UTRECHT UNIVERSITY

MASTER'S THESIS

GAME & MEDIA TECHNOLOGY

Blue Noise Distributed MCMC Decorrelation of ReSTIR

Author:
Oscar FICKEL

Supervisor:
Dr. ir. Peter VANGORP
Second Supervisor:
Prof. dr. ir. A.C. (Alex) TELEA

September 10, 2023

Abstract

Spatiotemporal resampling (ReSTIR) [Bitterli et al., 2020; Lin et al., 2021] is a popular new ray tracing technique. Unfortunately it can suffer from correlation artifacts if left unchecked. One solution for this is offered by Sawhney et al. [2022] in the form of Markov Chain Monte Carlo mutations. We reimplement and evaluate their proposed algorithm, and attempt to optimise it for blue noise. Our addition of blue noise mutations is unsuccessful, but still provides some insight into how the underlying characteristics of decorrelated ReSTIR work against a simple solution for achieving blue noise.

Contents

1 Goals

The decorrelated ReSTIR algorithm Sawhney et al. [2022], while being an improvement on previous iterations of ReSTIR Bitterli et al. [2020]; Ouyang et al. [2021]; Lin et al. [2021], can still stand to gain further improvements nonetheless, as is noted by the authors themselves as well. One way of reducing visual noise is incorporating blue noise. While blue noise could theoretically be added to any given path tracer, it might be possible to tweak the decorrelation procedure in a way that would yield better results than simply using a blue noise mask for generating random numbers for the samples.

The main research question of this thesis is therefore: Can blue noise be efficiently incorporated into the decorrelation of ReSTIR algorithms, in such a way that it produces more visually pleasing results using the same number of samples per pixel, and without costing too much additional time? As a sub goal it would be useful to test whether the resulting implementation also really performs better than simply applying a blue noise mask without any further optimisations.

2 Introduction

Ray tracing has become an increasingly more popular concept in computer graphics in the past few years. Especially its application in video games has helped it gain traction. In fact, according to Google Trends, the most popular search term to combine with ray tracing is 'Minecraft' [trends.google.com, 2023]. This shows the importance of real time ray tracing that can produce stunning images within milliseconds. The main idea behind ray tracing is to more accurately simulate the way light travels through a scene, resulting in realistic and physically accurate effects such as shadows, diffuse reflections, participating media, etc.

A simple Whitted-style ray tracer would shoot a ray from the camera through each pixel of the screen, test for each ray if it hits an object, and then shoot a ray towards the light source to test for visibility and brightness. This type of ray tracer unfortunately still lacks indirect lighting and soft shadows.

Path tracing takes it a step further by solving what is known as the rendering equation [Kajiya, 1986] for each pixel. This aims to encompass all (reflected) light that hits a certain surface point x, and is reflected in a given direction. One version of the equation requires integrating over all of the surface areas x' that potentially send light that bounces via x towards the camera s, to get the sum of all the incoming light. The amount of light traveling from x to s is then defined as follows:

$$L(s \leftarrow x) =$$

$$L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow x')L(x \leftarrow x')G(x \leftrightarrow x')dA(x')$$

Here A refers to the all the surface area that is integrated over. Furthermore, L_E is then the directly emitted light, f_r is the fraction of light that gets reflected, and G is the geometry factor that might hinder light transport. Note that this means that $L(x \leftarrow x')$ is actually recursively the rendering equation for the indirect light that has bounced or was emitted from x' towards x, and then via this equation is passed along to s.

