Spatiotemporal Blue Noise Masks Supplemental

1 INTRODUCTION

While the main paper is self-contained, this supplementary document details a number of additional discussions, results and extensions for the interested reader. Section 2 includes additional discussion, and contains background information about using blue noise masks in real-time rendering. Section 3 contains results beyond those shown in the main paper. Section 4 shows some best practices and simple extensions for using spatiotemporal blue noise. Finally, the appendix provides a theoretical frequency analysis of integration and inversion with blue noise offsets.

2 DISCUSSION

2.1 Masks vs. Samples

There have been many advancements in blue noise sampling in recent years, such as generating high quality sample points more quickly [4] as well as generating sample points with other desirable properties [12]. However, there has been little advancement in blue noise masks, apart from blue-noise dithered sampling which makes vector valued blue noise masks [8]. Due to this, there can be confusion and ambiguity when talking about blue noise without qualifying which is being referred to. Our paper focuses on blue noise masks.

Blue noise samples take integer indices as input and return N dimensional vectors. If considering blue noise samples as a function $\mathbf{y} = f(I)$, plotting the vector y values as dots on a binary image and taking the DFT of that image is what shows blue noise. Blue noise samples can be used when you need to take multiple samples from a sampling domain.

Blue noise masks take in integer coordinates and return N dimensional vectors. If considering blue noise masks as a function $\mathbf{y} = f(I_1, I_2, ..., I_M)$, the scalar or vector valued image made up of the M dimensional grid for all possible input locations are analyzed to define the samples as blue noise. Blue noise masks can be used when you need a random value per pixel, as in all the examples in our paper. In contrast, blue noise samples or point sets cannot give per-pixel values, which is needed for the rendering algorithms shown in the paper.

Figure 1 shows the difference between blue noise samples and masks, while showing their similarities in frequency space.

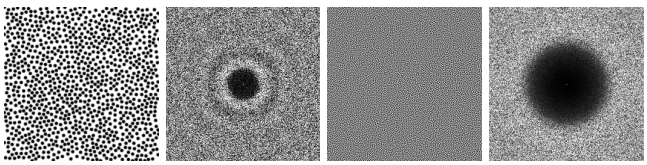


Figure 1: The left two images show blue noise sample points in 2D and the magnitude of the discrete Fourier transform (DFT). The right two images show the same for a blue noise mask.

Author's address:

2.2 Using Blue Noise Masks In Rendering

Blue noise masks are tiled in screenspace and used as a source of per pixel random numbers for stochastic rendering techniques, with the goal of making error that is harder to see and easier to remove than error left by either low discrepancy sequences or white noise.

The perceptual benefit of blue noise as well as the ease of denoising it can be seen in Fig. 2, where an image has been dithered with white and blue noise and quantized to 8 colors (1 bit per color channel). In the raw images, the blue noise looks better and finer details are more discernable. In the Gaussian blurred versions of the images, the blue noise dithered image becomes smooth, whereas the white noise dithered image retains randomized low frequency blobs.

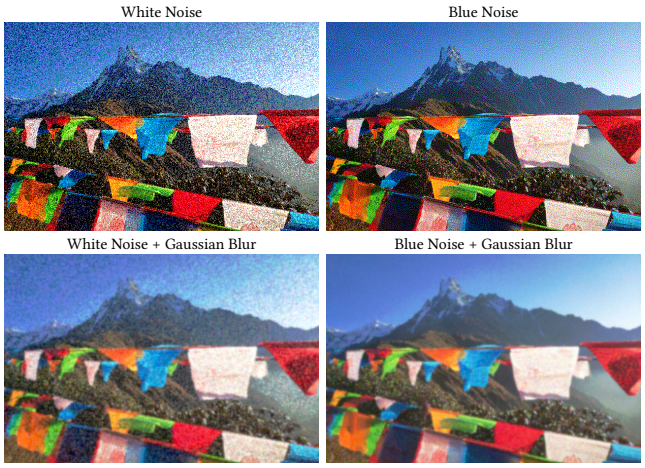


Figure 2: A comparison of white noise versus blue noise, and corresponding spatially filtered results. White noise results in visual clumps when filtered with a low-pass filter (blur) due to low frequencies present in the noise. Blue noise images do not contain these low frequencies, and thus do not contain clumps in the filtered output.

3 ADDITIONAL RESULTS

We now describe some additional applications and results (all generated using scalar spatiotemporal blue noise masks).

3.1 Stochastic Transparency

Stochastic transparency is the process of stochastically choosing whether to ignore a sample based on a material's transparency level. This is useful in situations such as deferred lighting where you are storing information about how to shade a pixel, instead of the shaded result itself, and it is impractical to store multiple layers to later calculate proper transparency.

Sophisticated algorithms have been developed by Enderton et al. [6], Wyman and McGuire [15], and are also discussed in the work by Wyman [14], but the core idea of stochastically accepting

