Hardware Accelerated Synthetic X-ray Medical Image Generation Using HBM-based FPGAs

Sam Aanhane[1] Per Knops[2] Rob de Jong[2] Casper Cromjongh[1] Zaid Al-Ars[1]
[1]*Delft University of Technology, Delft, Netherlands* [2]*Philips Medical Systems, Best, Netherlands*
samaanhane@gmail.com

2025 IEEE Nordic Circuits and Systems Conference (NorCAS) | 979-8-3315-1501-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/NORCAS66540.2025.11231289

***Abstract*—Synthetic image generation involves the creation of artificially generated images that are indistinguishable from real ones. Conventional simulation-based image synthesis approaches suffer from intensive computational and memory throughput demands associated with physically accurate ray tracing through volumetric datasets. In this work, we propose an FPGA-based accelerator architecture capable of handling the computations required to simulate physically accurate X-ray images in real time. In addition, an algorithm is developed that can calculate the path of an X-ray through a phantom representing a physical model. To ensure real-time performance, a parallel accelerator architecture is proposed using a chain of accelerator kernels combined with High Bandwidth Memory architecture, which can simulate many rays concurrently, addressing the computational and memory throughput demands associated with simulation-based X-ray image generation.**

Performance evaluation of the simulation on an AMD Alveo U50 Data Accelerator card shows that an average speed-up of 12x over CPU-based implementations is possible, and allows for real-time image synthesis at a frame rate of 60 images/s. These findings highlight the advantages of FPGA acceleration for deterministic, high-speed synthetic image generation.

***Index Terms*—Hardware acceleration, X-ray, FPGA, HBM**

I. Introduction

Synthetic image generation involves the creation of artificially generated images that are indistinguishable from real ones. Creation of synthetic images is motivated by the fact that the acquisition of real-world images can be problematic compared to the use of synthetic images. It may be dangerous or harmful to collect images in a particular scenario. It could also be difficult or expensive to collect real images because an event is rare to capture in real life. As demand for high-quality datasets continues to grow, reliance on real data alone proves increasingly impractical. In the healthcare industry especially, gathering real-world data can be unsafe for the patient, unethical, or restrictive due to privacy legislation that protects against sharing sensitive data. Synthetic data can offer an alternative in these scenarios.

Synthetic X-ray medical images can be generated using a number of different approaches. The first is machine learning-based. However, this method can generate unpredictable outputs, which is undesirable in mission-critical applications such as the medical domain. The second approach is physics-based simulation and is generally implemented on CPUs and GPUs. This method simulates the interaction between X-rays and matter to compute an image. Using volumetric datasets derived from computed tomography or reconstructed rotational X-ray images, a ray tracing application can be developed that can compute the received intensity of radiation emitted by a source and registered by a detector. Because of the memory footprint of these detailed models and the vast number of intersection calculations for the rays, a ray-trace application has a high computational complexity and suffers from memory throughput challenges that are hard to mitigate in CPU and GPU implementations.

This paper addresses the computational and memory throughput challenges of conventional simulation approaches using an HBM-based FPGA architecture. To handle the large number of intersection calculations, a sequential hardware kernel implementation is proposed that can be used replicated to process many rays concurrently. The custom memory architecture incorporates a High Bandwidth Memory (HBM) to address the memory throughput challenges of conventional simulation-based approaches. Combined with a tuned layout based on the access pattern of the application, memory throughput challenges can be effectively mitigated.

Ultimately, the goal of this paper is to realize an architecture that can be used to generate synthetic X-ray images that are indistinguishable from real X-ray images in real-time. To achieve this goal, this paper makes the following contributions:

1) A high-throughput sequential architecture to accelerate ray tracing computations.
2) A custom HBM-based memory architecture that can access data at low latency.
3) Superior performance of synthetic image generation over CPU- and GPU-based implementations, enabling real-time use

This paper is organized as follows. Section II introduces the volumetric ray tracing algorithms. Section III proposes an FPGA architecture to address the identified bottlenecks and shows implementation details. Section IV presents the results of the implementation. Section V discusses related work, and Section VI ends with the conclusions.

