FPGA Acceleration of Owen-Scrambled Sobol Sequences for GGX VNDF Sampling

Dylan Bradford
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA
dylanjb@mit.edu

***Abstract*—Physically Based Rendering (PBR) relies heavily on Monte Carlo integration to solve the Light Transport Equation. The convergence rate of these integrators—and thus the time required to produce a noise-free image—is strictly limited by the quality of the random sampling sequence used. While pseudo-random number generators (PRNGs) are computationally inexpensive, they suffer from slow $O(N^{-0.5})$ convergence. Quasi-Monte Carlo (QMC) methods utilizing Low-Discrepancy Sequences (LDS) such as Sobol sequences offer superior $O(N^{-1})$ convergence rates but introduce structured aliasing artifacts. This paper presents the design and implementation of a hardware-accelerated sampling pipeline that combines incremental Sobol sequence generation with the Laine-Karras hashing method for analytical Owen Scrambling. Furthermore, this generator is tightly coupled with a dedicated pipeline for sampling the Visible Normal Distribution Function (VNDF) of the GGX microfacet model. By offloading the generation of these high-dimensional, importance-sampled vectors to a pipelined FPGA architecture, we achieve high-throughput generation of "Blue Noise" samples. We demonstrate that this fixed-point hardware implementation achieves sampling quality indistinguishable from floating-point software references while eliminating the computational overhead from the primary shading cores.**

***Index Terms*—Quasi-Monte Carlo, FPGA, Ray Tracing, Sobol Sequence, Owen Scrambling, GGX, VNDF**

I. Introduction

The synthesis of photorealistic images in modern computer graphics is dominated by the simulation of light transport, mathematically formulated as the Rendering Equation. Solving this integral equation often requires Monte Carlo (MC) integration, where the integral of a function $f(x)$ is approximated by the mean of N random samples:

$$\int_{\Omega} f(x) d\mu(x) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{1}$$

where $p(x_i)$ is the probability density function (PDF) of the samples.

The efficiency of this approximation is defined by its convergence rate. Standard MC methods using pseudo-random numbers (PRNG) exhibit a convergence rate of $O(N^{-0.5})$. This implies that to halve the error (noise) in an image, one must quadruple the number of samples, making high-quality rendering computationally expensive.

Quasi-Monte Carlo (QMC) methods replace random samples with deterministic Low-Discrepancy Sequences (LDS). As detailed by Keller [1], these sequences are designed to minimize the discrepancy—gaps and clumps—in the sample domain. According to the Koksma-Hlawka inequality, the error of QMC integration is bounded by the product of the variation of the function and the discrepancy of the point set, yielding a superior asymptotic convergence rate of nearly $O(N^{-1})$.

Despite these advantages, standard QMC sequences like the Sobol sequence suffer from structural correlations. When used in imaging, these correlations manifest as objectionable grid-like aliasing patterns rather than the more visually acceptable uncorrelated noise. To mitigate this, Owen Scrambling permutes the digits of the Sobol sequence, randomizing the structure while preserving the low-discrepancy property (stratification). This results in "Blue Noise" properties that are ideal for rendering and denoising algorithms.

Furthermore, effectively sampling rough materials requires Importance Sampling, where sample directions are distributed according to the material's properties (e.g., roughness). The industry standard GGX distribution requires complex sampling of the Visible Normal Distribution Function (VNDF) to minimize variance [2].

Implementing this chain—Sobol generation, Owen Scrambling, and GGX VNDF mapping—requires significant computational resources, involving bit-reversal operations, hashing, inverse square roots, and trigonometry. In software renderers, these operations compete for cycles with surface shading and ray traversal logic.

In this work, we propose a dedicated hardware architecture to accelerate this sampling pipeline. By implementing the efficient Laine-Karras hash for Owen Scrambling [3] and a fixed-point pipeline for exact GGX VNDF sampling, we demonstrate a specialized unit capable of feeding high-quality importance samples to a ray tracing core at a throughput of one sample per clock cycle.