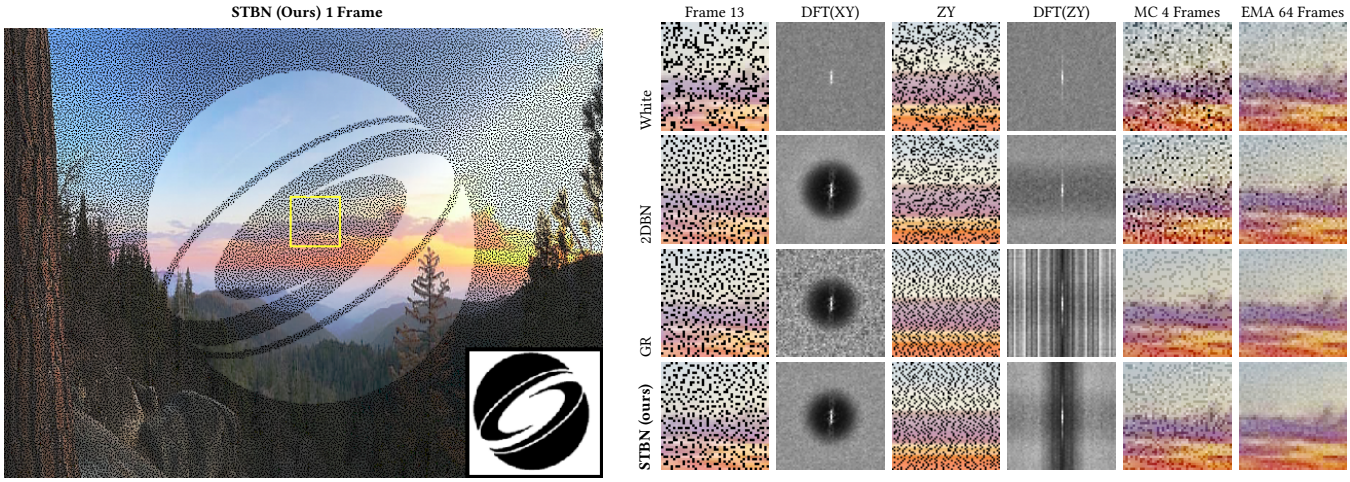


Figure 3: Stochastic transparency with $\alpha = 0.9$ using various noise types and integration schemes. At 4 frames of Monte Carlo integration, our STBN provides better quality than 2D blue noise, and is competitive with golden ratio (GR) animated blue noise. Under 64 frames of exponential moving average (EMA), our STBN provides better quality than golden ratio noise. Columns 2 and 4 are averaged DFTs to show expected spectra.

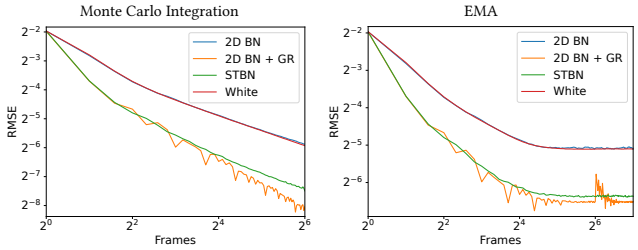


Figure 4: Convergence rates in stochastic transparency of various types of noise. Golden ratio (GR) animated blue noise converges marginally faster than our spatiotemporal blue noise (STBN), but damages frequencies spatially, and has an error spike under EMA when the sequence needs to restart to avoid numerical issues.

or rejecting a pixel remains the same. Our algorithm described here is aimed at low computational costs (a single texture read and comparison), giving blue noise distributed error in screen space as 2D blue noise does, but also converging faster. We show renderings of stochastic alpha using various types of scalar valued spatiotemporal noise under both Monte Carlo integration and EMA in Fig. 3.

When looking at frame 13 in isolation, independent 2D blue noise and spatiotemporal blue noise have the same quality which is correct and shows that our noise is as good for each slice in time. Under 4 frames of Monte Carlo or 64 frames of EMA, our spatiotemporal blue noise shows much better convergence than independent blue noise. Our method is competitive with golden ratio animated blue noise for 4 frames of Monte Carlo, but is superior at 64 frames of EMA where the golden ratio sequence restarts. Under motion, the golden ratio shows high-frequency strobing. Convergence graphs are shown in Fig. 4, which reveal the same story as what is seen visually.

3.2 Dithering

Dithering is the process of adding a small random value before quantization to turn quantization artifacts (banding) into noise instead. This allows less memory to be used while preserving image quality. Dithering rounds the quantized value up or down randomly, with probability to round down being higher as the value gets closer to the lower boundary of quantization. White noise is not often preferred in dithering, but Bayer [2] and the less real-time friendly techniques, such as Floyd-Steinberg error diffusion [7], do not consider the time axis. For a more in depth read about dithering, consult Christou's thesis [3].

The rendered results in Fig. 5 reveal that our scalar noise deliver approximately the same quality as golden ratio animated noise and much better than the other types of noise. Our noise again provides better image quality at 64 frames of EMA though. As before, the golden ratio noise has high-frequency strobing. Convergence graphs are nearly identical to Fig. 4, and so are omitted.

3.3 Volumetric Rendering

The paper included results from volumetric rendering as the teaser image, which showed a slightly different layout and details than other rendered results. Figure 6 shows results in the other format.

3.4 Preserving Blue Noise Over Time

The main paper shows that XY slices of our noise are 2D blue noise spatially, and that Z lines of our noise are 1D blue noise temporally. It also showed how using the noise to integrate the identity function resulted in spatial blue noise of lower magnitude than 2D blue noise for the sample sample counts. In Fig. 7 we show the same in an ambient occlusion technique.

