding author: Tiago Gomes (e-mail: mr.gomes@dei.uminho.pt).

Associate Editor: xxxx.

Digital Object Identifier xxxxxxxxxxxxxxxxxxxxxx

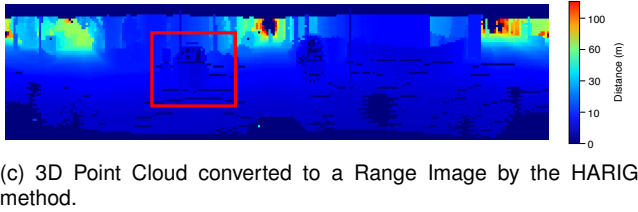
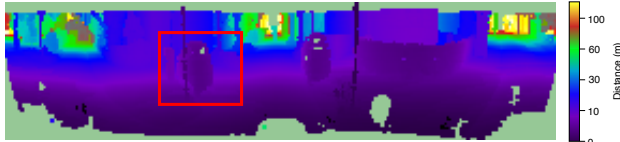
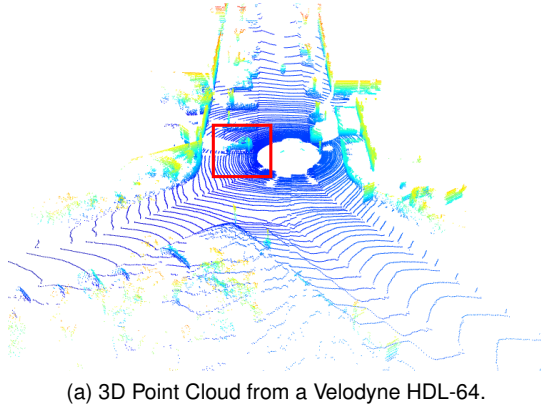


Fig. 1. HARIG vs PCL range image.

TABLE 1. Range Image Size for Resource Optimization (HDL-64).

RI Width	RI Height	PC Frame Size (MB)	RI Size (MB)	Quantization Error (cm)		Point Loss (%)	
				HARIG	PCL	HARIG	PCL
4500	64	1.85	1.09	2.29	0.49	-11.10	+81.20
	128	1.85	2.19	1.34	0.11	-1.60	+207.1
	256	1.85	4.39	0.69	0.04	-0.80	+264.2
4096	64	1.85	1.00	2.65	0.50	-12.30	+67.10
	128	1.85	1.99	1.34	0.19	-1.90	+195.0
	256	1.85	4.00	0.69	0.04	-1.04	+255.0
2048	64	1.85	0.50	2.67	2.38	-27.66	-16.10
	128	1.85	1.00	1.41	1.12	-10.33	+53.86
	256	1.85	2.00	0.78	0.69	-8.60	+92.30
1024	64	1.85	0.25	2.71	10.88	-62.20	-57.10
	128	1.85	0.50	1.54	4.77	-51.25	-12.40
	256	1.85	1.00	1.01	2.73	-48.60	+2.30

approach ensures efficient memory utilization during RI creation, enhancing processing speed by eliminating the need for dynamic memory management at runtime. However, static sizing can result in points being excluded from the RI, making it crucial to find an optimal balance between the RI size and the number of points that can be accurately represented. To evaluate different RI size configurations, HARIG was tested with the Seq.00 of the SemanticKITTI dataset, collected from a Velodyne HDL-64 sensor. For this sensor, the RI width would need to be at least 4500 pixels to capture every point on the horizontal plane. However, different configurations were evaluated and the results are summarized in Table 1.

HARIG allocates the nearest power-of-two width for efficient processing and memory allocation. Therefore, the nearest width for the 4500 pixels would be 4096. For selecting the RI height, the nearest value close to the sensor channels (the number of individual laser beams used to capture the point cloud) is 64. However, the HDL-64 sensor is non-linear, meaning its laser pulses are fired at varying vertical angles rather than being uniformly distributed causing point overlap and resulting in empty spaces in the RI. To address this, the selected RI height was 128, which reduces both quantization error and point loss. To achieve a fast RI generation, while keeping low its size (in MB), HARIG uses for this sensor a width of 2048 pixels. However, this width size comes at the cost of few points (10.33%) that cannot be recovered when the PC is reconstructed from the RI. In contrast, with the PCL implementation, the reconstructed PC contains more points (53.86%) than the original, which is due to point interpolation and causing bigger RI sizes. From this preliminary study we conclude that the optimal RI configuration for the Velodyne HDL-64 when using HARIG is 2048×128 , offering a good balance between memory size and quantization error, with a point loss of approximately 10.33%.