II. Background

A. X-ray Attenuation

X-rays are a form of electromagnetic radiation with wavelengths ranging from 0.01 to 10 nanometers, corresponding to an energy range of 100 eV to 100 keV. X-rays are primarily generated through two mechanisms: Bremsstrahlung, where high-speed electrons decelerate upon striking a target material,

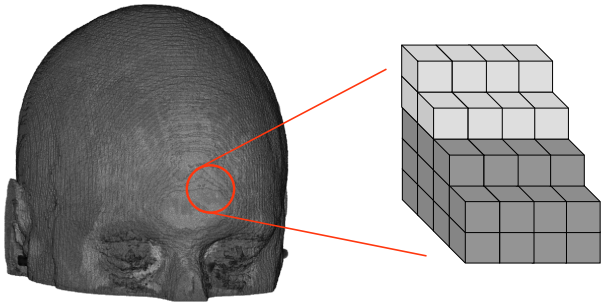


Fig. 1: Voxel model derived from a human subject

producing a continuous spectrum, and characteristic X-ray production, where electron transitions between atomic energy levels emit radiation at discrete wavelengths. When passing through matter, X-rays undergo interactions such as absorption and scattering, leading to attenuation. The X-ray attenuation depends on both the material properties and the X-ray photon energy. This determines the extent to which X-rays penetrate different materials.

Linear attenuation, expressed as μ, is a measure of how much the intensity of an X-ray beam is reduced the further it penetrates matter. Using the linear attenuation of a material, the Beer-Lambert's law [15] shown in Equation (1) describes the radiation intensity as it passes through a material, where I_0 describes the incident X-ray intensity, l the thickness of the material, and $\mu(x)$ the linear attenuation along the path of a ray at x.

$$I = I_0 e^{-\int_0^l \mu(x)dx} \tag{1}$$

A synthetic X-ray image can be generated by computing the intensity of a set of rays that strike a detector using the Beer–Lambert law. This requires evaluating a line integral, which is impractical within an FPGA. To resolve this, the problem is discretized by considering the attenuation for a given volume passed by a ray instead of an infinitesimal segment along its path. This approach replaces the integral with a summation, as shown in Equation (2).

$$I = I_0 e^{-\sum \mu_{x,y,z} \cdot l_{x,y,z}} \tag{2}$$

The attenuation coefficients for a given volume used in a simulation are derived from a real subject. The cubes that make up such a volume are called voxels. The voxel-based model used in the imaging process consists of three-dimensional pixels, or voxels, each assigned a linear attenuation coefficient that determines how X-ray intensity decreases as it passes through the material. A model, such as the voxel representation of a skull shown in Figure 1, is structured in layers that are processed sequentially. To compute the intensity of a ray, its path through the voxel model is computed and the intersected voxels are determined. The attenuation coefficients corresponding to these voxels are accumulated so that the ray intensity can be calculated.

B. Phantom

The phantom, also known as the physical model, used throughout this paper is a BrainWeb model described in [7, 6]. The model is a voxel-based representation of a healthy human brain created from magnetic resonance (MR) scans. It enables voxel-wise material identification, particularly the weight percent mix of different brain tissues. This data is then converted into effective X-ray attenuation values, forming the voxelized phantom used in the simulation. This process is described in [19]. For X-ray simulations of the human brain, the BrainWeb provides a three-dimensional voxelized map of a healthy brain, based on MR scans. Using it, distinctions of brain tissue types like white matter, gray matter, and cerebrospinal fluid can be made.

III. Design and Implementation

A. Modeled Components

In an X-ray imaging simulation, three key components are modeled: the X-ray source, the voxel model, and the detector. Each plays an important role in accurately simulating the imaging process.

The X-ray source serves as the origin of the rays that traverse the voxel model.

- Its position is defined as a three-dimensional vector with X, Y, and Z components, which can be transformed throughout the simulation. This transformation, consisting of both rotation and translation, allows the source to mimic the movement of a real imaging system.
- The source emits a pyramid-shaped beam composed of individual rays. In the simulation, the beam is simplified by assuming a monochromatic energy level of 75 keV, eliminating the complexities of polychromatic attenuation. This assumption is justified, as at higher energy levels, consistent attenuation behavior is maintained for soft tissues, fat, and bone [15]. However, this simplification does not account for beam hardening effects, which in real scenarios could introduce artifacts such as cupping or streaking [2]. We will not take these into account in our implementation.
- To simplify the simulation, the rays generated by the source ignore Compton scattering [9], a key phenomenon in real X-ray imaging that contributes to noise and image blurring [8]. So, while this omission makes the simulation more simplistic, it means that certain artifacts present in actual imaging systems will not appear in the generated images.

The voxel model represents the object being imaged.