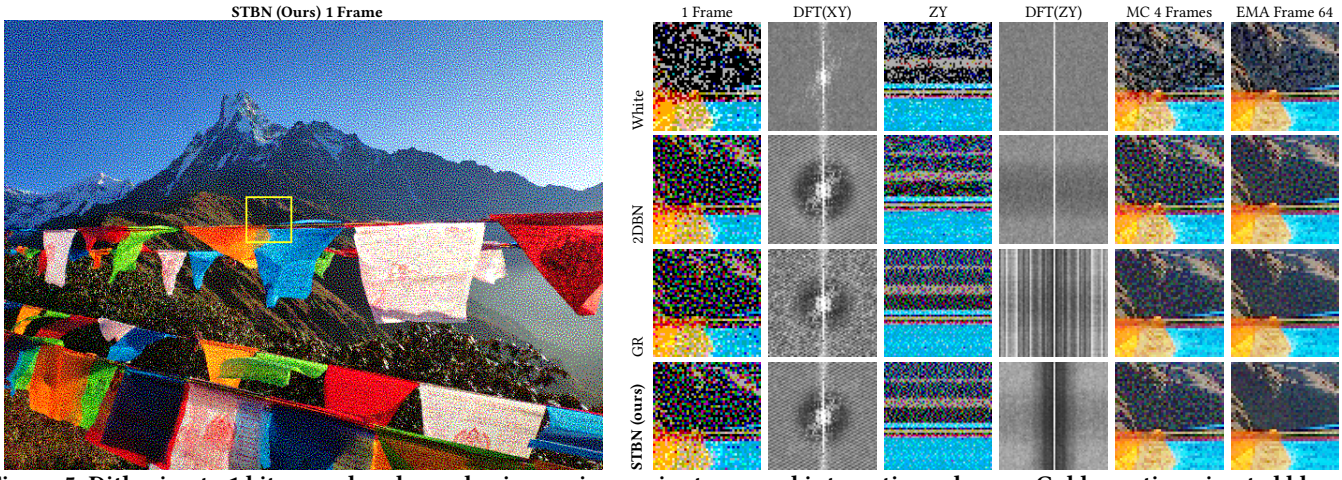


Figure 5: Dithering to 1 bit per color channel using various noise types and integration schemes. Golden ratio animated blue noise and our spatiotemporal blue noise generate images with better quality than the other types of noise, and look comparable under Monte Carlo integration. Our STBN generates better image quality than golden ratio under 64 frames of TAA, however.

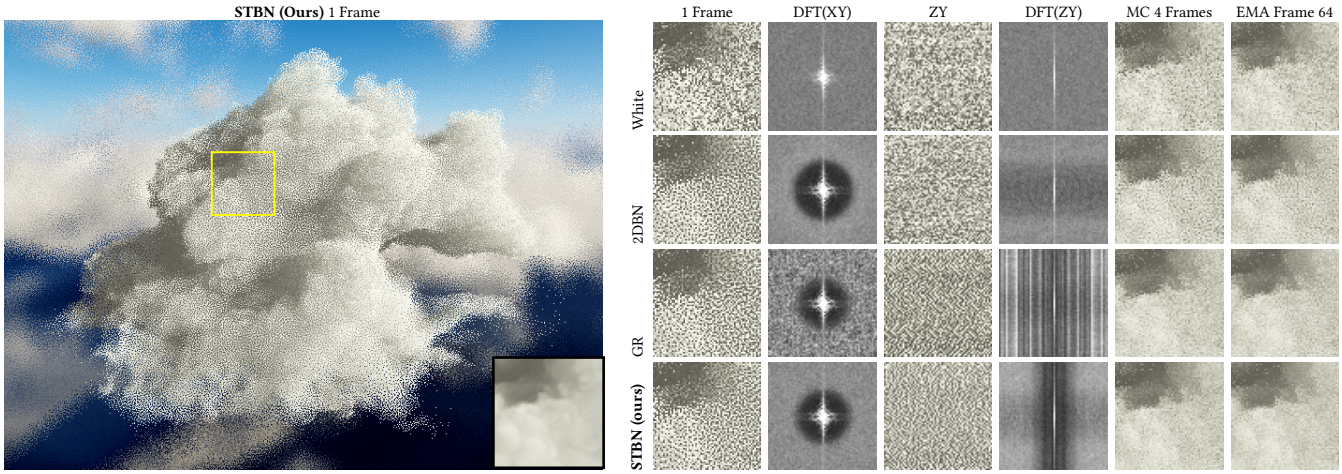


Figure 6: Single scattering volume rendering using various noise types and integration schemes. Our spatiotemporal noise and golden ratio animated blue noise are comparable, but we have slightly better convergence and are more temporally stable under EMA as can be seen in the supplemental materials. The ground truth is shown in the inset in the left image (lower right corner).

3.5 Comparing STBN to 2DBN + Golden Ratio

Figure 9 shows how blue noise animated with the golden ratio has uneven frequency content each frame and blue noise quality, whereas spatiotemporal blue noise has even frequency content each frame. The uneven frequency content means that each frame would need to use a different filter to optimally remove the blue noise. The uneven blue noise quality means that some frames will be more difficult to denoise than others.

The sequence also multiplies the frame number by the golden-ratio, which leads to numerical issues as the frame number increases. The common fix uses modulus, which creates a discontinuity as can be seen in Fig. 4 at frame 64.

The benefit of STBN over golden rato animated blue noise can also be seen in renderings, such as in the pica pica [1] scene in Fig. 8 where the albedo channel of a gbuffer has been quantized to 2 bits per color channel, dithered using the two types of noise in question, and denoised using a custom real time spatiotemporal filter. Spatiotemporal blue noise is more converged.

4 EXTENSIONS

We now describe a number of possible extensions of our method, which are useful in some practical situations.

4.1 Getting Multiple Values Per Pixel

It is fairly common to need more than 1 random value per pixel, such as when taking multiple samples in a frame. One way to handle this situation is to advance the z axis index for each sample one needs, instead of only advancing it by the frame number. Since