Figure 2 shows the distribution of points left outside the RI for the 2048×128 and 4096×64 configurations. Although both configurations have similar point loss, in the 2048×128 configuration (Figure 2b) point loss occurs primarily in denser areas of the PC. In contrast, the 4096×64 (Figure 2c) can result in the complete loss of a channel section due to overlapping points, visible by the horizontal lines of sequential lost points. While the PCL implementation achieves a lower quantization error, in some configurations, the interpolation process adds a considerable number of points to the reconstructed PC. This leads to increased resource utilization and slower processing times, as will be discussed further.

$$u = \left\lfloor \frac{1}{2} * \left(1 + \frac{\theta}{\pi}\right) * (w - 1) \right\rfloor \quad (1)$$

$$v = \left\lfloor \frac{\phi_{up} - \phi}{\phi_{up} - \phi_{down}} * (h - 1) \right\rfloor \quad (2)$$

III. HARIG ALGORITHM

Figure 1 illustrates a 3D PC frame from a HDL-64 sensor (Figure 1a), and its corresponding RIs using the PCL implementation (Figure 1b), and the HARIG algorithm (Figure 1c).

A. Design Considerations

HARIG was designed to run on embedded platforms with hardware acceleration support, while minimizing execution time and memory resource usage. Unlike the RI implementation provided by the PCL, HARIG statically allocates the size (width $\times$ height) of the RI, which must be selected based on the specific sensor in use. This

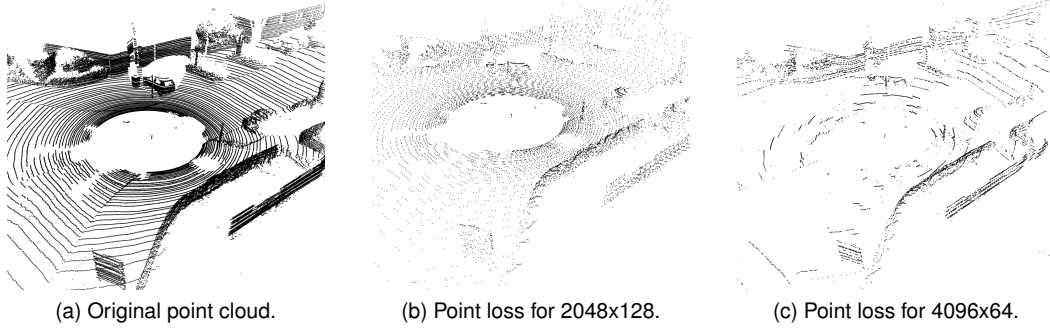


Fig. 2. Visual representation of the point loss for different point cloud frames reconstructed from an RI.

B. Implementation

HARIG was developed and tested as both a software and hardware-accelerated library within the ALFA framework. ALFA is a ROS-based tool that abstracts the underlying hardware and interfaces with LiDAR sensors, enabling seamless integration across various platforms and multiple LiDAR datasets. In this setup, the dataset coordinates are provided in a ROS-based format, thus, an additional step is required to convert the PC from Cartesian coordinates to spherical coordinates.

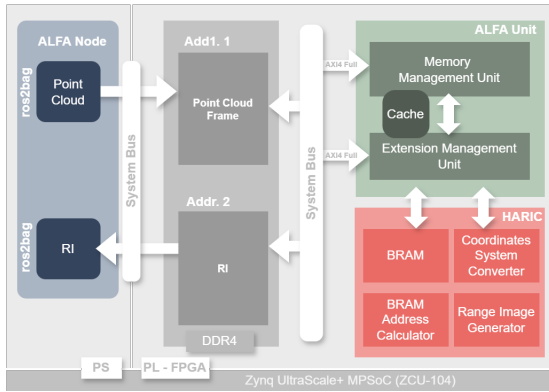


Fig. 3. HARIG Hardware Architecture.

1) *Hardware Architecture*: The hardware architecture of the HARIG accelerator is illustrated in Figure 3. Sensor data is written to the DDR4 memory by the ALFA-Node, while the ALFA-Unit manages the interaction with the HARIC hardware extension, controlling how data is transferred from the DDR4 memory. The HARIC accelerator first converts the point cloud frame from spherical to Cartesian coordinates in the Coordinates System Converter block. Then, the RI coordinates are computed, and the range values are stored in a Block RAM (BRAM), with the memory addresses computed from the RI coordinates. Once the RI is fully generated, the ALFA-Unit transfer it back to the DDR4 memory, making it accessible to the ALFA-Node. The entire process is fully pipelined, enabling fast RI generation while new data continues streaming from the ALFA-Node.

