An Efficient Hardware Volume Renderer for Convolutional Neural Radiance Fields

Xuexin Wang[1], Yunxiang He[1], Xiangyu Zhang[1], Pingqiang Zhou[1] and Xin Lou[1,2]
[1]*School of Information Science and Technology, ShanghaiTech University*
[2]*Key Laboratory of Intelligent Perception and Human-Machine Collaboration, Ministry of Education*
Shanghai, China
Emails: {wangxx1, heyx1, zhangxy10, zhoupq, louxin}@shanghaitech.edu.cn

2024 IEEE International Symposium on Circuits and Systems (ISCAS) | 979-8-3503-3099-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCAS58744.2024.10558395

***Abstract*—Neural Radiance Fields (NeRF) has attracted growing attention in the fields of 3D reconstruction and rendering. However, straightforward NeRF algorithms encounter challenges in accurately capturing complex surface details with rich high-frequency information. A recent development known as Convolutional Neural Radiance Field (ConvNeRF) has demonstrated state-of-the-art results for these tasks. But it comes with substantial irregular computational requirements, particularly in the convolutional volume rendering phase. In this paper, we introduce a hardware accelerator designed to enhance the efficiency of convolutional volume rendering in ConvNeRF. Our approach includes the creation of specialized computation modules and corresponding on-chip memory system optimized for seamless support of gated convolutions and skip connections in ConvNeRF. To validate our design, we implement it in VerilogHDL and build a prototype using Field Programmable Gate Array (FPGA). We also map our design to 40nm CMOS technology. The evaluation results underscore the superiority of our accelerator in terms of energy efficiency when compared to an implementation on an NVIDIA 2080Ti GPU, offering approximately 84.6× more frames per watt.**

***Index Terms*—Neural radiance fields (NeRF), convolutional volume rendering, gated convolution.**

I. Introduction

In recent years, 3D reconstruction and rendering have experienced a profound transformation, largely driven by the advent of of Neural Radiance Fields (NeRF). While NeRF has enabled a wide range of graphics and vision applications, it encounters limitations when dealing with scenes rich in high-frequency information. In cases where objects such as feathers, fur, and hair exhibit significant view-dependent variations in brightness and color, the intricacies of opacity become a prominent challenge, affecting both geometric and appearance reconstruction.

To enhance NeRF's capacity for rendering intricate objects, a novel Convolutional Neural Radiance Field (ConvNeRF) generation approach has been introduced [1]. As shown in Fig. 1, ConvNeRF employs an implicit neural radiance field, using a Multi-Layer Perceptron (MLP) to represent the scene and utilizes volumetric integration to predict density and color values along the casting rays. Moreover, ConvNeRF leverages a convolutional neural network (CNN) to further refine the view-consistent fine-detailed appearance and opacity output. The CNN used for volume rendering in ConvNeRF follows an encoder-decoder architecture similar to U-Net [11]. It distinguishes itself from conventional U-Net structures [2] by integrating gated convolution to capture nuanced features within the masked region, yielding superior results. Furthermore, through the amalgamation of features from diverse layers and images of varying scales, the gated convolutions enrich the feature set, thus facilitating the restoration of high-frequency details at the peripheries of objects.

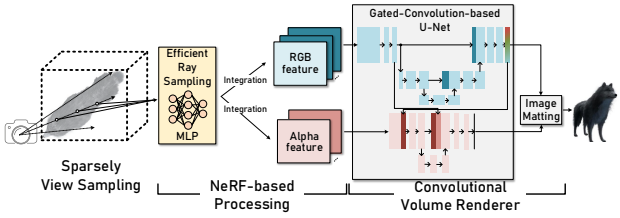

Fig. 1. The convolutional neural radiance field architecture, consisting of a NeRF-based processing module and a convolutional volume renderer.

Although ConvNeRF has delivered state-of-the-art results, it comes with substantial computational demands, especially in the volume rendering part. For the U-Net-like convolutional volume rendering in ConvNeRF, the inference of a single frame with a resolution of 500x800 necessitates approximately 291MB of features (FP32) and consumes a full power load of 250W with the NVIDIA RTX 2080Ti. While graphical processing units (GPUs) have historically managed significant computational and storage requirements, these demands still outstrip the capabilities of edge devices. While there are works focusing on efficient hardware accelerator for NeRF [12], there has been relatively little attention directed towards the convolutional volume rendering.