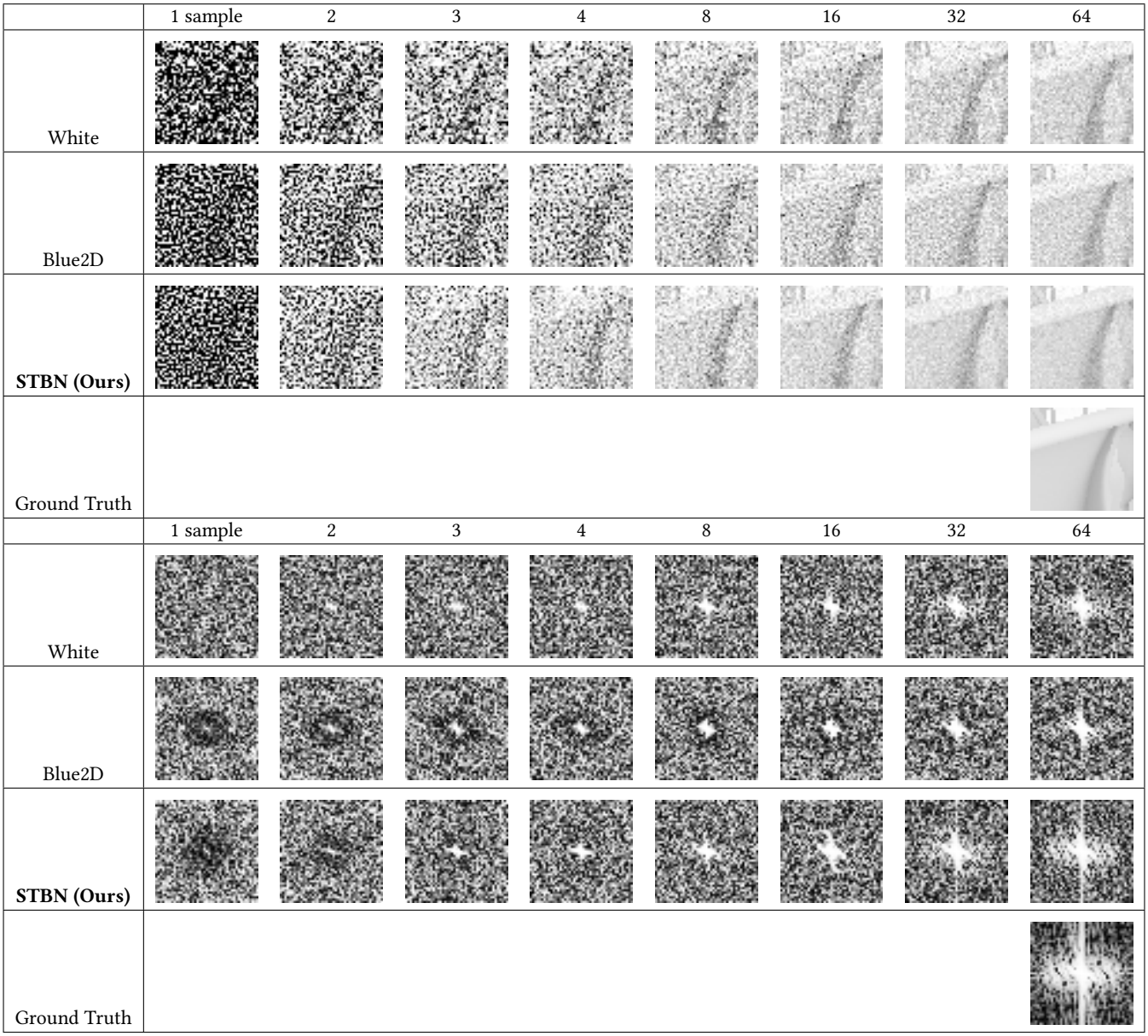


Figure 7: Top: Ambient occlusion rendered with different numbers of noise types and sample counts. Bottom: The discrete Fourier transform magnitude of the rendered images. Blue2D and our STBN both show blue noise frequencies in the render, but STBN converges toward the real render more quickly.

the values are progressive on the Z axis, taking multiple samples per frame along the Z axis is also progressive. One would use this method when taking multiple samples per pixel since the values would be correlated the correct way.

Another way to handle this situation is motivated by Fig 10, which shows that blue noise textures have correlation over small distances, but values are uncorrelated at longer distances. This second method for getting multiple values per pixel is to read N values per pixel at N different offsets, preferably with those offsets being as distant from each other as possible toroidally to be maximally uncorrelated. Doing this, one ends up with N spatiotemporal blue noise values that have no correlation to each other. You would use this method when you wanted multiple uncorrelated values.

We used this approach in Fig. 5, since dithering an RGB image requires three random values per pixel. Specifically, we used the R2 low discrepancy sequence [13] to get three nearly maximally spaced offsets to read the texture at to get three fairly independent spatiotemporal blue noise values per pixel.

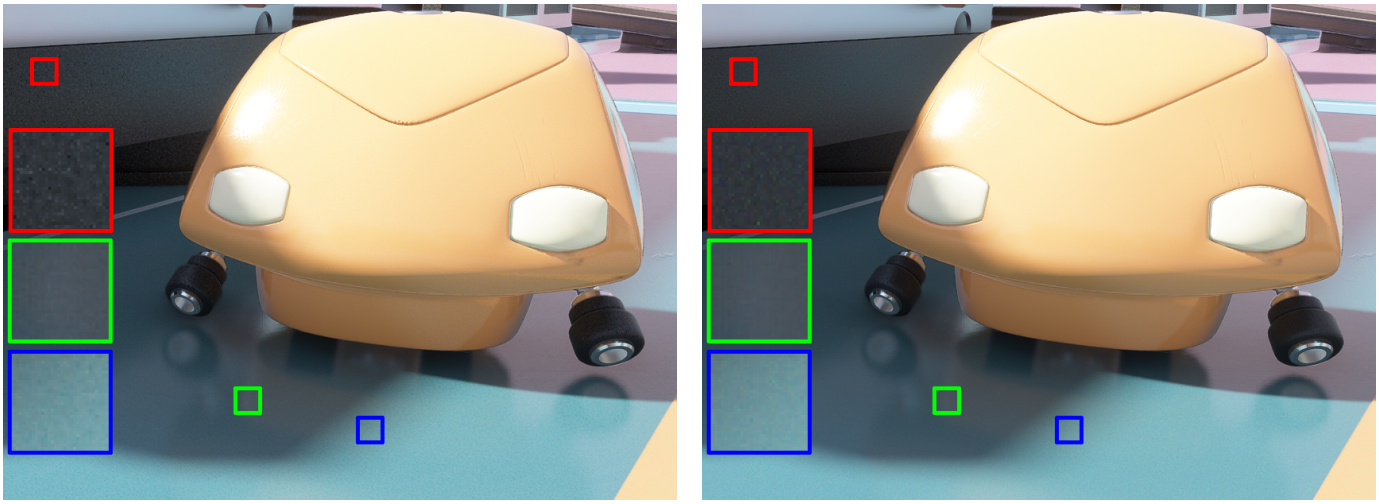


Figure 8: GBuffer albedo dithered before quantization to 2 bits per color channel and denoised with a custom real time spatiotemporal filter. Golden ratio animated blue noise (left) is noticeably much less converged than spatiotemporal blue noise (right).

