Eurographics Symposium on Rendering (2023)
T. Ritschel and A. Weidlich (Editors)

Fast Procedural Noise By Monte Carlo Sampling

Marcos Fajardo[1] Matt Pharr[2]

[1]Shiokara–Engawa Research, Madrid, Spain [2]NVIDIA, Santa Clara, CA, USA

Abstract

Procedural noise functions are widely used in computer graphics as a way to add texture detail to surfaces and volumes. Many noise functions are based on weighted sums that can be expressed in terms of random variables, which makes it possible to compute Monte Carlo estimates of their values at lower cost. Such stochastic noise functions fit naturally into many Monte Carlo estimators already used in rendering. Leveraging the dense image-plane sampling in modern path tracing renderers, we show that stochastic evaluation allows the use of procedural noise at a fraction of its full cost with little additional error.

CCS Concepts

• ***Computing methodologies*** → ***Rendering; Texturing;***

1. Introduction

A meaningful amount of the runtime of modern path-tracing renderers is spent in pattern generation—computing the local surface or volumetric scattering properties at points in the scene [GIF*18, KCSG18, BAC*18, CFS*18]. As the geometric ray tracing component of path tracing is becoming more efficient thanks to specialized hardware [Bur20], the fraction of runtime devoted to pattern generation has correspondingly increased.

Procedural noise functions are often a large contributor to this cost. Since their invention in the eighties [Per85, Lew89], they continue to be used today to render irregular or natural-looking objects such as marble, wood, rock, clouds and smoke. Their success lies in the ease with which layers of detail can be stacked upon and combined to form rich textures with little effort.

This shading overhead is problematic in the context of path tracing, where shaders are evaluated on every single ray and sometimes on every single ray–object intersection (for opacity mapping). Rendering volumes modeled (or augmented) with procedural noise is particularly expensive since many samples are taken along each ray.

Noise functions are traditionally viewed as black boxes that return exact, analytical results. We observe that by opening up these black boxes we can derive stochastic estimators of them that are more efficient and fit naturally with the stochastic sampling machinery in Monte Carlo ray tracing renderers. We apply this idea to a range of widely-used noise functions, including Perlin noise [Per85], sparse convolution noise [Lew89], cellular noise [Wor96], Gabor noise [LLDD09], as well as to sums of noise functions used to create fractal patterns. We show that these stochastic estimators introduce negligible additional error when used with Monte Carlo path tracing, especially given the generous pixel sampling rates in offline rendering today. For noise-heavy scenes, we show as much as a 6.1× performance improvement.

2. Background and Previous Work

Stochastic evaluation has been at the heart of rendering since the pioneering work of Cook et al. [CPC84, Coo86] and Kajiya [Kaj86] who showed that Monte Carlo integration is an effective technique for the complex and discontinuous integrals that are commonly used in physically based rendering [PJH23b]. In rendering, the integrands themselves are usually comprised of deterministic functions. However, because expectation and integration are linear operators, factors of the integrand may themselves be stochastically evaluated. If both the Monte Carlo algorithm and any stochastic factors of the integrand are unbiased, then the estimate of the integral remains unbiased.

This observation has been previously applied in a number of areas in rendering to improve efficiency. It was central to Kajiya's original formulation of path tracing, where only a single light source was sampled for direct lighting at each vertex. One of the earliest uses of Monte Carlo was in matrix inversion [FL50] and Monte Carlo radiosity applies stochastic techniques to systems of linear equations [Shi92]. Other examples include the work of Szécsi et al. [SSK03], who evaluated single randomly-sampled lobes of multi-lobe BSDFs. Hofmann et al. [HHCM21] replaced trilinear interpolation of volume densities with random sampling of a single voxel value. Enderton et al. [ESSL10] and Wyman and McGuire [WM17] avoid the z-sorting step in conventional transparency algorithms through stochastic sampling. Another notable example of this idea is *many-light* sampling algorithms that sample a limited number of light sources [SWZ96, EK18]. All of these approaches increase error due to their randomness, yet they improve performance by skipping relatively unimportant work.

DOI: 10.2312/sr.20231141

In this paper we apply stochastic evaluation to procedural noise functions [Per85, Lew89, Wor96, LLDD09]. See Lagae et al. [LLC*10] for a comprehensive survey of research on noise functions. Important recent advances include further improvements to Gabor noise [TNVT19], texton noise [GLM17], and Heitz and Neyret's high-performance by-example noise [HN18]. All procedural noise functions that we are aware of are evaluated deterministically, with the exception of the work of Galerne et al., who randomly subsampled sums of many terms of by-example Gabor noise [GLLD12]. We apply stochastic evaluation to both sums of noise as well as to the underlying noise functions, consider a wider variety of noise functions, and show application of stochastic evaluation to both surface and volumetric path tracing.

2.1. A Toolbox of Stochastic Evaluation Techniques

Our algorithms are based on the basic application of a few principles from probability theory that we summarize for reference. See for example Ross [Ros19] or Pharr et al. [PJH23b] for background.

Discrete sums: A sum of values f_i in a discrete set A, $S = \sum_{i=1}^{|A|} f_i$ has the unbiased single-sample estimator

$$\langle S \rangle = \frac{f_j}{p_j}, \tag{1}$$

with j sampled according to discrete probabilities p_j. Variance is reduced when p_j approximates f_j/S.