- The voxel model, introduced in Section II-B, represents the subject in the imaging process. It consists of numerous voxels (3D pixels), each assigned a linear attenuation coefficient that describes how much X-ray intensity is attenuated as it passes through the voxel.
- The model is divided into layers, traversed from front to back. It can take the form of a cube or rectangular

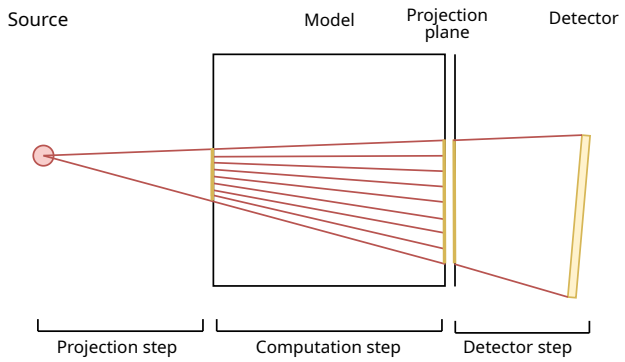


Fig. 2: The three steps of the simulation algorithm

cuboid, with the only requirement being that all layers run parallel.

- A big memory throughput reduction is achieved by compressing the attenuation coefficients of the voxels into sixteen distinct material types, each describing a different attenuation coefficient. These material types, defined as 32-bit fixed-point numbers, can be decoded within the simulation using a 4-bit index. This compression trick decreases the size of the memory transfer within the FPGA by a factor of four.

The detector is responsible for capturing X-rays that pass through the voxel model and forming the final simulated image.

- Its geometry is defined by its center point and horizontal and vertical spans, implicitly defining the four corner points in three-dimensional space. The detector model is based on the Allura Xper FD20 [16], with physical dimensions of 293.2 × 398.2 mm.
- Its resolution determines the level of detail in the final image; higher resolutions capture finer details in the model. However, each detector pixel corresponds to a single ray traced from the X-ray source, causing computational cost to scale linearly with the number of detector pixels.

B. Algorithm Steps

As shown in Figure 2, the simulation aims to accurately model the behavior of X-rays as they travel from a defined source, pass through a voxel-based model, and reach the detector [14]. To achieve this, a set of rays is generated from the source and traced through the voxel grid, with each ray contributing to the final image captured at a detector pixel. The positions of the rays are arranged in parallel rows, ensuring that each ray within a given voxel line enters at the same height. This structured alignment simplifies memory access and data transfer, as voxel data can be preloaded efficiently for each row. These parallel positions, referred to as projection pixels, are not perfectly aligned with the detector pixels. The detector pixels therefore need to be interpolated based on the values at the projection pixels. The algorithm is divided into three steps, illustrated in Figure 2.

1) Projection step: In this step the region of the voxel model that is relevant for the simulation is determined by identifying which parts of the model interact with the X-ray beam. This step involves projecting the detector onto the voxel model to establish the boundaries of the X-ray beam. The inputs to this step include the location of the source and the detector. Based on the projection, a grid of projection pixels can be created that mark the entry points of individual rays into the model.

2) Computation step: In computation step, the attenuation of the X-rays are calculated as they traverse the voxel model. This step is crucial for determining how much of the X-ray intensity is attenuated based on the material properties of the voxels. The key inputs to this step include the projection pixels, which define the rays, and the voxel model, which contains material-specific attenuation coefficients. The algorithm propagates rays through the voxel model, summing attenuation coefficients along each ray's path to compute the cumulative attenuation values. The output of this step is an attenuation map, which represents the total attenuation encountered by each ray measured at a different projection pixel.

3) Detector step: The final step determines the attenuation at the detector pixels by interpolating the computed attenuation values at the projection pixels. As a result, this step produces the detector image, which represents the final simulation output, showing the X-ray intensities measured by the detector.

C. Accelerator Architecture

The main accelerator architecture is shown in Figure 3. It consists of both a software component as well as a hardware component [1].

1) Embedded CPU: A processor is responsible for executing the projection step of the algorithm. It communicates with a host system to receive the input system configuration. It is implemented as a MicroBlaze Micro Controller System, a 32-bit RISC soft-core processor with a three-stage pipeline. Its primary tasks include constructing the projection boundary that is used in the computation step for ray generation. The processor receives simulation parameters from the host, such as source and detector positions, projection resolution, and the sixteen distinct material properties.

The processor is able to complete calculations within 2 milliseconds, allowing sufficient time for downstream components to process the data, supporting the real-time performance objectives of the system.