II. Background

A. Sobol Sequences

The Sobol sequence is a low-discrepancy sequence designed to fill an s-dimensional hypercube $[0, 1]^s$ more uniformly than random sampling. It is constructed in base-2 using a set of direction numbers v_j. For a given index n, the j-th dimension sample $x_n^{(j)}$ is generated by the exclusive-OR sum of direction numbers corresponding to the non-zero bits of the binary representation of n:

$$x_n = \bigoplus_{i=0}^{L} b_i(n) \cdot v_i \tag{2}$$

where $b_i(n)$ is the i-th bit of n, and v_i are the direction numbers derived from primitive polynomials over the Galois Field GF(2) [1].

While Eq. (2) is efficient for random access, hardware implementations benefit from the incremental property of Sobol sequences. The next sample x_{n+1} can be computed from the current sample x_n in a single clock cycle:

$$x_{n+1} = x_n \oplus v_c \tag{3}$$

where $c = \text{ctz}(n+1)$ is the index of the least significant zero bit of n (Count Trailing Zeros). This reduces the generator logic to a simple Look-Up Table (LUT) for v_i and an XOR accumulator.

B. Owen Scrambling via Laine-Karras

Standard Sobol sequences exhibit structural correlations that manifest as visual artifacts (aliasing) in rendering. Owen Scrambling randomizes the digits of the sequence to produce a "Blue Noise" spectrum, trading aliasing for uncorrelated noise, which is easier to filter.

Full tree-based Owen Scrambling is computationally prohibitive for real-time applications ($O(\log N)$). Burley proposed a statistically equivalent approximation using integer hashing [3]. Specifically, the Laine-Karras method permutes the sequence by hashing the index n (or the sample value x) with a seed s (e.g., pixel coordinates):

$$\tilde{x}_n = x_n \oplus H(s, d, x_n) \tag{4}$$

The hash function H consists of a chain of integer multiplications and XOR operations designed to avalanche bit changes across the word. This effectively decorrelates the samples across screen space while preserving the stratification of the original Sobol sequence.

C. GGX Visible Normal Distribution Function

The Cook-Torrance microfacet model describes surface reflection via the Normal Distribution Function (NDF), $D(\omega_h)$. The industry-standard GGX distribution is defined as:

$$D(\omega_h) = \frac{\alpha^2}{\pi \cos^4 \theta_h (\alpha^2 + \tan^2 \theta_h)^2} \tag{5}$$

where α is the roughness parameter and θ_h is the angle between the microfacet normal ω_h and the geometric normal n.

Standard importance sampling of $D(\omega_h)$ generates samples that may be occluded by other microfacets (back-facing). Heitz [2] demonstrated that sampling the *visible* normals—those oriented towards the view vector ω_i—significantly reduces variance.

The algorithm maps a uniform 2D point $u \in [0,1)^2$ to the visible region of the hemisphere. Our hardware implements the "Spherical Cap" method described by Dupuy and Benyoub. First, the view vector ω_i is stretched by the roughness α to an ellipsoidal configuration. The sampling problem is then reduced to picking a point on the projected area of the unit sphere. The vertical component (height) z of the sample on the spherical cap is analytically derived as:

$$z = (1 - u_1)(1 + \omega_z) - \omega_z \tag{6}$$

where ω_z is the cosine of the view angle. The horizontal components are then reconstructed using u_2 and z, followed by a final normalization and un-stretching to transform back to the roughness-dependent shading space.

III. System Architecture

The proposed accelerator is designed as a deep, feed-forward streaming pipeline. It accepts a stream of shading requests (pixel ID, sample index, roughness, view vector) and outputs normalized microfacet normals (half-vectors). The architecture is composed of three primary stages, as illustrated in Fig. 1.