Linear interpolation: From Equation 1, it follows that linear interpolation over the interval $[0,1]$, $lerp(f_0, f_1, t) = (1-t)f_0 + tf_1$ has the unbiased single-sample estimator

$$\langle lerp \rangle = \begin{cases} f_0, & \text{if } \xi > t \\ f_1 & \text{otherwise,} \end{cases} \tag{2}$$

where ξ is a uniform random value in $[0,1)$; note that only one of f_0 and f_1 need be evaluated.

Bilinear interpolation: In two dimensions, bilinear interpolation over $[0,1]^2$ is expressed by nested linear interpolations:

$$bilerp\left(f_{00}, f_{10}, f_{01}, f_{11}, x, y\right) = lerp\left(lerp\left(f_{00}, f_{10}, x\right), lerp\left(f_{10}, f_{11}, x\right), y\right)$$

Applying Equation 2 gives an estimator that avoids evaluating the f_i four times, and which uses two independent random samples:

$$\langle bilerp \rangle = \begin{cases} f_{00}, & \text{if } \xi_1 > x \text{ and } \xi_2 > y \\ f_{01}, & \text{if } \xi_1 > x \text{ and } \xi_2 \le y \\ f_{10}, & \text{if } \xi_1 \le x \text{ and } \xi_2 > y \\ f_{11}, & \text{if } \xi_1 \le x \text{ and } \xi_2 \le y \end{cases} \tag{3}$$

Conveniently, a single random sample ξ can be reused to sample all dimensions by remapping it to $[0,1)$ after each sampling decision [SWZ96], which is useful when ξ is well-distributed (e.g. with a blue noise spectrum [GF16] or with low discrepancy), allowing all dimensions to benefit from its distribution:

$$\langle bilerp \rangle = \begin{cases} f_{00}, & \text{if } \xi > x \text{ and } (\xi - x)/(1-x) > y \\ f_{01}, & \text{if } \xi > x \text{ and } (\xi - x)/(1-x) \le y \\ f_{10}, & \text{if } \xi \le x \text{ and } \xi/x > y \\ f_{11}, & \text{if } \xi \le x \text{ and } \xi/x \le y \end{cases} \tag{4}$$

Arbitrary d-dimensional interpolation is approached similarly by repeated nesting of linear interpolations. The number of evaluations of f_i is thus reduced by a factor of 2^d.

Russian roulette: Given a function $\hat{f}$ that approximates another function f, an unbiased estimator of f is given by:

$$\langle f \rangle = \begin{cases} \hat{f}(x) & \xi < p \\ \frac{f(x) - p\hat{f}(x)}{1-p} & \text{otherwise,} \end{cases} \tag{5}$$

where p is the probability of evaluating the approximation.

Transformation of noise values: Values from noise functions are often transformed in shaders by additional functions. However, applying a convex function g to a random variable X does not preserve expectation: $\langle g(X) \rangle \neq g(\langle X \rangle)$. This is a consequence of *Jensen's inequality*; its significance was first brought to the attention of the graphics community by Raab et al. [RSK08] for transmittance estimation. In the context of our work, this implies that any stochastic values that have a non-affine effect on the function being integrated will introduce bias, though if g is well-approximated by a Taylor expansion, an efficient unbiased estimator of $g(X)$ can be derived [Boo07, MBGJ22].

3. Perlin Noise

The most popular form of procedural noise was introduced to graphics by Perlin [Per85]. Perlin noise is a smooth d-dimensional interpolation of pseudo-random, isotropic gradient values g_i at the integer lattice points surrounding the shading point p. The gradients are calculated via repeated permutation and hashing of values stored in a small table. Following Perlin's improved noise reference implementation [Per02], two-dimensional noise has the form

$$Perlin(p) = bilerp(g_{00}, g_{10}, g_{01}, g_{11}, fade(\{p_x\}), fade(\{p_y\})) \tag{6}$$

where $fade(x)$ is a smooth quintic curve, and $\{x\}$ is the fractional part of x. The implementation of this analytical function requires the evaluation of gradients at 2^d lattice corners, as well as $2^d - 1$ linear interpolations.

One way to derive a stochastic form of Perlin noise is to apply Equation 4, which gives an algorithm with no linear interpolations and a single gradient evaluation, using a single random number ξ; see Algorithm 1. The same mechanism can be applied to 3D and 4D noise, where the number of gradient evaluations is reduced by $8\times$ and $16\times$ respectively, while still using a single ξ.

The top row of Figure 1 compares stochastic evaluation of Perlin noise to a reference image; even at low sampling rates the estimator converges quickly. Note that variance is lower close to lattice points since the contribution of a single gradient that is likely to be sampled dominates the final result there.

Simplex Noise. Perlin [Per01] later introduced a lower-cost procedural noise that replaces the regular lattice with a more isotropic simplicial one. This can be stochastically sampled using the same approach; its $2^d - 1$ linear interpolations and $d+1$ gradients would then reduce to no linear interpolations and a single gradient.