2) Compute engines: The simulation compute engines serve as the main computational units responsible for processing voxel data and computing interactions between rays and the voxel model. Each engine is designed to handle a specific portion of the workload, allowing for efficient distribution of computations across multiple instances. Their role is to perform intersection calculations between rays and voxels, accumulate the resulting values, and pass the data along the accelerator architecture shown in Figure 3. Using multiple compute engines improves both simulation performance and scalability.

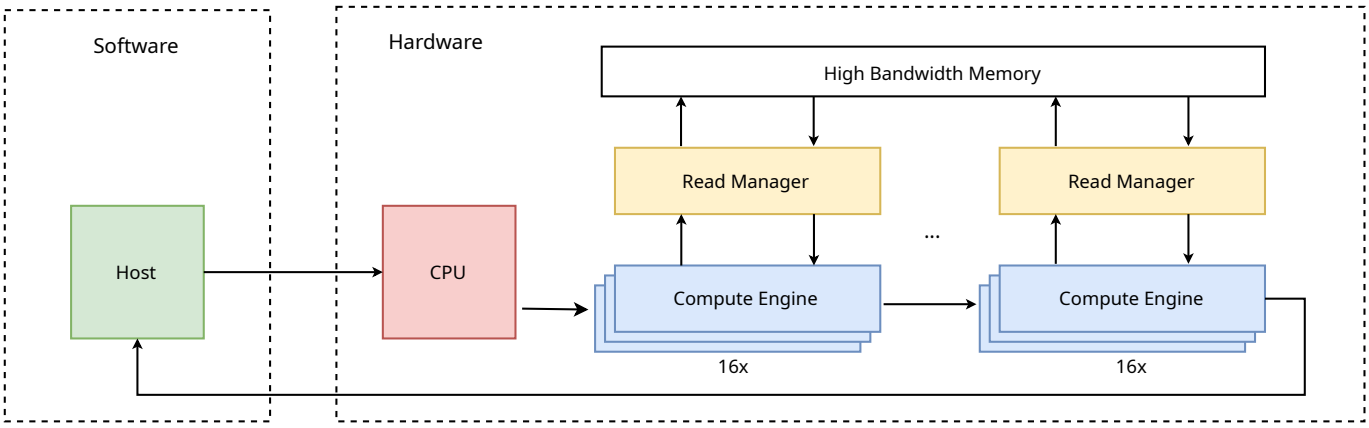


Fig. 3: Illustration of the accelerator architecture

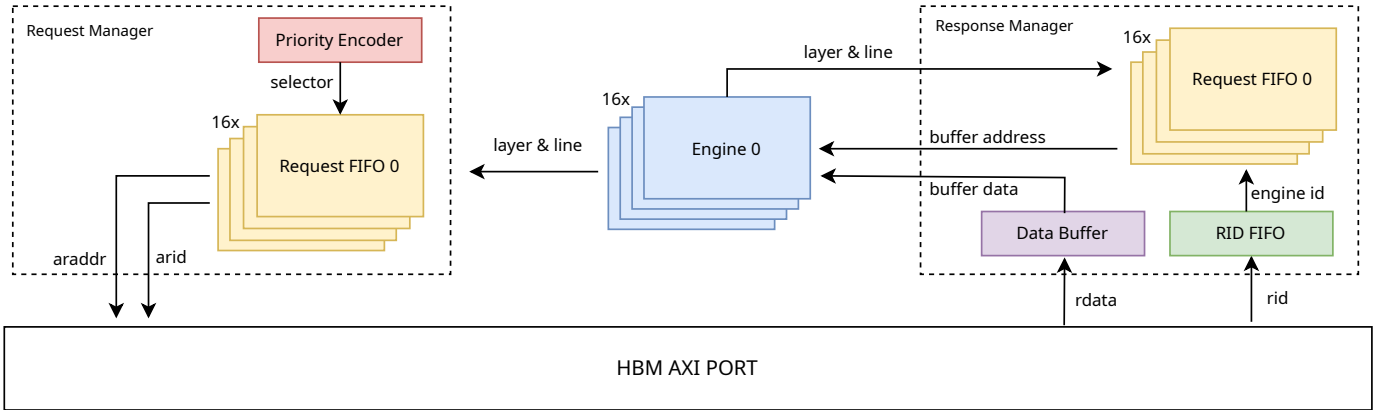


Fig. 4: Illustration of the memory architecture

- Engines fetch voxel lines in advance and store them in a circular buffer. This technique ensures that the voxel lines are present in advance, preventing bottlenecks. Since every ray at a given voxel line enters at the same height, the voxel data can be transferred efficiently in aligned rows. The engines are grouped in blocks of sixteen engines, which are attached to a single read manager that implements the interface to the physical memory. The number of groups is dependent on the size of the model.
- Digital Signal Processing (DSP) slices are used within the engine for performing high-speed arithmetic operations. These slices ensure high performance for the core calculations required to determine how rays interact with the voxel model, significantly reducing computation time.
- Each voxel in the simulation contains a material index that identifies the attenuation coefficient associated with it, as explained in Section III-A. The engines decode this material index into the voxel's attenuation value, decreasing the size of the memory transfer.
- The engines are connected in a chain, where data is passed between engines. This means that the maximum number of rays that can be processed at a time is equal to the number of engines. This improves the amount of parallelism in the implementation.