II. Related Work

A. Hardware Accelerators for NeRF

Hardware accelerators for NeRF have garnered significant attention in recent years. ICARUS [3] introduced a specialized architecture designed for NeRF that exhibits superior energy efficiency compared to GPUs. NeuRex [4] delved into the intricacies of leveraging sparsity in matrix-vector pairs and proposes a efficient architecture, albeit with increased power

This work was supported by the Shanghai Rising-Star Program (21QC1401400) and Central Guided Local Science and Technology Foundation of China (YDZX20223100001001). Corresponding arthor: Xin Lou.

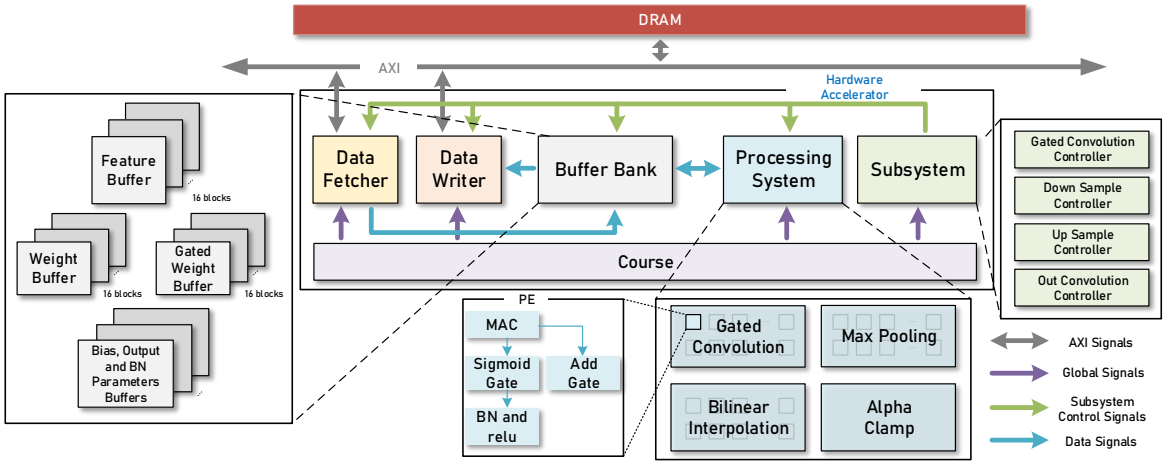

Fig. 2. The architecture of the proposed accelerator for convolutional volume rendering.

and area consumption. Gen-NeRF [5] proposed an algorithm-hardware co-design framework dedicated to accelerating generalizable NeRFs, enabling real-time, versatile NeRF applications. The realm of hardware acceleration for NeRF continues to undergo rapid development.

B. Hardware Accelerators for U-Net

Although there are many hardware accelerators for CNNs, there is only a few specialized designs for U-net like architectures that used in ConvNeRF. Two recent works [6] and [10] presented hardware accelerators for 3D U-Net, but without up-sampling modules for dimension alignment. The scale and architecture of the convolution modules is also not align with the requirements of convolutional volume rendering. [8] introduced a U-Net algorithm incorporating super-resolution techniques, coupled with a custom hardware accelerator. But the hardware accelerator proposed in [8] does not support gated convolutions and utilizes a pixel shuffle-based method for up-sampling, which is also unsuitable for ConvNeRF. On the other hand, when mapping the convolutional volume rendering computation to other existing CNN accelerators, the distinctive features of gated convolutions and skip connection can lead to notable inefficiencies.

To address the above-mentioned limitations in existing designs, we propose a specialized energy efficient hardware convolutional volume renderer for ConvNeRF. Our design includes native supports for gated convolution operations, a crucial feature frequently employed in convolutional neural rendering applications. We build a prototype system to validate the proposed accelerator based on FPGA, which reconstructs high frequency and global consistent appearance and opacity of fuzzy objects. We also implement and evaluate our design using 40nm CMOS technology. Demonstrations and evaluation results underscore the superior energy efficiency of the proposed accelerator in comparison to GPU implementations, as well as other dedicated accelerators, for convolutional neural rendering tasks.