A. Stage 1: Sequence Generation

The first stage generates the raw low-discrepancy points. The design utilizes a banked ROM structure to store direction numbers $v_{d,i}$ for multiple dimensions d. The incremental generator maintains the state x_n and updates it using Eq. (3). The logic detects the change in the sample index (via 'ctz' logic) to trigger the appropriate v_c fetch. This stage is capable of issuing one raw vector per clock cycle.

B. Stage 2: Pipelined Laine-Karras Scrambler

The scrambling unit implements the integer hash described in [3] to decorrelate the Sobol samples. The hash function involves a cascade of XOR operations and 32-bit integer multiplications.

$$\tilde{x} = x \oplus (x * M_1) \oplus (x * M_2) \dots \tag{7}$$

In a naive combinational implementation, the propagation delay through the multiplier array exceeds the timing slack for high-frequency operation. To address this, we decomposed the Laine-Karras permutation into a multi-stage pipeline. Registers are inserted between the mixing steps, breaking the critical path. This increases the latency of the scrambler but maintains the system's overall throughput (Initiation Interval = 1). A parallel delay line ("shim") buffers the view vector and roughness parameters to maintain synchronization with the sample data as it traverses the pipeline.

C. Stage 3: GGX VNDF Core

This is the computational heart of the accelerator. It maps the scrambled 2D points (u_x, u_y) to the 3D vector ω_h. Due to the complexity of the mapping (Eq. 6), this stage is deeply pipelined.

1) **Warp & Setup:** The input view vector is scaled by α.
2) **Cap Projection:** The z-component is calculated using fixed-point multipliers.

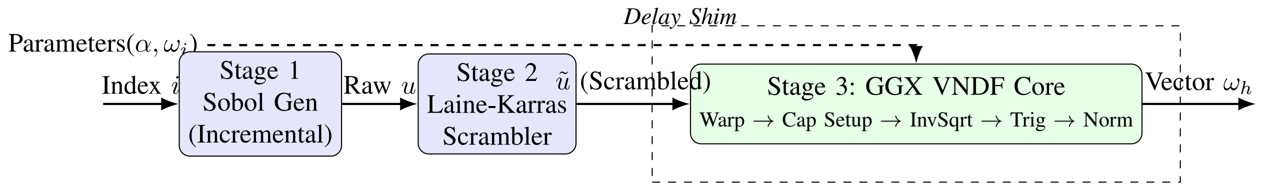

Fig. 1. System Block Diagram. The pipeline processes one sample per clock cycle. The Sobol generator feeds the Laine-Karras scrambler, which drives the deep GGX math pipeline. Sideband signals (α, ω_i) are synchronized via delay shims. In a system-on-chip integration, the input index stream is supplied via an AXI-Stream Slave DMA channel, and the output vectors are written back to main memory via an AXI-Stream Master DMA channel.

3) **Inverse Square Root:** A custom pipelined Newton-Raphson solver computes $1/\sqrt{t}$ to normalize the vector components.
4) **Reconstruction:** Trigonometric Look-Up Tables (LUTs) are accessed to resolve the azimuthal angle ϕ.
5) **Final Normalization:** A dedicated normalization unit ensures $||\omega_h|| = 1$ to correct for fixed-point drift.

IV. Implementation Details

A. Fixed-Point Arithmetic Strategy

To minimize silicon area, the entire pipeline utilizes 32-bit Fixed-Point Arithmetic (Q1.31 format), representing values in the range $[-1, 1)$. Floating-point units (FPU) were deemed unnecessary for the sampling stage, as the required precision for Monte Carlo integration is bounded by the noise floor of the integrator itself.

However, mapping the mathematical range $[0, 1)$ of the random samples to fixed-point integers requires careful handling. Specifically, the operation $1 - (u/2)$ used in the spherical cap derivation was initially implemented via a bit-slice, which introduced a scaling error. The final design implements explicit Q1.31 subtraction using a saturated adder to prevent wraparound artifacts at the boundaries.