While this is perhaps the more intuitive version of the rendering equation, more often it is defined as an integral over the hemisphere:

$$L_o(x, \omega_o) = L_E(x, \omega_o) + \int_\Omega f_r(x, \omega_o, \omega_i)L_i(x, \omega_i)cos\theta_i d\omega_i$$

Here L_o is the amount of outgoing light, L_i is the amount of incoming light, Ω is represents the hemisphere, ω is a direction in this hemisphere. L_E is still the emitted light and f_r still the fraction of reflected light. The cosine how incoming radiance gets converted to irradiance, which is the amount (power) of light incoming per unit surface area. Note that f_r in this formula is often refered to as the Bidirectional Reflectance Distribution Function (BRDF). It should always obey the Helmholtz reciprocity, meaning that $f_r(x, \omega_o, \omega_i) = f_r(x, \omega_i, \omega_o)$.

The rendering equation is in most cases too complex to be solved analytically, so it is usually solved with the help of Monte Carlo integration [Metropolis and Ulam, 1949]. This is a method used to estimate an integral by taking the average of multiple random samples. In the case of the rendering equation, these samples would be rays shot from x into the scene, to compute the amount of incoming light in those directions. If enough samples are taken in random directions, eventually the average will converge to the total amount of incoming light, from all directions.

An unbiased estimator of the rendering equation is one for which the expected value is equal to the true value of the rendering equation. So if the estimator is run many times separately, the average result should be correct. On the other hand a biased estimator can still be a consistent one, in which case it will still eventually converge to the correct result as the number of samples increases [Fajardo et al., 2001; Kirk and Arvo, 1991]. While one would naturally prefer for their estimator to be unbiased to guarantee correct renders, there is a trade-off in that biased estimators can often perform faster than unbiased estimators.

To speed up this process of Monte Carlo integration, the concept of importance sampling can be quite helpful. If there is only light coming from one specific direction, you don't want to waste valuable resources shooting rays in directions where there is no light. It would be beneficial to shoot more rays towards bright areas of the hemisphere, and weigh these accordingly to still get a correct average. Especially if we only take a few samples, the probability that we don't shoot a sample towards the light and miss it is now much lower, which means that the variance will be lower as well. The importance of each direction is usually described with a probability density function (PDF). This is a continuous function that represents how the expected amount of incoming light is distributed over the hemisphere. Ideally you would want the PDF you use to determine which samples to take to be exactly proportional to the rendering equation. This of course defeats the purpose of the PDF if it requires computing the rendering equation to better approximate the rendering equation. Therefore cheaper functions are usually used as a PDF that approximate the rendering equation without its more expensive to compute components. An example of such an approximation could be simply the positions and sizes of the light sources in the scene. Techniques like Resampled Importance Sampling (RIS) [Talbot et al., 2005] exist to use these cheaper methods in a clever way to still obtain samples correctly distributed according to a more accurate and expensive PDF.

3 Background

3.1 ReSTIR

The ReSTIR algorithm was initially put forward in 2020 by Bitterli et al. [2020]. It allows for faster rendering of scenes with many lights by reusing samples of direct lighting between neighbouring pixels and between frames. It is based on a combination of Resampled Importance Sampling (RIS) [Talbot et al., 2005] and Weighted Reservoir Sampling (WRS) [Chao, 1982]. ReSTIR was quickly extended to cover global illumination [Ouyang et al., 2021] and volumetric rendering [Lin et al., 2021] as well. In 2022 a more general framework was introduced under the name of GRIS [Sawhney et al., 2022], which aimed to generalise RIS and use this to implement a version of ReSTIR that allows more advanced reusing of paths with the help of shift maps.

3.1.1 Resampled Importance Sampling

As mentioned ReSTIR is partly based on RIS [Talbot et al., 2005]. RIS is used as a cheaper method of importance sampling. Usually with importance sampling, we assume a PDF f that resembles a distribution over the hemisphere at a certain point, with the importance or value of light samples over that hemisphere. When searching for a direction to shoot a ray in, we usually want to use importance sampling and pick a random sample according to its importance. Since an accurate PDF may be expensive to evaluate, RIS proposes to sample from a cheaper function that roughly approximates f, while still obtaining accurate samples overall.