III. Proposed Hardware Architecture

Fig.2 illustrates the overall architecture of the proposed hardware accelerator for convolutional volume rendering. The RGB and Alpha features [1] generated by the preceding NeRF model, which serve as the input for the accelerator, are stored in the dynamic random access memory (DRAM). An AXI interface establishes the connection between the DRAM and accelerator. Within the system, the *Course* module operates as the central control unit, responsible for orchestrating the sequence in which various subsystems are utilized. It additionally disseminates configuration information of different layers in the algorithm to all modules to accommodate the requirements of different layer computation. The *Subsystem* module consists of four distinct sub-modules, corresponding to specific operations in the convolutional volume rendering, i.e., gated convolution, down-sampling, up-sampling, and output convolution. When a specific module within the subsystem is activated, the associated data fetcher and computation modules within the processing system are invoked as required. A portion of the data residing in DRAM will be pre-fetched via the AXI bus and directed to the *Data Fetcher* module, where it is consolidated to match the required width and subsequently stored in the *Buffer Bank* module. The *Processing System* retrieves data from the *Buffer Bank* for further processing. The outcomes of the *Processing System* are subsequently preserved within the output buffer housed within the same buffer bank. Once a specific data threshold is reached, typically 4KB, the *Data Writer* module is triggered. It retrieves data from the output buffer and transfers it back to DRAM via the AXI bus. In cases where the *Processing System* requires data not present in the buffer bank, the *Data Fetcher* module is reactivated to pre-fetch the necessary data back into the *Buffer Bank*.

A. Buffer Bank

1) Feature Buffer: To facilitate computations with varying channel counts and improve hardware flexibility, we design a *Feature Buffer* module comprises 16 blocks of SRAM, each with a depth of 4096 and a width of 256 bits. A single block within the *Feature Buffer* can accommodate a maximum

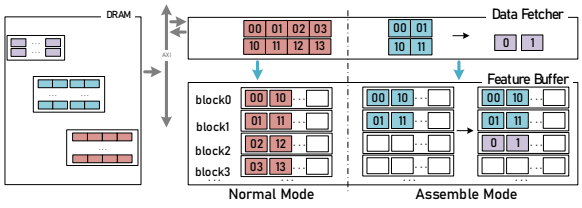

Fig. 3. *Feature Buffer* configurations for different computation modes.

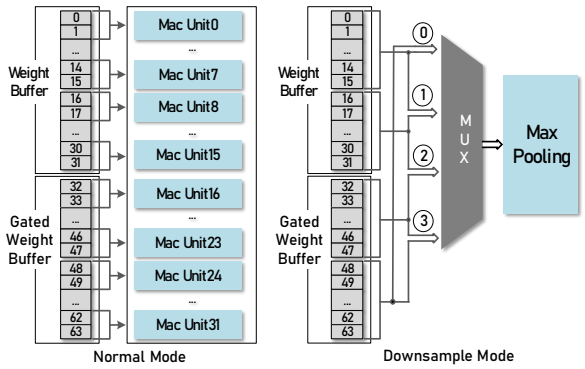

Fig. 4. *Weight Buffer* configurations for different modes.

storage of 4096 sets of 16-bit features, each with 16 channels. When a greater number of channels is required, our design supports the concatenation of multiple *Feature Buffer* blocks to augment the input channel count. The Assemble Mode is designed to merge features from different layers in the convolutional volume rendering architecture. As depicted in Fig. 3, when the Assemble Mode is activated, chip-select signals are used to dictate the activation of different blocks within the *Feature Buffer*. The concatenated features are subsequently retrieved from *Feature Buffer* forwarded to the *Processing System* for further processing.