3) Host PC: The detector step is implemented on a host PC rather than hardware because it is not timing-critical and has a relatively low computational cost. This step involves mapping the computed attenuation values from the projection plane onto the detector plane. Because these planes are not always parallel, interpolation techniques are necessary to accurately assign attenuation values to detector pixels. The decision to execute the final step on the host PC in software allows post-processing techniques, such as upsampling, to further improve performance.

D. Memory Architecture

Efficient memory management is crucial to optimizing system performance, particularly when handling larger voxel models. The voxel line-based access method improves performance by ensuring that memory is accessed in a structured and predictable manner. Instead of fetching individual voxels, which would lead to inefficient memory usage and frequent stalls, the system preloads entire lines of voxels into buffers before they are needed. This approach reduces the number of memory transactions and allows data to be streamed efficiently into the processing engines. Since all rays within a voxel line enter at the same height, memory can be accessed sequentially, minimizing latency and maximizing data throughput. By or-

ganizing memory transfers around voxel lines, the simulation reduces bottlenecks, optimizes caching, and ensures a steady flow of data, ultimately leading to higher processing efficiency and faster computation times.

The custom memory architecture is built around Xilinx's HBM architecture. An illustration of the full architecture is shown in Figure 4. The HBM features a wide array of parallel ports, enabling simultaneous access to multiple data streams. This architecture significantly boosts data transfer speeds and reduces latency compared to traditional memory. The data accelerator card chosen for the implementation is the Alveo U50 Data Accelerator card. This card features two HBM stacks of 8 GB, with each stack having a reported bandwidth of 201 GB/s [3]. One stack features sixteen parallel ports.

To each of these ports, a *read manager* can be attached that acts as an interface for sixteen different engines. A read manager is responsible for efficiently retrieving voxel data for computation. It serves as an intermediary between the memory architecture and the compute engines, ensuring that data is accessed and delivered in a structured and efficient manner. At its core, the read manager consists of two primary subcomponents: the *request manager* and the *response manager*. The request manager is responsible for translating voxel data requests from the compute engines into appropriate memory addresses using the AXI4 standard. These requests are stored in First-In-First-Out (FIFO) buffers, ensuring orderly processing. To manage multiple simultaneous requests, a priority encoder is employed, following a round-robin scheduling scheme to provide fair access across all engines. The selected request is then converted into an AXI4 address and sent to the memory controller, which supports multiple outstanding requests. A unique identifier (arid) corresponding to the engine number is assigned to each request, which ensures memory responses can be identified.

The response manager handles the retrieval and processing of voxel data from HBM. Since memory returns data in bursts of 256 bits, the response manager is responsible for segmenting and distributing this data into 64-bit blocks, aligning with the compute engines' interfaces. It ensures that data is delivered to the correct engine by utilizing the read address identifier (rid), which maps each response to its corresponding request. By implementing this structured read management system, the FPGA can efficiently handle high-throughput memory requests while minimizing latency and ensuring correct data distribution. This architecture significantly enhances the performance of voxel-based computations, providing a scalable solution for memory-intensive applications. Based on a realistic access pattern a bandwidth of 12.1 GB/s per channel was measured, which is 96% of the maximum throughput per channel.

IV. Results and Analysis

In the performance evaluation of the FPGA-based simulation, significant improvements were observed compared to traditional CPU-based implementations. The system's performance is primarily influenced by factors such as the detector resolution and the model size. The number of rays simulated is a critical determinant of speed, and higher resolutions require more rays to be simulated. In some cases, memory access-induced stalls can occur. This mainly happens at lower resolutions when the frequency at which voxel lines are requested is higher. The stall duration is proportional to the number of voxels being fetched per line. Therefore, the size of the model also affects the performance of the system.