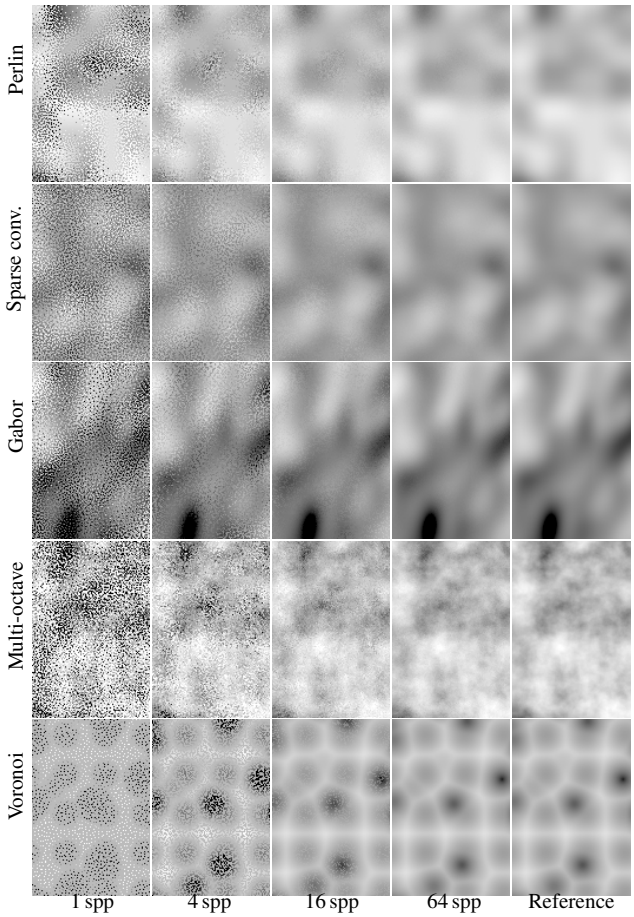


Figure 1: *2D slices of various 3D noise functions, stochastically evaluated with increasing samples per pixel (spp). At even moderate sampling rates, results are very close to the reference images.*

4. Sparse Convolution Noise

Lewis introduced sparse convolution noise to avoid the structured artifacts in lattice-based interpolation [Lew89]. A kernel function k of finite support (e.g., a quintic) is convolved with a set of point impulses of random weights w_i distributed at random 3D locations p_i. Lewis provided an efficient algorithm to procedurally generate these p_i and w_i by associating them with a grid of unit cell size equal to the kernel radius where w_i and p_i inside each cell are generated on the fly using a random number generator that is seeded based on the cell's coordinates. Given this grid, only the block of 27 cells surrounding the point of interest p has to be considered. The density of points n relative to the kernel radius controls the quality of the noise. (Typically $n \leq 10$.) The noise function is then

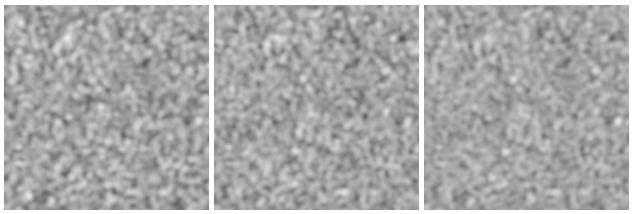
$$SparseConv(p,n) = \frac{1}{n}\sum_{c=1}^{27}\sum_{i=1}^{n} w_{i,c}\,k(|p-p_{i,c}|), \tag{7}$$

where $p_{i,c}$ is the ith point in the cth cell.

Using Equation 1, we can uniformly sample either of the sums, giving respective reductions of $n\times$ and $27\times$ in computation, or we can sample a single term from both sums, giving a savings of $27n\times$.

ALGORITHM 1: Stochastic 2D Perlin Noise

```
input: position (x,y) and a uniform random ξ ∈ [0,1)
x',y' ← x − ⌊x⌋, y − ⌊y⌋;
f_x, f_y ← fade(x'), fade(y');
if ξ > f_x then
    ξ ← (ξ − f_x)/(1 − f_x);
    if ξ > f_y then
        return grad(⌊x⌋, ⌊y⌋, x', y');
    else
        return grad(⌊x⌋, ⌊y⌋ + 1, x', y' − 1);
    end
else
    ξ ← ξ/f_x;
    if ξ > f_y then
        return grad(⌊x⌋ + 1, ⌊y⌋, x' − 1, y');
    else
        return grad(⌊x⌋ + 1, ⌊y⌋ + 1, x' − 1, y' − 1);
    end
end
```



(a) *Random* w_i **(b)** *Stratified* w_i **(c)** *Constant* w_i

Figure 2: *Our modified sparse convolution algorithm (b) retains the overall visual properties of Lewis' original algorithm (a).*

Of course, the more aggressively the sums are sampled, the greater the variance.

Importance sampling. We can design a better estimator by sampling points according to their weights. However, because the weights are computed on demand, all random weights in a cell must be generated to sample from them. The p_i are random enough already, so a more efficient approach is to slightly modify Lewis' original algorithm to use deterministic, implicitly sorted weights $w_i = (i+1/2)/n$, giving a discrete PDF $p(i) = w_i/2$ that can be importance sampled directly with $i = \lceil n\sqrt{\xi}\rceil$. The estimator is then

$$\langle SparseConv(p,n)\rangle = \frac{1}{2}\sum_{c=1}^{27} k(|p-p_{i,c}|). \tag{8}$$

See the second row of Figure 1 for images of this estimator. These *stratified* weights seem to also improve the noise signal compared to Lewis' original algorithm, as there is less sporadic clumping. Our stratified w_i sit somewhere between Lewis' random w_i and Tavernier et al.'s constant w_i [TNVT19] in terms of clumpiness as shown in Figure 2.