2) Weight Buffer: As gated convolution requires two convolution operations with identical size, we incorporates two weight buffers, each spanning 16 blocks, within the *Buffer Bank*. Each block comprises two SRAM units, each with a depth of 128 and a width of 512 bits. When performing gated convolution operations, these 32 blocks, each associated with its respective MAC unit, provid the necessary weight data. When conducting the 2×2 max-pooling operation during down-sampling, these 32 blocks are divided into 64 units, wherein every 16 units retrieve data from one row of features sourced from the data fetcher. As illustrated in Fig.4, the data from each row will be cyclically stored in ⓪, ①, ②, and ③. Note the parallelism of the weight RAM, when the maxpooling operation starts, Weight Buffer will cyclically output ⓪①, ①②, ②③, and ③⓪, which corresponds to two rows of feature. This mechanism enhances the efficiency of the maxpooling operation.

B. Gated Convolution Module

As depicted in Fig.5, the *Gated Convolution* module consists of 16 identical processing elements (PEs). The MAC unit within a PE comprises two parts, corresponding to the generation of feature and mask in the gated convolution operation,

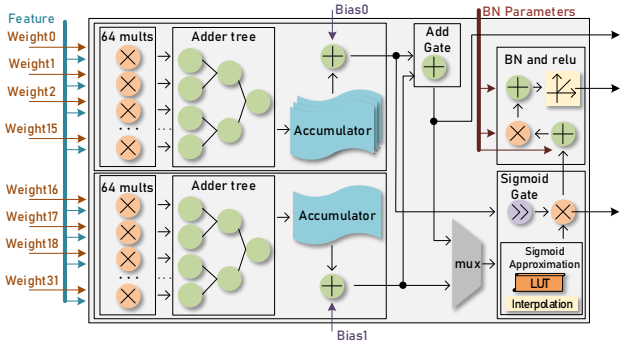

Fig. 5. Detailed architecture of the PEs in the Gated Convolution Module.

enabling parallel MAC operations. Specifically, each MAC unit consists of 64 16-bit by 16-bit fixed-point multipliers, a 64-to-1 adder tree, an accumulation unit, and a bias adder. There are slight differences in the design of the accumulation module for the two parts of the MAC unit. When conducting convolution operations to generate features, the continuously generated 16 convolution results are first stored in a register group. However, during convolution operations for mask generation, the 16 results are continually directed to either the add gate or the Sigmoid gate for mask generation. Subsequently, the feature undergoes a *masking* process, involving addition or multiplication operations with the mask. The Sigmoid Gate and Add Gate correspond to the gated convolution and output convolution operations, respectively. In our design, different modules are activated at distinct stages of the neural network to align with the specific operational requirements. The *Sigmoid Approximation* module employs a 'look-up table (LUT) + interpolation' technique [7]. Batch normalization operations are integrated into the MAC unit in our design, diminishing the necessity for repetitive memory access. Every operation within the unit is pipelined to enhance computational efficiency.

C. Bilinear Interpolation Module

Bilinear interpolation is a commonly used technique in convolutional volume rendering. As depicted in Fig. 6, a method similar to sliding-window based convolution is used in this work to implement bilinear interpolations. Two rows of features are sequentially read from the *Feature Buffer* and temporarily stored in two FIFOs, each with a depth of 3. Once the data is prepared, the computation unit undertakes interpolation operations on the elements located at positions ⓪, ①, ②, and ③. Once the interpolation operations completed, the elements at positions ②, ③, ④, and ⑤ within the FIFOs will be collectively shifted to positions ⓪, ①, ②, and ③. Since the required data is readily prepared in the FIFOs, the computation can proceed without waiting, allowing for continuous processing. As the data shifted forward, position ④ will be replaced by newly read data from the *Feature Buffer*. During the interpolation operations, data at position ⑤ will be filled. This cycle of operations continues until all rows of features have been traversed, i.e., completing the bilinear interpolation. Note that each interpolation computation

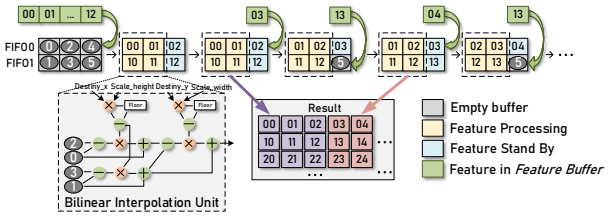

Fig. 6. Data flow and architecture of the Bilinear Interpolation module.