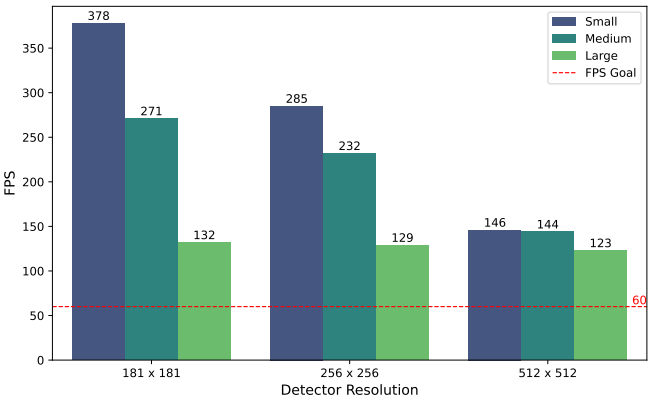
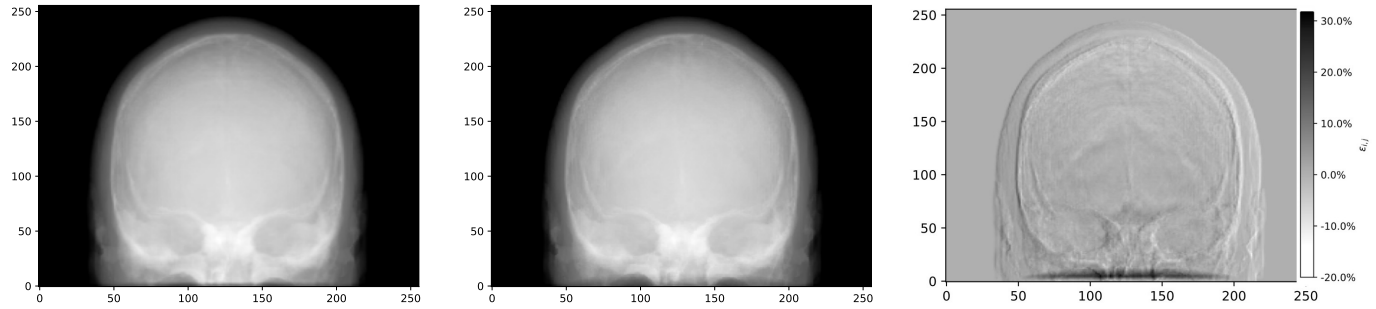


Fig. 5: FPS performance for different model and detector sizes

The frame rate achieved varies depending on the combination of model size and detector resolution. Some results are summarized in Figure 5. For a small model, which is 384 × 297 × 384 voxels, the frame rate achieved ranges between 146 FPS and 378 FPS. For a medium-sized model, with a size of 512 × 512 × 512 voxels, the performance ranges between 144 FPS to 271 FPS. The largest model, which contains 1024 × 1024 × 1024 voxels, still maintained frame rates between 123 FPS and 132 FPS. At the highest detector resolution, stalling no longer occurs for the small and medium-sized models, resulting in a similar performance. For the largest model, stalling occurs at every resolution which results in small variance in the performance. Notably, the system consistently exceeded the 60 FPS benchmark required for real-time applications, demonstrating its suitability for high-performance scenarios.

To validate the FPGA-based simulation, it was compared with a CPU-based simulation. The CPU implementation uses single-precision floating-point arithmetic instead of fixed-point and avoids errors introduced by the detector steps. Both methods used the same compressed model for fairness. Figure 6 shows a result from both approaches. The results showed near-identical outputs except for higher attenuation values at the bottom in the FPGA simulation, as confirmed by error analysis. This discrepancy likely stems from multiple factors, including ray length inaccuracies, fixed-point precision limitations, and simulation artifacts caused by the voxel model's cut-off. Elsewhere, the FPGA map appeared slightly shifted left compared to the CPU reference which is likely a result of bit truncation. Despite these errors, bilinear interpolation handled high-contrast areas well, and the FPGA result is



(a) Synthetic image produced by FPGA-based simulation

(b) Synthetic image produced by CPU-based simulation

(c) Normalized difference between CPU and FPGA

Fig. 6: Comparison between FPGA and CPU-based simulation for a voxel model

virtually indistinguishable from the CPU result. Using the CPU simulation as a reference, the image produced by the FPGA has a mean absolute error of only 2.26%.