IV. EVALUATION

The HARIG algorithm was evaluated across three different environments: a desktop system, an embedded system with a software-only implementation, and an embedded system with hardware

acceleration. The framework executes on a Ubuntu 22.04.3 LTS Linux distribution, using PCL version 1.12.1 and the ROS2 Humble distribution. The desktop system featured an Intel Core i5-8279U processor with 4 CPU cores at 2.4 GHz, and 8 GB of RAM. The embedded setup included a ZCU104 Evaluation Kit, powered by the Zynq UltraScale+ MPSoC. This MPSoC includes a quad-core Arm Cortex-A53 application processor, a dual-core Cortex-R5 real-time processor, and FPGA technology with 504K logic cells and 1728 DSP slices, supported by 2 GB of DDR4 memory.

The evaluation used datasets collected in various environments with different Velodyne LiDAR sensors, including the VLS-16, HDL-32, HDL-64, and VLP-128. The main goal is to evaluate the algorithm's performance with different RI sizes, optimized for each sensor. For the VLP-16 and HDL-32 sensors, where the laser channels are uniformly distributed, RI heights of 16 and 32 pixels were selected, respectively. For the HDL-64 and VLP-128 sensors, RI heights of 128 and 256 pixels were used, respectively. The quantization error was calculated using Equation 3, and represents the average difference between the original point cloud and the reconstructed point cloud, where m is the number of points in the reconstructed point cloud, P_i is the original point, and P_i' is the reconstructed point.

$$QE = \frac{1}{m} \cdot \sum_i \|P_i - P_i'\| \quad (3)$$

A. Quantization Error and Point Loss

Table 2 summarizes the quantization error and the evaluation of points loss. The VLP-16 achieves 1.7 cm quantization error with 1.23% point loss in software, and 1.6 cm with 1.8% point loss in hardware. The HDL-32 shows 1.9 cm quantization error with 0.1% point loss in software, and 1.7 cm with 0.2% point loss in hardware. The HDL-64 shows 1.4 cm quantization error with 10.33% point loss in software, and 1.8 cm with 10.8% point loss in hardware. The VLS-128, with the highest resolution, exhibits 3.4 cm quantization error and 28.9% point loss in software, and 3.8 cm quantization error with 31.2% point loss in hardware.

B. Execution Performance

Table 3 compares the execution times (in ms) of HARIG and the PCL RI implementation across VLP-16, HDL-32, HDL-64, and VLS-128 sensors, including desktop, embedded software, and hardware accelerator versions. The measurements account for overheads such

TABLE 2. Quantization Error and Point Loss.

	RI Width	RI Height	PC Frame Size (MB)	RI Size (MB)	Version	Quantization Error (cm)	Point Loss (%)
VLP-16	4096	16	0.39	0.25	SW	1.7	-1.23%
					HW	1.6	-1.80%
HDL-32	4096	32	0.64	0.50	SW	1.9	-0.10%
					HW	1.7	-0.20%
HDL-64	2048	128	1.85	1.00	SW	1.4	10.33%
					HW	1.8	-10.80%
VLS-128	2048	256	3.14	2.00	SW	3.4	-28.93%
					HW	3.8	-31.20%

TABLE 3. Performance execution evaluation (ms).

	VLP-16	HDL-32	HDL-64	VLS-128
Desktop - PCL	27.17	39.23	123.1	246.2
Total (Desktop - PCL)	27.52	39.72	132.8	265.1
Desktop - HARIG	6.91	8.44	22.46	35.80
Total (Desktop - HARIG)	7.37	8.96	23.56	37.60
vs PCL	(273.43%)	(343.65%)	(463.49%)	(605.05%)
Embedded - PCL	50.1	76.6	236.2	406.5
Total (Embedded - PCL)	51.6	79.2	243.1	416.9
Embedded - HARIG	12.32	20.60	70.13	110.82
Total (Embedded - HARIG)	13.84	23.01	75.76	120.02
vs PCL	(272.8%)	(244.2%)	(220.9%)	(247.35%)
Hardware - HARIG	1.71	2.82	8.10	13.70
vs PCL embedded	(2829%)	(2616%)	(2816%)	(2867%)
vs HARIG embedded	(620.64%)	(630.57%)	(765.81%)	(708.91%)
Total (Hardware - HARIG)	8.83	18.12	40.09	73.13
vs PCL embedded	(464.3%)	(337.1%)	(506.38%)	(470.1%)
vs HARIG embedded	(56.77%)	(27.00%)	(89.00%)	(63.70%)

as loading, processing, publishing, and data transfer between the software and FPGA.