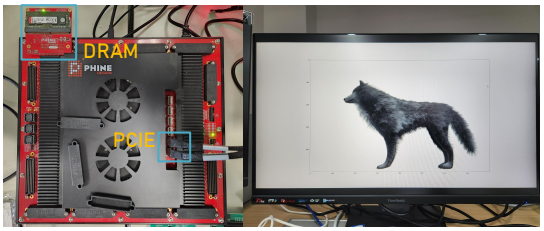

Fig. 7. The FPGA-based prototype system.

computes all the results involving the four data points at positions ⓪, ①, ② and ③. In this way, all features need to be read out only twice, which significantly reduces the on-chip SRAM read/write operations. The *Bilinear Interpolation* module supports simultaneous computation of features with a maximum of 64 channels.

D. Max Pooling and Alpha Clamp

The *Max Pooling* module is specifically designed for the 2x2 max pooling (the stride is 1) in down-sample operation. As discussed, this module will receive 2 rows of feature continuously from the *Weight Buffer*. It then compares and finds the largest number of the 4 elements in a 2x2 window. The *Max Pooling* module in our design supports simultaneous computation of features for up to 32 channels. Once the RGB and Alpha features have been fully prepared, the *Alpha Clamp* module will then execute an image matting process to produce the final RGB results.

IV. Implementation and Evaluation Results

To validate the proposed design, we implement our system in VerilogHDL. A proof-of-concept FPGA prototype system is further developed, comprising an AMD Virtex UltraScale+ VU19P, a DRAM daughter card, as well as a host PC for loading features, weights and other parameters. We employed the commonly used Wolf dataset from [9] to showcase the rendering quality. As shown in Fig.8, our design successfully restores high-frequency details in fuzzy objects, with no loss in peak signal-to-noise ratio (PSNR).

To evaluate the performance of our proposed design, we map our system to 40nm CMOS technology and employed Synopsys Design Compiler R-2020.09-SP5 for synthesis. The results are presented in Table I. As we can see, our design shows much better energy efficiency than the GPU implementation for ConvNeRF, delivering approximately 84.6× more frames per watt (F/W). While [8] and our design serve diverse neural network tasks, the proposed design demonstrates superior performance when compared to UArch [8].

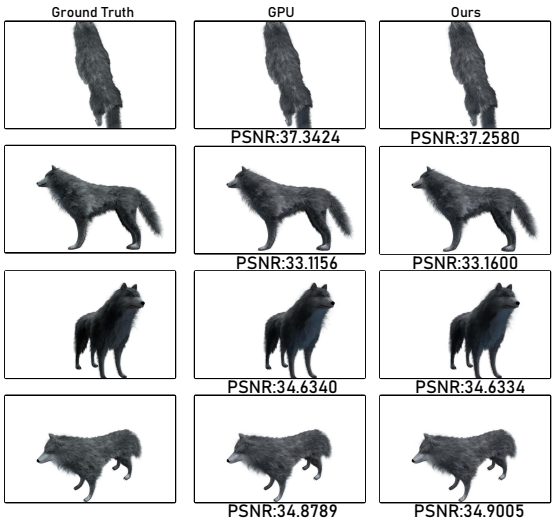

Fig. 8. Comparison of rendering results between GPU and the proposed hardware accelerator. PSNRs are presented below each respective results.

TABLE I
Comparison with other designs and implementations.

	NVIDIA 2080Ti	Sagitta [10]	UArch [8]	Ours
Network	ConvNeRF	3D U-Net	USR	ConvNeRF
Data Type	32b-float	8b	12b	16b
Tech Node (nm)	12	55	28	40
Area (mm^2)	754	13.5	7	13
Frequency (MHz)	1350	20-170	156	220
SRAM (MB)	29.5	0.225	0.777	2.51
Power (W)	250	0.027-0.605	-	0.302
Frame/W	0.087	-	-	7.358
TOPS/W	0.054	2.73-7.2	2.26	3.75

Frame/W: Frames per watt. Results in this table are obtained base on the Wolf dataset with rendering resolution of 500 × 800.