One of the key optimizations explored in the study was the use of upsampling to improve performance without significantly compromising accuracy. By running the simulation at a lower detector resolution of 181 × 181 pixels and then upsampling the output, a 33% speed improvement was achieved compared to running the system at a regular resolution of 256 × 256 pixels. The upsampling process involved interpolating the lower-resolution output to a higher resolution using bilinear interpolation. This method effectively reduced the computational burden while having a small impact on image quality. The mean absolute error (MAE) introduced by this technique was only 0.6%, making it a viable performance enhancement. However, in high-contrast regions, the maximum observed error reached 6%. In addition, upsampling can increase noise levels in a map, which is undesirable, particularly for machine learning and CT (computed tomography) applications [21].

V. Related Work

Synthetic X-ray image generation is not a novel field, with research on rendering techniques and computational algorithms dating back to 1994. [10, 11] propose using a stream-based approach for medical image processing to reduce the memory bandwidth requirements of such algorithms. [18] describes the usage of attenuation fields that extend ray tracers, allowing most computations to be performed in a preprocessing step. In order to speed up the computations, graphics processing units (GPUs) are good candidates [4] due to the large number of processing cores available. [20, 22] achieve high GPU performance through algorithmic simplifications and specialized ray casting techniques. However, these computational approaches involving the simulation of rays moving through the model, also known as ray tracing, are computationally expensive and require a high memory bandwidth. More modern approaches advocate the usage of machine learning algorithms to aid the X-ray image generation process. The problem with these techniques is that they rely on large amounts of labeled images, which are rarely available. To address this, [5] introduces a multi-stage Generative Adversarial Network (GAN) to generate synthetic images with semantic labels for data augmentation, which performs well on small datasets. [17] attempts to generate synthetic CT images from Magnetic Resonance Imaging (MRI) data. The implementation included additional improvements such as perceptual loss, coordinated convolutional layers, and super-resolution techniques. Such machine learning algorithms can also be accelerated on FPGAs [12, 13]. However, these machine learning methods can generate unpredictable outputs, which is undesirable in mission-critical applications such as the medical domain.

VI. Conclusion

This paper presents the architecture and implementation of an FPGA-based hardware ray-tracing accelerator for generating real-time synthetic X-ray medical images. The study demonstrates that the developed system can produce realistic images with a mean absolute error of only 2.26% compared to CPU-based implementations. The separation of computational tasks between the host machine and FPGA has proven to be a good design choice, enabling real-time image generation by leveraging hardware acceleration for timing-critical operations. The proposed architecture includes a highly optimized memory architecture, particularly through the Xilinx High Bandwidth Memory, with improved access patterns to maximize data throughput to ensure that the FPGA achieves near-maximum performance.

Performance evaluation shows that the generation speed primarily depends on the detector resolution, for it directly determines the number of simulated rays. However, model size can also become a limiting factor due to stalling by increased memory transfer delays. The study further explores upsampling techniques, concluding that performance can be improved by 33% with only a minor 0.6% increase in average error compared to CPU-based implementations, making it a viable option in cases where slight accuracy reductions are acceptable.

Acknowledgment

This research was performed with the support of the Eureka Xecs project TASTI (grant no. 2022005).

References

[1] P.J. Aanhane. *Synthetic X-Ray Image Generation Using FPGA-Based Hardware Acceleration*. 2025. URL: https://resolver.tudelft.nl/uuid:19926e3a-9096-4e9c-8ccc-2181def9512b (visited on 08/30/2025).

[2] Shiras Abdurahman et al. "Beam Hardening Correction Using Cone Beam Consistency Conditions". In: *IEEE Transactions on Medical Imaging* 37.10 (2018), pp. 2266–2277. DOI: 10.1109/TMI.2018.2840343.

[3] *Alveo U50 Data Center Accelerator Card Data Sheet*. DS965. v1.2. Xilinx. Nov. 2019.

[4] P. Bhosale et al. "GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications". In: *Medical Imaging 2018: Image Processing*. Ed. by Elsa D. Angelini and Bennett A. Landman. Vol. 10574. International Society for Optics and Photonics. SPIE, 2018, 105740R. DOI: 10.1117/12.2293098. URL: https://doi.org/10.1117/12.2293098.

[5] Giorgio Ciano et al. "A Multi-Stage GAN for Multi-Organ Chest X-ray Image Generation and Segmentation". In: *Mathematics* 9.22 (2021). ISSN: 2227-7390. DOI: 10.3390/math9222896. URL: https://www.mdpi.com/2227-7390/9/22/2896.

[6] C.A. Cocosco et al. "BrainWeb: Online Interface to a 3D MRI Simulated Brain Database". In: *NeuroImage* (1997). URL: https://api.semanticscholar.org/CorpusID:14864257.