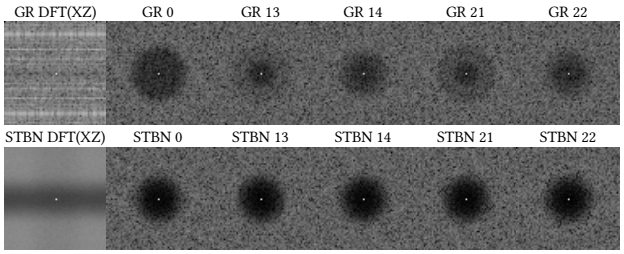


Figure 9: A comparison of blue noise quality between golden ratio (GR) blue noise and STBN (ours) over time. The blue noise qualities of GR are periodically damaged over time, resulting in a strobing pattern. STBN results in consistent blue noise qualities from one frame to the next.

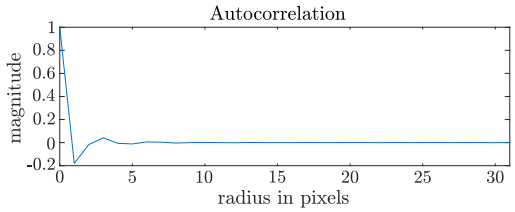


Figure 10: For this diagram, we computed the autocorrelation of 64 blue noise textures of resolution 64×64, which resulted in a circularly symmetric image. We show a radial plot of the average of those. In blue noise textures, neighbors have very different values, which causes a rippling of correlation and anti-correlation for small values of the radius, but rapidly decays to zero, i.e., to decorrelated values.

4.2 Higher Dimensional STBN

Our generalized void and cluster algorithm (for scalar STBN) runs in D dimensions to generate a mask M, where D is the number of parameters needed to index into the mask to get a scalar value. These parameters are the axes of the blue noise mask. The D dimensions are broken up into one or more sets G, where each set of G contains one or more dimensions. A specific set g of G with a membership count of d implies that all d dimensional projections of the D dimensional blue noise mask should be d dimensional blue noise, when only the axes within that group vary, and all other axes are held constant. We will also define a group of axes h as being all axes which are not in g.

Once the dimensions are grouped, each group g is naturally mapped to an energy function E_g by only using the dimensions present in the group (denoted as $\mathbf{p}_g$ and $\mathbf{q}_g$) within the usual Gaussian energy function, so long as the axes from the corresponding group h are equal between the two pixels. The energy function between two pixels is the sum of all E_g functions between those pixels, and the energy field F is the sum of energy at each pixel, from every other pixel. This is summarized as

$$
\begin{aligned}
E_g(\mathbf{p},\mathbf{q}) &= \begin{cases} \exp\left(-\frac{\|\mathbf{p}_g-\mathbf{q}_g\|^2}{2\sigma_g^2}\right), & \text{if } p_h = q_h \\ 0, & \text{otherwise.} \end{cases} \\
E(\mathbf{p},\mathbf{q}) &= \textstyle\sum_{\mathbf{g}\in G} E_g(\mathbf{p},\mathbf{q}), \\
F(\mathbf{p}) &= \textstyle\sum_{\mathbf{q}\in M} E(\mathbf{p},\mathbf{q}).
\end{aligned}
\tag{1}
$$

Each dimension can be of different size, can use a different σ values to control the frequency content of the result, and can also choose to compute distances toroidally or not. The original void and cluster algorithm can be seen as a special case such that D is any arbitrary value, and that there is only a single group in G which contains all axes. Thus, the void and cluster algorithm makes D dimensional blue noise masks. When considering spatiotemporal blue noise, D is 3, and G has two groups in it: g_{xy} and g_z. In that sense, spatiotemporal blue noise can also be seen as a 2D×1D blue noise mask.

Table 1: The storage size of masks, and generation time.

Dimensions	Size	Generation Time
64×64	4 kB	< 1 second
$32 \times 32 \times 16$	16 kB	< 1 second
$32 \times 32 \times 32$	32 kB	2 seconds
$64 \times 64 \times 16$	64 kB	10 seconds
256×256	64 kB	10 seconds
$128 \times 128 \times 8$	128 kB	44 seconds
$64 \times 64 \times 64$	256 kB	3 minutes
$128 \times 128 \times 32$	512 kB	12 minutes
$64 \times 64 \times 16 \times 16$	1 MB	48 minutes
$64 \times 64 \times 64 \times 64$	16 MB	206 hours (Estimated)

Frequency analysis of a 2D×1D×1D and 2D×2D mask can be seen in Fig. 11, which shows the desired frequency behaviors for axis pairs. The 2D×2D blue noise shows 2D blue noise on the XY plane and the ZW plane, but white noise everywhere else. The 2D×1D×1D blue noise shows 2D blue noise on the XY plane and shows 1D blue noise on both the Z and W axes.

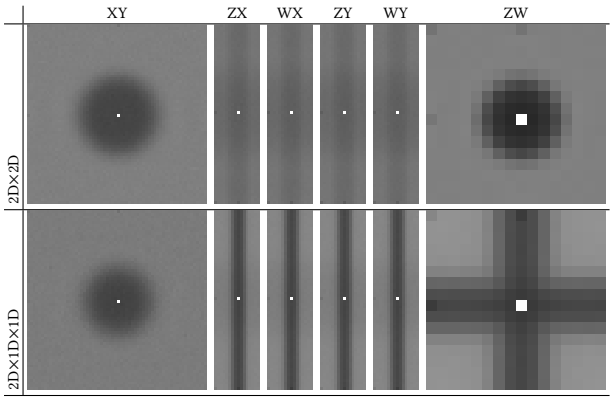


Figure 11: DFTs of the 2D projections of 4D blue noise masks that are $64 \times 64 \times 16 \times 16$. All projections averaged to show expected frequency spectra.

Generation time of blue noise masks using void and cluster is a function of the total pixel count n, and is nearly $O(n^2)$ so that doubling the pixel count roughly quadruples the generation time. Table 1 shows some reasonable mask sizes, their size in bytes assuming a single channel 8-bit texture, and the time taken to generate them. We have found that smaller textures such as $64 \times 64 \times 16$ (64 kB) for spatiotemporal blue noise, and $64 \times 64 \times 16 \times 16$ (1 MB) have been sufficient in our rendering tests.