Gabor Noise. Lagae et al. [LLDD09] introduced a sparse convolution noise with a more complex kernel around each point that allows anisotropy and better control of frequency content, and n_c Poisson-distributed points per cell with parameter λ. We use strat-

ified weights $w_{i,c} = (i + 1/2)/n_c$ and sample a single term of each inner sum using $i = \lceil n_c \sqrt{\xi} \rceil$. See Figure 1 for images.

5. Multi-octave Noise

Perlin also showed that summing multiple octaves of a noise function, each with increasing frequency but lower contribution, leads to fractal patterns similar in appearance to fractional Brownian motion (fBm) [Per85]. For n octaves and lacunarity $1/w$, where for ease of notation we use $w = 1/lacunarity$,

$$Fractal(p,n,w) = \sum_{i=1}^{n} w^{i-1} Noise\left(\frac{p}{w^{i-1}}\right). \tag{9}$$

Applying Equation 1 makes it possible to compute an estimate at the cost of a single octave, regardless of the number of octaves requested. Thus, $n-1$ evaluations of the underlying noise function are saved. Furthermore, the noise function used may itself be stochastic.

Importance sampling. The weights w^{i-1} are a truncated geometric series, so the probability for sampling each term proportional to its weight is

$$p_i = w^{i-1} \frac{1-w}{1-w^{n+1}}. \tag{10}$$

The corresponding cumulative distribution can be inverted, allowing for direct importance sampling with a uniform sample ξ:

$$i = \left\lfloor \frac{\log(1 + \xi(w^{n+1} - 1))}{\log w} \right\rfloor. \tag{11}$$

However, even with an optimized logarithm function, we found that it is more efficient to sample by linearly searching the CDF, incrementally computing its values. Factors of w^{i-1} and the p_i cancel in the final estimator, leading to Algorithm 2. See the fourth row of Figure 1 for images.

ALGORITHM 2: Stochastic 2D Multi-Octave Fractal Noise

```
input: position (x, y), inverse lacunarity w, uniform ξ ∈ [0, 1)
p ← (1 − w)/(1 − w^{n+1});
while ξ > p do
    ξ ← ξ − p;
    p ← p × w;
    x, y ← x/w, y/w;
end
return Noise(x, y) × (1 − w^{n+1})/(1 − w);
```

6. Voronoi Noise

Worley introduced the concept of cellular textures, sometimes called Voronoi textures [Wor96]. We follow the smooth Voronoi implementation by Quilez that computes the distance to procedurally-generated points in a $3 \times 3 \times 3$ grid around the evaluation point [Qui10]:

$$SmoothVoronoi(p) = -\frac{1}{32} \log\left(\sum_{x=-1}^{1} \sum_{y=-1}^{1} \sum_{z=-1}^{1} e^{-32|p - p_{x,y,z}|} \right), \tag{12}$$

where $p_{x,y,z}$ is the random point in a cell. Applying Equation 1 to sample a single summand gives a biased estimator, since the logarithm is nonlinear. Debiasing with a Taylor expansion is not effective since the logarithm is not well-approximated by a polynomial.

In this case, the estimator in Equation 5 is effective; as an approximation to the noise function, we randomly select a single adjacent cell (x,y,z), compute the point's distance to the cell's sample point, and clamp it to the average of the noise function—approximately 0.5. Likely-accurate distance estimates are kept, while likely-inaccurate ones are set to the average. We find slightly lower error by randomly selecting $p_{x,y,z}$ from just the $2 \times 2 \times 2$ cube of cells closest to p; see Algorithm 3. Images of the estimator are shown in Figure 1.

ALGORITHM 3: Stochastic 3D Voronoi Noise

```
input: position p, probability q, uniform random ξ ∈ [0, 1)
x, y, z ← ⌊p_x⌋, ⌊p_y⌋, ⌊p_z⌋;
if p_x − ⌊p_x⌋ < 0.5 then x ← x − 1;
if p_y − ⌊p_y⌋ < 0.5 then y ← y − 1;
if p_z − ⌊p_z⌋ < 0.5 then z ← z − 1;
bits ← ⌊8 × ξ⌋;
ξ ← 8 × ξ − bits;
x, y, z ← x + (bits&1), y + (bits ≫ 1)&1, z + (bits ≫ 2)&1;
approx ← min(0.5, ‖p − p_{x,y,z}‖);
if ξ < q then
    return approx;
else
    n ← SmoothVoronoi(p);
    return (n − q × approx)/(1 − q);
end
```

7. Results

We have implemented our stochastic noise evaluation algorithms in *pbrt-v4* [PJH23a]. Random samples are generated using *pbrt*'s sampling classes, allowing both stratification and blue noise dithering [GF16]. Our implementation, evaluation code, and test scenes are all included in the paper's supplemental material and will be made publicly available.