It is noteworthy that [8] and [10] lack support for gated convolution operations, which may introduce additional computational complexity and memory requirements when execute convulitional volume rendering tasks like ConvNeRF.

V. Conclusion

In this work, we propose an energy-efficient hardware volume renderer for ConvNeRF, which has demonstrated state-of-the-art results for reconstructing and rendering complex surface details. The proposed design incorporates specialized computation module and on-chip memory system optimized for seamless support of gated convolutions and skip connections in the ConvNeRF architecture. To validate the proposed design, we implement our system in VerilogHDL and build a proof-of-concept FPGA prototype system. The rendering results demonstrate that the proposed hardware accelerator produces outputs without any quality loss. To evaluate the energy efficiency, we further implement our design in 40nm CMOS technology. The evaluation results underscore the superiority of our accelerator in terms of energy efficiency when compared to GPU for ConvNeRF, offering approximately 84.6× more frames per watt.

REFERENCES

[1] H. Luo, A. Chen, Q. Zhang, B. Pang, M. Wu, L. Xu and J. Yu, "Convolutional Neural Opacity Radiance Fields," in Proceedings of IEEE International Conference on Computational Photography (ICCP), Haifa, Israel, 2021, pp. 1-12, doi: 10.1109/ICCP51581.2021.9466273.

[2] Y. Jo and J. Park, "SC-FEGAN: Face Editing Generative Adversarial Network With User's Sketch and Color," in Proceedings of IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019, pp. 1745-1753, doi: 10.1109/ICCV.2019.00183.

[3] C. Rao, H. Yu, H. Wan, J. Zhou, Y. Zheng, M. Wu, Y. Ma, A. Chen, B. Yuan, P. Zhou, X. Lou and J. Yu, "ICARUS: A Specialized Architecture for Neural Radiance Fields Rendering". ACM Transactions on Graphics. 41, 6, Article 234, 2022, 14 pages. https://doi.org/10.1145/3550454.3555505

[4] J. Lee, K. Choi, J. Lee, S. Lee, J. Whangbo and J. Sim. "NeuRex: A Case for Neural Rendering Acceleration". in Proceedings of the Annual International Symposium on Computer Architecture (ISCA). Association for Computing Machinery, New York, NY, USA, 2023, Article 21, 1–13. https://doi.org/10.1145/3579371.3589056

[5] Y. Fu, Z. Ye, J. Yuan, S. Zhang, S. Li, H. You and Y. Lin. "Gen-NeRF: Efficient and Generalizable Neural Radiance Fields via Algorithm-Hardware Co-Design". in Proceedings of the Annual International Symposium on Computer Architecture (ISCA). Association for Computing Machinery, New York, NY, USA, 2023, Article 74, 1–12. https://doi.org/10.1145/3579371.3589109

[6] T. Glint, M. Awasthi and J. Mekie, "REDRAW: Fast and Efficient Hardware Accelerator with Reduced Reads And Writes for 3D UNet," in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10137190.

[7] https://github.com/xiph/rnnoise/blob/master/src/rnn

[8] X. Duan, Y. Chen, M. Li, Y. Rong, R. Xie and J. Han, "UArch: A Super-Resolution Processor With Heterogeneous Triple-Core Architecture for Workloads of U-Net Networks", IEEE Transactions on Biomedical Circuits and Systems, vol. 17, no. 3, pp. 633-647, June 2023, doi: 10.1109/TBCAS.2023.3261060.

[9] https://github.com/HaiminLuo/ConvNeRF

[10] C. Zhou, M. Liu, S. Qiu, X. Cao, Y. Fu, Y. He and H. Jiao, "Sagitta: An Energy-Efficient Sparse 3D-CNN Accelerator for Real-Time 3D Understanding", IEEE Internet of Things Journal, doi: 10.1109/JIOT.2023.3306435.

[11] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, 2015, pp. 234–241.

[12] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in Proceedings of European Conference on Computer Vision (ECCV), 2020, pp. 405–421.