Note that this generation time is a preprocess, and is simply used to generate a blue noise texture. At run-time only a simple texture-read needs to be performed, so our algorithm (spatiotemporal blue noise masks in the main paper or higher-dimensional masks discussed here) has essentially no overhead, and can be included in any real-time image synthesis method.

The same energy function modifications that allow the void and cluster based algorithm to generate higher dimensional scalar valued masks can also be applied to the swap based algorithm to generate higher dimensional vector valued masks. That algorithm runs until a user specified number of iterations have occurred, or until a user specified error level has been reached. Because of this, the runtime is tuneable for quality vs speed.

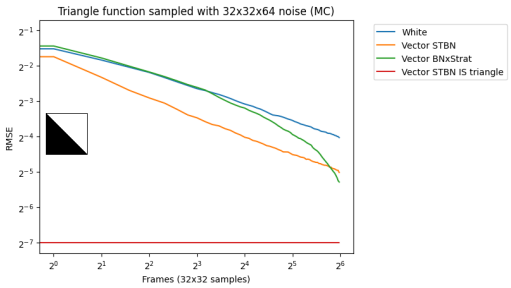


Figure 12: Noise that is blue over space but stratified over time converges well, but not until all samples are used.

4.3 Spatial BN Stratification Over Time

Our research was motivated by the desire for spatial blue noise, which also converges rapidly over time. However, it is not known how to generate such spatiotemporal masks, which are blue over space but have a particular temporal sequence. We solved this in the main paper by developing algorithms for spatiotemporal blue noise. But we were also interested in considering other fast-converging temporal sequences, such as stratified sampling over time, for which we were able to find an initial solution.

The algorithm to make vector valued spatiotemporal blue noise can be modified to have an energy function which returns zero energy between pixels if they are from different slices (different z value) or if the temporal histograms of the pixels involved in the swap get worse from the swap. Doing this generates noise that is blue over space but stratified over time, with a per pixel randomized stratification order.

However, stratification is not progressive, so this noise does not converge well until all samples have been taken. It does converge quite well at that point though, as can be seen in Fig 12. Given these limitations, the main paper focuses on spatiotemporal blue noise.

4.4 Spatiotemporal Point Sets

Since our (scalar) spatiotemporal blue noise masks are created using the void and cluster algorithm, they have the property that if you threshold the mask values to some percentage, the same percentage of the pixels will survive. Thresholding a blue noise mask produces a *sample point set*. Our thresholded masks produce blue noise sample patterns over both space and time. While there are better algorithms for generating blue noise sample patterns, such as blue noise through optimal transport (BNOT) [4], they do not handle the time axis. To the best of our knowledge, our algorithm is the first method for making spatiotemporal blue noise sample patterns.

Figure 13 shows how thresholded masks are not as high quality spatially as BNOT, but they are able to make point sets of any density. Figure 14 shows how the thresholded point sets keep their desired frequency spectra over axis groups.

An example usage case is doing sparse ray tracing, using an importance map to decide which regions should have more samples. For each pixel $\mathbf{p}$, we shoot a ray if and only if the importance map value $f(\mathbf{p}) \in [0,1]$ is greater than the mask value $g(\mathbf{p}) \in [0,1]$. Using white noise over space and time, the spatial pattern will have redundant samples and large holes, while also having duplicated samples over time. Using spatial blue noise makes the sampling more even over space, but still has the problems of white noise over time. Using spatiotemporal blue noise means that samples are evenly distributed over both space and time. This is shown in Fig. 16.

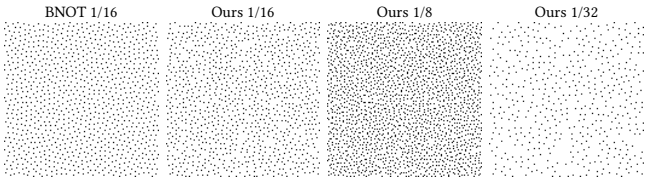


Figure 13: Comparing 1024 BNOT samples with a slice of a $128 \times 128 \times 10$ spatiotemporal blue noise mask (ours) thresholded to different levels. BNOT is much higher quality over space, but has a fixed density and gives no treatment to the time axis, necessitating independent sample sets to be white noise over time.

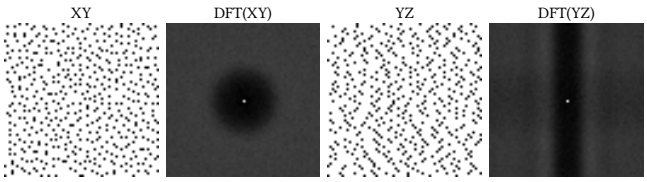


Figure 14: 2D projections of our $64 \times 64 \times 64$ spatiotemporal blue noise (STBN) masks thresholded to point sets, which keep their desired spectra. DFTs are averaged over 64 slices.

4.5 Curve Inversion

Putting a scalar spatiotemporal blue noise value through a space filling curve can give you a vector spatiotemporal blue noise value. We have had better luck with Hilbert curves than Morton curves, but neither makes vector valued noise which is as good as noise made through the swapping algorithm, described in the main paper, see Fig 15. This method is extremely quick at generating vector valued masks from scalar ones, however.

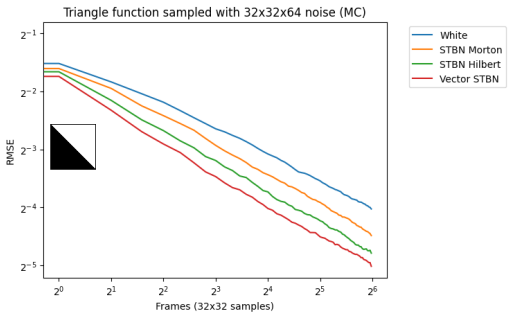


Figure 15: Vector valued spatiotemporal blue noise is best made with the swap algorithm, but Hilbert and Morton curves can also be used to convert scalar masks to vector.