All measurements were taken using an AMD 3970X CPU and an NVIDIA RTX 4090 GPU. All images are rendered at 1080p resolution. The multi-octave fractal noise functions summed 10 octaves of noise with $lacunarity = 2$, we set $n = 10$ for sparse convolution noise and $q = 0.875$ for stochastic Voronoi noise. Neither the reference noise functions nor our stochastic variants have seen low-level optimization (e.g., explicit SIMD instructions on CPU); we expect that both would benefit from such optimizations roughly equally.

Table 1 lists the runtime required to evaluate various noise functions, measured by averaging the time for one billion evaluations. All stochastic noise algorithms are faster than the deterministic ones, with the greatest performance benefits for the most expensive noise functions.

7.1. Contemporary Bathroom Scene

We evaluated stochastic noise for texturing of surface albedos using the *Contemporary Bathroom* scene from the *pbrt-v4* scenes distri-

	Deterministic	Stoch.	Speedup
Perlin	7.36 ns	3.84 ns	1.92×
Perlin (fractal)	71.85 ns	5.85 ns	12.28×
Sparse convolution	449.00 ns	41.21 ns	10.89×
Sparse conv. (fractal)	3311.53 ns	39.00 ns	84.90×
Gabor (Poisson)	814.26 ns	135.72 ns	6.00×
Voronoi	28.56 ns	7.02 ns	4.07×

Table 1: *Runtime for a single evaluation of each noise function in 3D on an NVIDIA RTX 4090 GPU.*

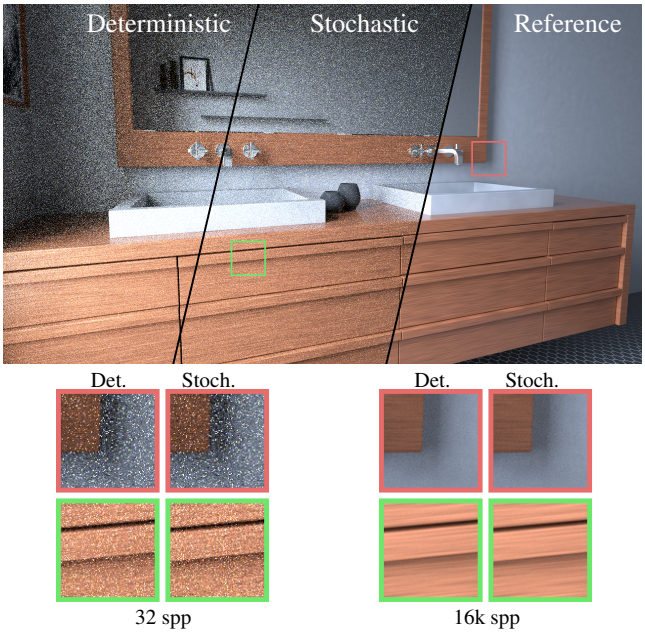


Figure 3: *The* Contemporary Bathroom *scene, rendered with path tracing. At 32 samples per pixel (left and middle slices), the error due to stochastic evaluation of noise functions is not visually evident, since the error from sampling the indirect lighting is much higher. At 16k spp, which is necessary to sample the indirect lighting well, there is no visual harm from stochastic procedural noise.*

bution, modified to use noise heavily, with multi-octave sparse convolution noise for the wood grain pattern on the cabinets, Gabor noise adding mottling to the walls, and multi-octave Perlin noise adding variation to the sinks' base layer; see Figure 3. Even at low sampling rates, error from using stochastic noise functions is not evident. Numerical measurement of error confirms these visual results; see Table 2, which includes error measured with the ꟻLIP metric [ANSA21], mean relative squared error (MRSE) and runtime. Stochastic evaluation only slightly increases error, with a substantial performance benefit.

7.2. Cloud Scene

We also evaluated stochastic noise functions for volumetric rendering, using two versions of a procedurally-defined cloud. The first, shown in Figure 4, defines the cloud's density using 10 octaves of sparse convolution noise, squaring the noise value, and further

Noise	ꟻLIP	MRSE	CPU	GPU
Deterministic	0.396	2.640	41.20s	1.51s
Stochastic	0.400	2.663	17.40s	1.16s

Table 2: *Error and rendering time for the* Contemporary Bathroom *scene with 32 spp. Stochastic evaluation of noise functions gives a* 2.37× *speedup on the CPU and a* 1.30× *speedup on the GPU.*