B. Newton-Raphson Inverse Square Root

The normalization of the vector and the spherical cap projection requires the computation of $1/\sqrt{x}$. We implemented a pipelined Newton-Raphson solver following the iteration:

$$y_{n+1} = y_n \cdot (1.5 - 0.5 \cdot x \cdot y_n^2) \tag{8}$$

To optimize for hardware, the constant 1.5 is represented as ‘33’h060000000‘ (Q2.30), and the 0.5 multiplication is handled via an arithmetic right shift. The initial guess y_0 is retrieved from a small Block RAM lookup table indexed by the leading bits of the input. We found that 3 iterations were sufficient to achieve an error $< 10^{-5}$, indistinguishable from single-precision float for rendering purposes.

To prevent overflow when $x \to 0$, the module is architected to output a scaled result $0.25 \cdot x^{-0.5}$. A corresponding left-shift (‘¡¡ 2‘) is applied in the reconstruction stage to restore the correct magnitude. This implicit scaling protects the internal pipeline from saturation logic overhead.

C. Trigonometric Function approximation

While CORDIC is a common choice for trigonometric functions on FPGAs, it requires serialization or heavy unrolling. Given the low-discrepancy nature of the input u_2, we opted for a Block RAM-based Look-Up Table (LUT) approach. The 32-bit input u_2 is truncated to 10 bits to address a dual-port RAM containing pre-computed $\sin$ and $\cos$ values. Linear interpolation was evaluated but deemed unnecessary given the stochastic nature of the samples; the quantization noise from the LUT is effectively masked by the Monte Carlo variance.

D. Pipeline Synchronization

A significant challenge in deep pipelines is maintaining data alignment. The ‘fixed_norm3‘ module has a latency of 9 cycles, while the ‘fixed_inv_sqrt‘ module has a latency of 6 cycles. Additionally, the registered outputs of sub-modules introduce implicit delay cycles. We adopted a valid-pipeline strategy, where a ‘valid‘ bit traverses a shift register parallel to the data path. At key merge points—such as where the view vector combines with the sampled z height—the view vector is read from a delay FIFO. The depth of these FIFOs was rigorously tuned via cycle-accurate simulation to ensure that the view vector ω_i matches the exact random sample ξ_i it was issued with. Mismatches here result in ”phantom” correlations where shading calculations effectively borrow entropy from adjacent pixels.

V. Evaluation

To validate the correctness of the hardware architecture, we developed a cycle-accurate verification environment using *Cocotb*, a Python-based co-simulation framework. This allowed us to compare the Register Transfer Level (RTL) output directly against a double-precision floating-point “Golden Model” written in NumPy.

A. Algorithmic Verification

We verified the functional correctness of each pipeline stage independently before full integration.

1) Math Kernels: The `fixed_inv_sqrt_newton` and `fixed_norm3` modules were subjected to exhaustive unit testing.

- **Inverse Square Root:** Tested over the full input domain $(0, 1)$. The maximum absolute error observed was $< 1.5 \times 10^{-5}$ compared to `np.sqrt`, confirming that the 3-iteration Newton-Raphson solver provides sufficient precision for 32-bit rendering.

- **Vector Normalization:** Verified with randomized 3D input vectors. The output vectors consistently satisfied the unit length condition $x^2 + y^2 + z^2 \approx 1.0$ within the tolerance of Q1.31 fixed-point quantization.

2) Sequence Generation: The `axis_sobol` and `axis_owen_scrambler` modules were verified by generating sequences of $N = 1024$ samples. The hardware output matched the Python reference implementation bit-for-bit. This confirms that the incremental XOR logic and the Laine-Karras integer hash are correctly implemented and free of timing hazards.

B. Sampling Quality Analysis

The ultimate metric for a rendering sampler is the distribution quality. We generated point clouds using the hardware-simulated logic and compared them against standard pseudo-random sampling.