A FREQUENCY ANALYSIS OF INTEGRATION AND INVERSION

We develop a simple frequency analysis of integration with blue noise offsets, and explain the inversion method in Heitz and Belcour [9]. While our theory is closely inspired by previous work [5, 10, 11], we do not believe this frequency analysis of the spatial distribution of error has appeared before, and it may provide valuable insight into previous algorithms. However, it is not required for understanding our method in the main paper.

For simplicity, consider a 1D integral at a single pixel,

$$I_N = \int s(y) f(y)\, dy \qquad s(y) = \frac{1}{N} \sum_{i=1}^{N} \delta(y - y_i), \tag{2}$$

where $s(y)$ is the sampling pattern, $f(y)$ is the integrand we seek to integrate by Monte Carlo, I_N is the output image pixel radiance from using N samples, and y is the variable we are integrating over. Note that the sampling pattern s is actually N points y_i, which can be treated as delta functions. The integral can also be estimated in the Fourier domain [5],

$$I_N = (S(\omega) \otimes F(\omega))|_{\omega=0} = \int_{-\infty}^{\infty} S(\omega) F^*(\omega)\, d\omega, \tag{3}$$

where we use capital letters to denote the Fourier transforms, and $*$ denotes the complex conjugate (note that $F^*(\omega) = F(-\omega)$).

We now consider perturbing the sampling pattern by a constant γ. That is, we replace all y_i by $y_i + \gamma$. In practice, the values of γ will differ at each pixel, in our case with a blue noise-like pattern. We now have,

$$s(y;\gamma) = \frac{1}{N} \sum_{i=1}^{N} \delta(y - y_i - \gamma) = s(y - \gamma) \tag{4}$$

$$S(y;\gamma) = e^{-i\omega\gamma} S(y), \tag{5}$$

where the last line follows simply from the addition theorem for Fourier series. Note that the magnitude of the Fourier spectrum for the sampling pattern remains the same, only its phase is shifted, with different phase shifts at each pixel corresponding to each γ. If we now plug the shifted sampling pattern for the integral we seek into Equation 3, we obtain,

$$I_N = \int S(\omega) F^*(\omega) e^{-i\omega\gamma}\, d\omega. \tag{6}$$

Now, we can define $G(\omega) = S(\omega)F^*(\omega)$, where the corresponding angular domain function is given by a convolution[1] $g(x) = f(x) \otimes s(x)$. Note that

[1] Since $F^*(\omega) = F(-\omega)$ so that $G(\omega) = S(\omega)F(-\omega)$, this convolution has a slightly non standard form with the plus sign instead of the conventional minus in the primal angular domain, $s(x) \otimes f(x) = \int s(x+y) f(y)\, dy$.

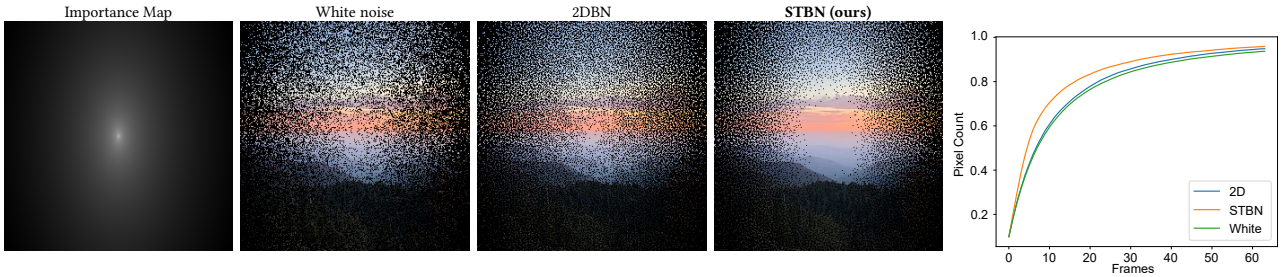


Figure 16: To the left, we show an importance map, which was used to generate the next three images. These were generated by simulating adaptive sampling using the importance map to guide where the samples should be over five frames. The images were generated in such a way that the more pixels that are lit, the more unique pixels have been sampled. STBN provides the best results here, since it has substantially more lit pixels. Right: a diagram showing the percentage of unique pixels sampled over time as a function of frame number. White noise can have redundant sampled pixels each frame, and over time, 2D blue noise reduces redundant pixels over space, and STBN (ours) reduces them over time as well.

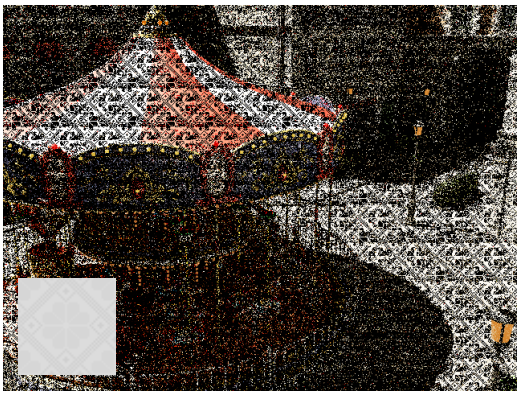


Figure 17: One SPP path tracing, rearranging the seeds using the Heitz/Belcour technique to make the render noise match the stylized texture shown in the lower left.

Equation 6 can be viewed as an inverse Fourier transform, evaluated at the value of $-\gamma$ (we omit normalizations),

$$g(x) = s \otimes f = \int \left(S(\omega)F^*(\omega)\right) e^{i\omega x}\, d\omega \tag{7}$$

$$I_N = g(-\gamma). \tag{8}$$

Note that this analysis holds not just for the value I_N but for the error as well, if we simply zero out the DC term in S (the DC term corresponds to the ideal sampling pattern resulting in the true integral value).