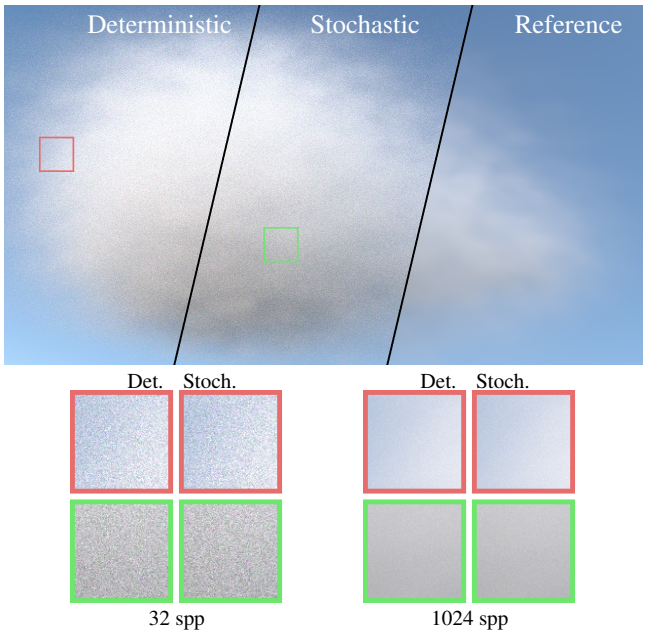
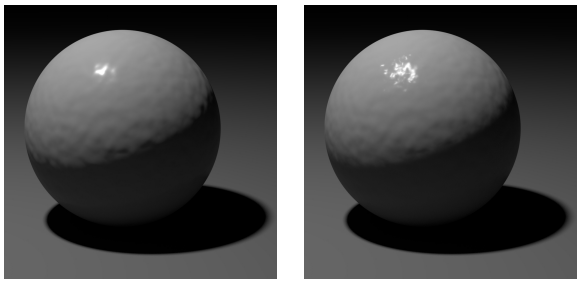


Figure 4: *Procedural cloud model defined using 10 octaves of sparse convolution noise. This image is rendered using volumetric path tracing and compares traditional sparse convolution noise to our more efficient stochastic estimator of it. GPU rendering is up to* 5.9× *faster with our estimator, without an increase in error.*

modulating the density based on height and proximity to a number of points that define the cloud's general shape. Because the square of a random variable is given by the product of two independent realizations of it, using a stochastic noise function requires evaluating it twice with independent $[0,1]$ samples, doubling the cost. (This can also be seen as a simple example of debiasing.) In order to evaluate our approach in the context of less expensive noise functions, a second version of this scene instead uses 10 octaves of Perlin noise. (Images are included in the supplemental material.)

Integrator. The underlying volumetric path tracer is based on delta tracking with null scattering [KHLN17, MGJ19]. The cloud's density is used to scale the underlying absorption and scattering coefficients, and because those coefficients are factors of terms of the volumetric light transport equation, using stochastic (and even negative [SGM*17]) values for them does not introduce bias. For transmittance algorithms like ratio tracking [NSJ14] and power-series estimators [GMH*19], those coefficients are used as multiplicative factors and stochastic coefficients also do not introduce bias. (This is unfortunately not so for Kettunen et al.'s estimator [KdPN21].)

Majorants. However, delta tracking requires a majorant for sam-



(a) *Deterministic* **(b)** *Stochastic (Equation 8)*

Figure 5: *Plastic sphere bump-mapped with sparse convolution noise. Stochastic evaluation biases the distribution of normals.*

pling the volume; if the majorant does not bound the true maximum density, variance may be high [KHLN17]. Because our stochastic noise estimators span a wider range of values than the true noise functions, they require a larger majorant. In turn, smaller steps are taken in the volume, requiring more noise function evaluations. Nevertheless, we have found that even with majorants set to bound the noise function, stochastic noise improves rendering performance by as much as 3.8×; see the top half of Table 3.

Non-bounding majorants can further improve performance if they are chosen with care. We found optimized majorants for each noise function variant via a parameter sweep, choosing the smallest majorant that left MRSE unchanged. Interestingly, we found that for the deterministic noise functions, the majorants could only be reduced by 65–80% before variance increased rapidly, but majorants could be much lower for the stochastic noise functions—as low as 40% of the true majorants. Consequently, there was a greater performance benefit for the stochastic noise functions—as high as 6.14× for sparse convolution noise; see the bottom half of Table 3.

8. Discussion and Future Work

We have shown that replacing procedural noise functions with stochastic estimators improves the efficiency of Monte Carlo path tracing, where the error from sampling indirect lighting is much higher than that from our estimators. Beyond the specific noise functions examined here, the underlying techniques are directly applicable to other functions such as texton noise [GLM17], phasor noise [TEZ*19], and high-performance noise by-example [HN18].

If a stochastic noise function does not make an affine contribution to the Monte Carlo estimator in which it is being used, it will introduce bias in the final result unless a debiasing technique is used [MBGJ22]. This limits the settings where they can be used—notably, they cannot be used to modulate surface roughness or for bump or displacement mapping, as seen in Figure 5. However, a shader compiler [GSKC10] could transparently switch to stochastic evaluation when appropiate on a case by case basis. Debiasing Monte Carlo estimators and providing bounds on the Jensen gap is an active area of research, including in machine learning, physics and finance.

We have not evaluated our stochastic estimators with post-rendering denoising algorithms. While their error is minimal, it is nevertheless present in the albedo images that are often taken as input to denoisers, which generally assume an accurate albedo.

There are ample opportunities to develop efficient stochastic variations of other functions used in rendering. For example, textures are often organized in graphs [Coo84] that could be traversed stochastically, greatly reducing the number of textures evaluated. Stochastic evaluation may also allow the development of new noise functions where no closed-form representation is available.

Acknowledgements

Thanks to the anonymous reviewers for their constructive criticism, and to Iliyan Georgiev and Chris Wyman for feedback. Thanks also to Aaron Lefohn and NVIDIA for supporting this research, and to Autodesk for providing software licenses.