[7] D.L. Collins et al. "Design and Construction of a Realistic Digital Brain Phantom". In: *IEEE Transact. On Med. Imaging* 17.3 (1998), pp. 463–468.

[8] Haruki Hattori et al. "Learning Scatter Artifact Correction in Cone-Beam X-Ray CT Using Incomplete Projections with Beam Ho le Array". In: *Journal of Nondestructive Evaluation* 43.3 (Aug. 2024), p. 99. ISSN: 1573-4862. DOI: 10.1007/s10921-024-01113-5. URL: https://doi.org/10.1007/s10921-024-01113-5.

[9] Trang Hoang and Ayush Goel. *Compton effect*. en. Aug. 2014. DOI: 10.53347/rid-30308. URL: http://dx.doi.org/10.53347/rID-30308.

[10] J. Hoozemans et al. "Frame-based Programming, Stream-Based Processing for Medical Image Processing Applications". In: *Journal of Signal Processing Systems* 91.1 (Jan. 2019), pp. 47–59. ISSN: 1939-8115. DOI: 10.1007/s11265-018-1422-3. URL: https://doi.org/10.1007/s11265-018-1422-3.

[11] J. Hoozemans et al. "VLIW-Based FPGA Computation Fabric with Streaming Memory Hierarchy for Medical Imaging Applications". In: *Applied Reconfigurable Computing*. Ed. by Stephan Wong et al. Cham: Springer International Publishing, 2017, pp. 36–43. ISBN: 978-3-319-56258-2.

[12] M. Ji et al. "FPQNet: Fully Pipelined and Quantized CNN for Ultra-Low Latency Image Classification on FPGAs Using OpenCAPI". In: *Electronics* 12.19 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12194085. URL: https://www.mdpi.com/2079-9292/12/19/4085.

[13] M. Ji et al. "Fully Pipelined FPGA Acceleration of Binary Convolutional Neural Networks with Neural Architecture Search". In: *Journal of Circuits, Systems and Computers* 33.10 (2024), p. 2450170. URL: https://doi.org/10.1142/S0218126624501706.

[14] H.J.M.T. Knops. *Hardware acceleration of artificial X-ray image generation*. 2024. URL: https://resolver.tudelft.nl/uuid:f243012c-1380-45d9-ae98-de6211defac1 (visited on 12/24/2024).

[15] Andreas Maier et al., eds. *Medical Imaging Systems: An Introductory Guide*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018. URL: http://link.springer.com/10.1007/978-3-319-96520-8 (visited on 09/17/2024).

[16] Philips. *The Allura Xper FD20 detector*. URL: https://www.philips.nl/healthcare/product/HC722012CA/allura-xper-fd20-cardiovascular-x-ray-system%5C#features.

[17] Denis Prokopenko et al. "Unpaired Synthetic Image Generation in Radiology Using GANs". In: *Artificial Intelligence in Radiation Therapy*. Ed. by Dan Nguyen, Lei Xing, and Steve Jiang. Cham: Springer International Publishing, 2019, pp. 94–101. ISBN: 978-3-030-32486-5.

[18] Daniel B Russakoff et al. "Fast generation of digitally reconstructed radiographs using attenuation fields with application to 2D-3D image registration". en. In: *IEEE Trans Med Imaging* 24.11 (Nov. 2005), pp. 1441–1454.

[19] M. Simon et al. "Physical image simulation of human brain in case of acute stroke". In: *Medical Imaging 2020: Physics of Medical Imaging*. Ed. by Guang-Hong Chen and Hilde Bosmans. Vol. 11312. International Society for Optics and Photonics. SPIE, 2020, p. 1131234. DOI: 10.1117/12.2549551. URL: https://doi.org/10.1117/12.2549551.

[20] Jakob Spoerk et al. "High-performance GPU-based rendering for real-time, rigid 2D/3D-image registration and motion prediction in radiation oncology". In: *Zeitschrift für Medizinische Physik* 22.1 (2012), pp. 13–20. ISSN: 0939-3889. DOI: https://doi.org/10.1016/j.zemedi.2011.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0939388911000651.

[21] Wolfram Stiller. "Basics of iterative reconstruction methods in computed tomography: A vendor-independent overview". In: *European Journal of Radiology* 109 (2018), pp. 147–154. ISSN: 0720-048X. DOI: https://doi.org/10.1016/j.ejrad.2018.10.025.

[22] G.J. Tornai, G. Cserey, and I. Pappas. "Fast DRR generation for 2D to 3D registration on GPUs". In: *Med Phys* 39.8 (Aug. 2012), pp. 4795–4799.