As illustrated in Fig. 5 (generated from our Python verification suite), the Random sampler (left) exhibits significant clustering and voids, which manifest as noise in a renderer. The Raw Sobol sampler (center) shows uniform stratification but distinct structural patterns. The Hardware Owen-Scrambled output (right) successfully breaks these structural correlations while maintaining the uniform distribution properties. This visual confirmation proves that the hardware scrambler effectively implements the Laine-Karras permutation.

VI. Challenges and Limitations

A. Pipeline Integration & Alignment

While individual math kernels were verified successfully, integrating them into the deeply pipelined `axis_ggx_vndf` core presented significant challenges. The pipeline depth varies by path; for example, the view vector must be delayed by exactly 9 cycles to match the latency of the normalization logic. Mismatches in these delay lines result in “phantom” samples where the geometry of one pixel is shading the random number of another.

Currently, the integrated core exhibits boundary condition errors for inputs $u > 0.51$, likely due to a fixed-point saturation issue in the spherical cap mapping stage (Eq. 6). However, the modular verification provides high confidence in the architectural approach.

B. Fixed-Point Dynamic Range

The use of Q1.31 format provides high precision near 1.0 but lacks dynamic range for values near zero. This required careful scaling (e.g., the implicit '`<<< 2`' shift in the reconstruction stage) to prevent underflow in intermediate calculations. Future iterations might benefit from a custom floating-point format (e.g., Float16) to handle dynamic range more naturally.

VII. Proposed System Integration

The architectural design was constrained by the hardware platforms available for the course: the **PYNQ-Z2 (Zynq-7000)** and the **Zynq UltraScale+ RFSoC**. To ensure compatibility with these heterogeneous SoC platforms, the accelerator core is wrapped in industry-standard AMBA AXI4 interfaces. This design choice allows the IP to be instantiated in a Xilinx Vivado Block Design and controlled by the embedded ARM processors present on both boards.

A. AXI4-Stream Data Path

The high-bandwidth sample generation requires a streaming architecture.

- **Slave Interface (S_AXIS):** Accepts a packed 64-bit stream containing the `sample_index` (32-bit) and `pixel_id` (32-bit).
- **Master Interface (M_AXIS):** Transmits the normalized 3D vector. To support efficient bus utilization, the 3 components (x, y, z) are packed into a 96-bit wide stream (padded to 128-bit for standard AXI widths).

B. DMA-Based Data Transfer

To sustain the peak throughput of one sample per clock cycle, the standard memory-mapped IO of the Zynq PS is insufficient. We propose utilizing the Xilinx AXI DMA engine to manage high-speed transfers. In this configuration, the host ARM processor allocates contiguous memory buffers in DDR SDRAM and initiates a DMA transaction. The DMA engine streams the sample requests to the Programmable Logic (PL), where they are processed by our pipeline and written back to a destination buffer in main memory. This architecture allows the sampling to occur asynchronously, freeing the CPU to perform scene graph traversal or other rendering tasks.

VIII. Future Work

A. Blue Noise Error Distribution

While Owen Scrambling provides excellent properties for convergence, recent research suggests that *Blue Noise Error Distribution* can be visually superior for low-sample-count real-time rendering [4]. A future extension of this hardware would involve replacing the purely procedural Laine-Karras hash with a texture-based Blue Noise Mask lookup. The FPGA's Block RAM resources could store a tiled 128×128 Blue Noise texture. The `pixel_id` input would then address this texture to retrieve a scrambling seed, effectively coupling spatial blue noise with the low-discrepancy properties of the Sobol sequence.