References

[ANSA21] ANDERSSON P., NILSSON J., SHIRLEY P., AKENINE-MÖLLER T.: Visualizing the Error in Rendered High Dynamic Range Images. In *Eurographics Short Papers* (May 2021). doi:10.2312/egs.20211015. 5

[BAC*18] BURLEY B., ADLER D., CHIANG M. J.-Y., DRISKILL H., HABEL R., KELLY P., KUTZ P., LI Y. K., TEECE D.: The design and evolution of Disney's Hyperion renderer. *ACM Transactions on Graphics 37*, 3 (July 2018), 33:1–33:22. doi:10/gfjm8w. 1

[Boo07] BOOTH T. E.: Unbiased Monte Carlo estimation of the reciprocal of an integral. *Nuclear Science and Engineering 156*, 3 (July 2007), 403–407. doi:10.13182/NSE07-A2707. 2

[Bur20] BURGESS J.: RTX on—The NVIDIA Turing GPU. *IEEE Micro 40*, 2 (2020), 36–44. doi:10.1109/MM.2020.2971677. 1

[CFS*18] CHRISTENSEN P., FONG J., SHADE J., WOOTEN W., SCHUBERT B., KENSLER A., FRIEDMAN S., KILPATRICK C., RAMSHAW C., BANNISTER M., RAYNER B., BROUILLAT J., LIANI M.: RenderMan: An advanced path-tracing architecture for movie rendering. *ACM Transactions on Graphics 37*, 3 (Aug. 2018), 30:1–30:21. doi:10/gfznbs. 1

[Coo84] COOK R. L.: Shade trees. *Computer Graphics (Proceedings of SIGGRAPH) 18*, 3 (1984), 223–231. 6

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (Jan. 1986). doi:10/cqwhcc. 1

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH) 18*, 3 (July 1984), 137–145. doi:10.1145/964965.808590. 1

[EK18] ESTEVEZ A. C., KULLA C.: Importance sampling of many lights with adaptive tree splitting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (Aug. 2018), 25:1–25:17. doi:10/ggh89v. 1

[ESSL10] ENDERTON E., SINTORN E., SHIRLEY P., LUEBKE D.: Stochastic transparency. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010). 1

[FL50] FORSYTHE G. E., LEIBLER R. A.: Matrix inversion by a Monte Carlo method. *Mathematics of Computation 4*, 31 (1950), 127–129. 1

[GF16] GEORGIEV I., FAJARDO M.: Blue-noise dithered sampling. In *ACM SIGGRAPH Talks* (2016), ACM Press, pp. 35:1–35:1. doi:10/gfznbx. 2, 4

[GIF*18] GEORGIEV I., IZE T., FARNSWORTH M., MONTOYA-VOZMEDIANO R., KING A., LOMMEL B. V., JIMENEZ A., ANSON O., OGAKI S., JOHNSTON E., HERUBEL A., RUSSELL D., SERVANT F., FAJARDO M.: Arnold: A brute-force production path tracer. *ACM Transactions on Graphics 37*, 3 (Aug. 2018). doi:10/gfznb2. 1

Noise	Majorant	ꟻLIP	MRSE	CPU	Speedup	GPU	Speedup
Sparse conv., deterministic	Strict	0.075	0.0281	1111.00s		49.15s	
Sparse conv., stochastic	Strict	0.075	0.0276	288.50s	3.85×	12.72s	3.86×
Perlin, deterministic	Strict	0.082	0.0341	150.10s		4.35s	
Perlin, stochastic	Strict	0.082	0.0338	55.80s	2.69×	3.48s	1.25×
Sparse conv., deterministic	Optimized	0.076	0.0290	720.10s		33.78s	
Sparse conv., stochastic	Optimized	0.075	0.0298	117.20s	6.14×	5.73s	5.90×
Perlin, deterministic	Optimized	0.083	0.0345	126.90s		3.83s	
Perlin, stochastic	Optimized	0.082	0.0343	44.00s	2.88×	2.99s	1.28×

Table 3: *Comparison of error and rendering time for the cloud scene in Figure 4. All images were rendered at 1080p resolution with 32 spp.*

[GLLD12] GALERNE B., LAGAE A., LEFEBVRE S., DRETTAKIS G.: Gabor noise by example. *ACM Trans. Graph. 31*, 4 (July 2012). doi:10.1145/2185520.2185569. 2

[GLM17] GALERNE B., LECLAIRE A., MOISAN L.: Texton Noise. *Computer Graphics Forum* (Jan. 2017). 2, 6

[GMH*19] GEORGIEV I., MISSO Z., HACHISUKA T., NOWROUZEZAHRAI D., KŘIVÁNEK J., JAROSZ W.: Integral formulations of volumetric transmittance. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Nov. 2019), 154:1–154:17. doi:10/dffn. 5

[GSKC10] GRITZ L., STEIN C., KULLA C., CONTY A.: Open Shading Language. SIGGRAPH 2010 Talks, July 2010. 6

[HHCM21] HOFMANN N., HASSELGREN J., CLARBERG P., MUNKBERG J.: Interactive path tracing and reconstruction of sparse volumes. *Proc. ACM Comput. Graph. Interact. Tech. 4*, 1 (Apr. 2021). doi:10.1145/3451256. 1