Finally, we analyze the spatial distribution of the noise, which has not usually been considered in previous frequency analyses. We follow the reasoning from Heitz and Belcour [9], which assumes the function f (and hence g) is locally constant in a patch. From the above results, we can then define,

$$I_N = g(-\gamma) \quad \Rightarrow \quad I_N(x) = g(-\gamma(x)). \tag{9}$$

In Heitz and Belcour's work [9] and in this paper, we want the error in $I_N(x)$ to be distributed as a blue noise pattern. The simplest approach [8] is to choose the offsets/seeds $\gamma(x)$ in a blue-noise pattern. However, the pattern in I_N is not necessarily blue-noise since it is transformed by the function $g(-\gamma(x))$. Instead, if we want the distribution $I_N(x)$ (or equivalently the error in the distribution) to achieve some desired pattern $I_N(x) \sim \alpha(x)$, then we seek that $g(-\gamma(x)) = \alpha(x)$ where $\alpha(x)$ is the desired blue noise pattern,

$$\gamma(x) = g^{-1}(\alpha(x)), \tag{10}$$

where we have ignored the negative sign, and we need to explicitly invert the function g. However, this seems intractable, since we do not know g nor its inverse (or even the integral of f); indeed that's what we are trying to find by Monte Carlo integration. Instead, Heitz and Belcour [9] perform this inversion in an ordinal numerical fashion within a local patch. They sort the seeds $\gamma(x)$ and the function values $g(\gamma(x))$ from just applying naive blue-noise patterns in the first step. Then, instead of doing exact inversion and interpolation, one can just map the function ordinally in terms of the sorted order. So now, in the next frame, if you have a seed $\alpha(x)$ at pixel x, we get the corresponding ordinal value of α and see which $g(\gamma)$ gave that corresponding ordinal value, effectively inverting $\gamma(x) = g^{-1}(\alpha(x))$.

Note that the inversion method is orthogonal to the specific desired pattern $\alpha(x)$ (or for that matter the initial pattern $\gamma(x)$). Indeed, we have successfully used the method even to produce stylized error patterns as can be seen in Fig. 17.

We have integrated our spatiotemporal blue noise into Heitz and Belcour's method [9], using both seed sorting and retargeting and show improved convergence performance, as shown in Fig. 18.

REFERENCES

[1] Joghan Andersson and Colin Barré-Brisebois. 2018. Shiny Pixels and Beyond: Rendering Research at SEED. In *Game Developer's Conference*.

[2] B.E. Bayer. 1973. An optimum method for two-level rendition of continuous-tone pictures. *IEEE International Conference on Communications* 1, 11–15.

[3] Cameron N. Christou. 2008. Optimal Dither and Noise Shaping in Image Processing. MSc thesis, University of Waterloo.

[4] Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue Noise through Optimal Transport. *ACM Transactions Graphics* 31, 6, Article 171 (2012), 11 pages.

[5] Frëdo Durand. 2011. *A Frequency Analysis of Monte-Carlo and Other Numerical Integration Schemes.* Technical Report TR-2011-052. MIT CSAIL.

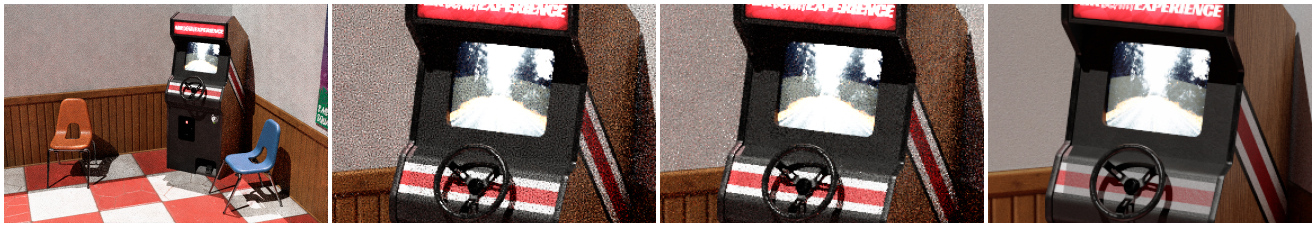
[6] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke. 2011. Stochastic Transparency. *IEEE Transactions on Visualization and Computer Graphics* 17, 08 (2011), 1036–1047.

[7] R. W. Floyd and L. Steinberg. 1976. An Adaptive Algorithm for Spatial Grayscale. *Proceedings of the Society of Information Display* 17, 2, 75–77.

[8] Iliyan Georgiev and Marcos Fajardo. 2016. Blue-Noise Dithered Sampling. In *ACM SIGGRAPH Talks.* Article 35, 1 pages.

[9] E. Heitz and L. Belcour. 2019. Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames. *Computer Graphics Forum* 38, 4 (2019), 149–158.

[10] Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. 2015. Variance Analysis for Monte Carlo Integration. *ACM Transactions on Graphics* 34, 4 (2015), 124:1–124:14.



(a) Heitz and Belcour w/ STBN **(b) 2D BN** **(c) STBN (Ours)** **(d) Ground Truth**

Figure 18: A comparison of blue noise versus spatiotemporal blue noise (center) applied to the seed rearranging technique by Heitz/Belcour, with a ground truth image on the right. Both noise signals were filtered using an exponential moving average and $\alpha = 0.1$. This example illustrates that our spatiotemporal blue noise can be used in combination with state of the art techniques such as the Heitz/Belcour algorithm.

[11] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. 2012. A Theory of Monte Carlo Visibility Sampling. *ACM Transactions on Graphics* 31, 5 (2012), 121:1–121:16.

[12] Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. 2016. Projective Blue-Noise Sampling. *Computer Graphics Forum* 35, 1 (2016), 285–295.

[13] Martin Roberts. 2018. The Unreasonable Effectiveness of Quasirandom Sequences. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/. [Online; accessed 3-March-2021].

[14] Chris Wyman. 2016. Exploring and Expanding the Continuum of OIT Algorithms. In *High Performance Graphics*. 1–11.

[15] Chris Wyman and Morgan McGuire. 2017. Hashed Alpha Testing. In *Symposium on Interactive 3D Graphics and Games*. Article 7.