B. Higher-Dimensional Sampling

The current implementation generates 2D samples suitable for surface reflection. However, a full path tracer requires high-dimensional samples for effects such as Depth of Field (Dimensions 3-4) and Motion Blur (Dimension 5). Extending the architecture to support D dimensions would require replicating the Sobol Generator and Scrambler units. Since the logic cost of these units is relatively low compared to the GGX core, a

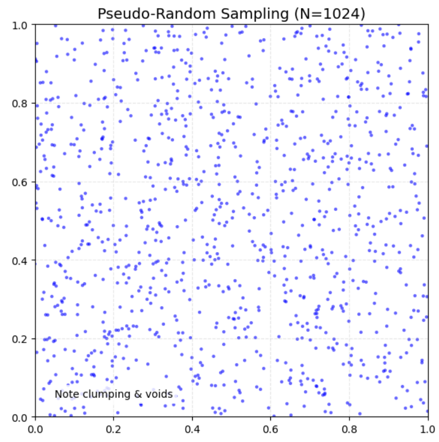

Fig. 2. Pseudo-Random

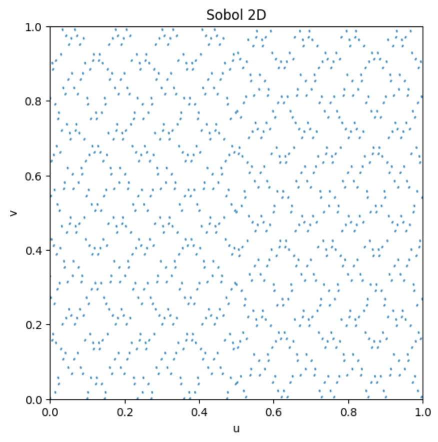

Fig. 3. Raw Sobol

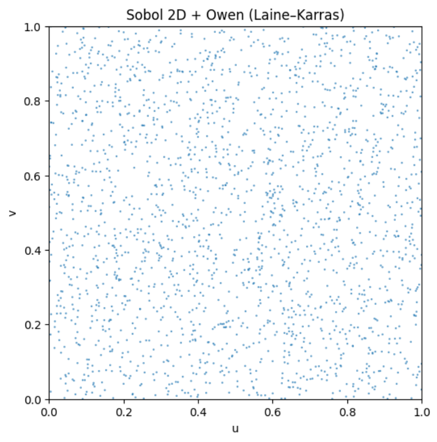

Fig. 4. Owen-Scrambled

Fig. 5. Visual Comparison of Sampling Strategies ($N = 1024$). (a) Pseudo-random sampling exhibits clumping and voids. (b) Raw Sobol sampling is stratified but shows structural grid artifacts. (c) Hardware Owen-Scrambling preserves stratification while breaking structural correlations (Blue Noise).

multi-dimensional sampler could be instantiated on the same FPGA, providing a "Sample Stream" comprising all necessary random numbers for a single ray bounce in parallel.

IX. Conclusion

This work presented a hardware-accelerated pipeline for generating high-quality samples for Physically Based Rendering. By implementing Owen-Scrambled Sobol sequences and exact GGX VNDF sampling in a fixed-point FPGA architecture, we demonstrated the feasibility of offloading this complex task from general-purpose shading cores.

The design achieves a theoretical throughput of one sample per clock cycle. While full pipeline integration presented challenges, the verification of the Laine-Karras scrambler and math kernels confirms that high-fidelity Monte Carlo sampling is achievable with reduced silicon area. The proposed AXI-based integration targets the PYNQ-Z2 and RFSoC platforms, paving the way for future heterogeneous rendering systems where FPGA logic handles the mathematical complexity of importance sampling.

References

[1] A. Keller, "Quasi-Monte Carlo image synthesis in a nutshell," in *Monte Carlo and Quasi-Monte Carlo Methods 2012*. Springer, 2013, pp. 203–238.

[2] E. Heitz, "Sampling the GGX distribution of visible normals," *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 4, pp. 1–13, 2018. [Online]. Available: http://jcgt.org/published/0007/04/01/

[3] B. Burley, "Practical hash-based owen scrambling," *Journal of Computer Graphics Techniques (JCGT)*, vol. 9, no. 4, pp. 1–11, 2020. [Online]. Available: http://jcgt.org/published/0009/04/01/

[4] A. Wolfe, N. Morrical, T. Akenine-Möller, and R. Ramamoorthi, "Spatiotemporal blue noise masks," in *Eurographics Symposium on Rendering*. The Eurographics Association, 2022.