The PCL desktop versions take 27.52 ms, 39.23 ms, 123.1 ms, and 246 ms for the respective sensors, while HARIG reduces these times to 7.37 ms, 8.96 ms, 23.56 ms, and 37.60 ms. On the embedded system, PCL requires 51.6 ms, 79.2 ms, 243.1 ms, and 416.9 ms, while HARIG processes the same data in 13.84 ms, 23.01 ms, 75.76 ms, and 120.02 ms, showing performance improvements of 272.8%, 244.2%, 220.9%, and 247.3%, respectively. In hardware, HARIG further reduces processing times to 8.83 ms, 18.12 ms, 40.09 ms, and 73.13 ms for the four sensors.

C. Hardware Resources

The performance improvements for the hardware implementation come at the cost of FPGA resources. Table 4 summarizes the hardware components required for deploying the HARIG algorithm alongside the ALFA framework used in the evaluation. This includes Flip-Flops (FFs), look-up tables (LUTs), LUT random access memories (LUTRAMs), Block RAMs (BRAMs), and DSP blocks. The most utilized resource is BRAM, which accounts for over 76% of available capacity, primarily due to the memory required to store the PC frame and generate the RI. Other components, such as FFs, LUTs, and LUTRAMs, remain below 4% utilization.

V. CONCLUSION

This letter presents HARIG, a software-based and hardware-accelerated RI generator that achieves significant performance

TABLE 4. Hardware resources utilization.

Comp.	FF (460800)	LUT (230400)	LUTRAM (101760)	BRAM (312)	DSP (1720)
ALFA-Unit	10872 (2.36%)	3064 (1.32%)	0	0	1 (0.06%)
ZCU104	5710 (1.24%)	4386 (1.90%)	939 (0.92%)	0	0
HARIG	692 (0.15%)	1734 (0.75%)	0	240 (76.9%)	4 (0.23%)
Total	17274 (3.75%)	9181 (3.98%)	939 (0.92%)	240 (76.9%)	5 (0.29%)

improvements while minimizing hardware resource usage. Given that most LiDAR sensors operate at a frame rate of 10 Hz, generating the RI in under than 100 ms is crucial to avoid frame loss, especially for high-resolution sensors like the VLS-128. In contrast to the PCL RI implementation, the hardware-accelerated version of HARIG generates an RI for the VLS-128 in less than 75 ms, ensuring no frame loss during processing. Future work will focus on exploring new techniques to further reduce the point loss rate of HARIG and exploring data compression methods to decrease the size of the RI. This design can serve as a foundation for future real-time LiDAR applications in robotics and autonomous systems.

ACKNOWLEDGMENT

This work has been supported by FCT - *Fundação para a Ciência e Tecnologia* within the R&D Unit Project Scope UID/00319/Centro ALGORITMI (ALGORITMI/UM). ALFA Project is available at <https://github.com/alfa-project> under Apache License 2.0.

REFERENCES

- [1] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Trans. on Intelligent Transportation Systems*, vol. 23, pp. 6282–6297, 2022.
- [2] Y. Feng, S. Liu, and Y. Zhu, "Real-Time Spatio-Temporal LiDAR Point Cloud Compression," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 766–10 773.
- [3] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 669–12 678.
- [4] F. Moosmann and C. Stiller, "Velodyne SLAM," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 393–398.
- [5] M. Velas, M. Spanel, M. Hradis, and A. Herout, "CNN for IMU assisted odometry estimation using velodyne LiDAR," in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2018, pp. 71–77.
- [6] H. Wang, B. Wang, B. Liu, X. Meng, and G. Yang, "Pedestrian recognition and tracking using 3D LiDAR for autonomous vehicle," *Robotics and Autonomous Systems*, vol. 88, pp. 71–78, 2017.
- [7] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.
- [8] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments," in *Robotics: science and systems*, vol. 2018, 2018, p. 59.
- [9] T. W. et al., "Detailed analysis on generating the range image for lidar point cloud processing," *Electronics*, vol. 10, no. 11, 2021.
- [10] W. Huang, H. Liang, L. Lin, Z. Wang, S. Wang, B. Yu, and R. Niu, "A Fast Point Cloud Ground Segmentation Approach Based on Coarse-To-Fine Markov Random Field," *IEEE Trans. on Intell. Transp. Syst.*, vol. 23, pp. 7841–7854, 2022.
- [11] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.
- [12] S. Wang, J. Jiao, P. Cai, and L. Wang, "R-PCC: A Baseline for Range Image-based Point Cloud Compression," in *2022 Int. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 10 055–10 061.