[HN18] HEITZ E., NEYRET F.: High-performance by-example noise using a histogram-preserving blending operator. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 2 (Aug. 2018). doi:10.1145/3233304. 2, 6

[Kaj86] KAJIYA J. T.: The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH) 20*, 4 (Aug. 1986), 143–150. doi:10/cvf53j. 1

[KCSG18] KULLA C., CONTY A., STEIN C., GRITZ L.: Sony Pictures Imageworks Arnold. *ACM Transactions on Graphics 37*, 3 (Aug. 2018), 29:1–29:18. doi:10/gfjkn7. 1

[KdPN21] KETTUNEN M., D'EON E., PANTALEONI J., NOVAK J.: An unbiased ray-marching transmittance estimator. arXiv:2102.10294. 5

[KHLN17] KUTZ P., HABEL R., LI Y. K., NOVÁK J.: Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 36*, 4 (July 2017), 111:1–111:16. doi:10/gbxjxg. 5, 6

[Lew89] LEWIS J. P.: Algorithms for solid noise synthesis. *Computer Graphics (Proceedings of SIGGRAPH) 23*, 3 (July 1989). 1, 2, 3

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., S.EBERT D., LEWIS J. P., PERLIN K., ZWICKER M.: A survey of procedural noise functions. *Computer Graphics Forum 29*, 8 (2010), 2579–2600. 2

[LLDD09] LAGAE A., LEFEBVRE S., DRETTAKIS G., DUTRÉ P.: Procedural noise using sparse Gabor convolution. In *ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), SIGGRAPH '09, Association for Computing Machinery. doi:10.1145/1576246.1531360. 1, 2, 3

[MBGJ22] MISSO Z., BITTERLI B., GEORGIEV I., JAROSZ W.: Unbiased and consistent rendering using biased estimators. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 41*, 4 (July 2022), 48:1–48:13. doi:10/gqjn66. 2, 6

[MGJ19] MILLER B., GEORGIEV I., JAROSZ W.: A null-scattering path integral formulation of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 38*, 4 (July 2019), 44:1–44:13. doi:10/gf6rzb. 5

[NSJ14] NOVÁK J., SELLE A., JAROSZ W.: Residual ratio tracking for estimating attenuation in participating media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 33*, 6 (Nov. 2014), 179:1–179:11. doi:10/f6r2nq. 5

[Per85] PERLIN K.: An image synthesizer. *Computer Graphics (Proceedings of SIGGRAPH) 19*, 3 (July 1985), 287–296. doi:10/bbsdxj. 1, 2, 4

[Per01] PERLIN K.: Noise hardware. In *Realtime Shading*, ACM SIGGRAPH Courses. 2001. 2

[Per02] PERLIN K.: Improving noise. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 21*, 3 (July 2002), 681–682. doi:10.1145/566654.566636. 2

[PJH23a] PHARR M., JAKOB W., HUMPHREYS G.: pbrt-v4 rendering system, 2023. URL: https://github.com/mmp/pbrt-v4. 4

[PJH23b] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*, 4th ed. MIT Press, Cambridge, MA, 2023. 1, 2

[Qui10] QUILEZ I.: Smooth voronoi, 2010. URL: https://iquilezles.org/articles/smoothvoronoi. 4

[Ros19] ROSS S. M.: *A First Course in Probability*, tenth ed. Pearson, 2019. 2

[RSK08] RAAB M., SEIBERT D., KELLER A.: Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods*, Keller A., Heinrich S., Niederreiter H., (Eds.). Springer-Verlag, 2008, pp. 591–605. doi:10.1007/978-3-540-74496-2_35. 2

[SGM*17] SZIRMAY-KALOS L., GEORGIEV I., MAGDICS M., MOLNÁR B., LÉGRÁDY D.: Unbiased estimators to render procedurally generated inhomogeneous participating media. *Computer Graphics Forum (Proceedings of Eurographics) 36*, 2 (2017). 5

[Shi92] SHIRLEY P.: Time complexity of Monte Carlo radiosity. *Computers & Graphics 16*, 1 (1992), 117–120. doi:10/bkkjq9. 1

[SSK03] SZÉCSI L., SZIRMAY-KALOS L., KELEMEN C.: Variance reduction for Russian-roulette. *Journal of the World Society for Computer Graphics (WSCG) 11*, 1–3 (2003). 1

[SWZ96] SHIRLEY P., WANG C., ZIMMERMAN K.: Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (Jan. 1996), 1–36. doi:10/ddgbgg. 1, 2

[TEZ*19] TRICARD T., EFREMOV S., ZANNI C., NEYRET F., MARTÍNEZ J., LEFEBVRE S.: Procedural phasor noise. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 38*, 4 (July 2019), 57:1–57:13. doi:10/ggfgzm. 6

[TNVT19] TAVERNIER V., NEYRET F., VERGNE R., THOLLOT J.: Making Gabor noise fast and normalized. *Proceedings of Eurographics Short Papers* (2019). doi:10.2312/egs.20191009. 2, 3

[WM17] WYMAN C., MCGUIRE M.: Hashed alpha testing. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2017), pp. 7:1–7:9. 1

[Wor96] WORLEY S.: A cellular texture basis function. *Annual Conference Series (Proceedings of SIGGRAPH)* (1996), 291–294. doi:10.1145/237170.237267. 1, 2, 4