To achieve this, a number of samples are first generated from the simplified function g. Using f we can then assign resampling weights to these samples. Then with the help of the RIS estimator function, the final accurate sample is drawn from these weighted cheaper samples. The contribution weight of the final sample is the sum of resampling weights of the cheap samples over the evaluated contribution of the resampled sample.

3.1.2 GRIS

Generalised Resampled Importance Sampling (GRIS) [Lin et al., 2022] generalises the RIS theory by taking a single sample from a number of cheaper samples from different domains, rather than assuming a consistent domain. It does this with the help of shift mappings, which map a sample from one domain to another. This is especially useful for ReSTIR, as different pixels naturally have different PDFs for incoming light. The original ReSTIR algorithm did not always converge, as a direct result of using the single domain RIS theory while resampling from multiple different domains. Therefore applying the GRIS theory fixes these convergence issues of ReSTIR.

3.1.3 Weighted Reservoir Sampling

WRS [Chao, 1982] is a method for obtaining a single sample from a stream of weighted potential samples, within constant memory. A reservoir stores only its currently selected sample x, the sum of all the weights it has previously seen w_{sum}, and the total number of samples it has seen M. When used in combination with GRIS, the reservoir also needs to store the contribution weight corresponding to the currently selected sample, which is used in the RIS estimator. Every time a new potential sample is presented to the algorithm, it replaces the selected sample with the current sample in the loop with a probability of w/w_{sum}. M is incremented, and the new potential sample's weight is always added to w_{sum} regardless of this probability.

The reservoirs used in ReSTIR work slightly differently to regular reservoir sampling, as we are now dealing with reservoirs from different domains. The value M is now interpreted as the reservoir's temporal confidence weight (Not to be confused with the contribution weight). When combining reservoirs, this confidence weight is used to compute resampling weights for the reservoirs' current samples. These affect which of the two samples the combined reservoir ends up selecting.

A shift mapping is then applied to the sample x_j to obtain a new sample x'_j. This shift mapping is used to map the other reservoir's sample that was obtained from a different PDF, to the domain of the current reservoir. There are multiple ways of implementing such a shift mapping. For combining the reservoirs of two pixels at different times or positions, this means that the shift map turns the other reservoir's light contribution sample into one that could have originated from the current pixel and its PDF. Such a shift mapping should result in a new sample that is as similar to the input sample as possible, to take as much advantage of valuable samples as possible.

3.1.4 Shift Maps

The main shift mapping used by Lin et al. [2022] is the hybrid shift map, which combines two different shift mapping strategies. It involves first performing a random replay, where the random numbers of the input sample are reused to generate the new sample. As soon as certain constraints are met however, reconnection is applied, which connects a point in the new sample to a point in the input sample, and reuses the path from there on. This reconnection cannot be applied to glossy surfaces, as there will be barely any throughput at the angles created by the reconnection.

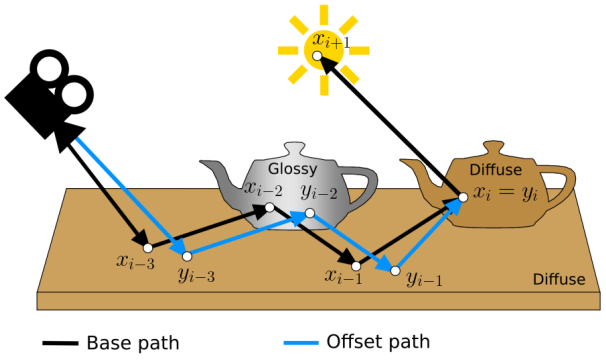


Figure 1: A hybrid shift map. The base path of x is mapped to y by reusing its random numbers, then at y_{i-1} the conditions are met for a vertex reconnection with x_i.

3.2 Decorrelating ReSTIR

3.2.1 Metropolis-Hastings

Markov chain Monte Carlo algorithms are a way of using mutations for local exploration. The most commonly applied algorithm in this category is Metropolis-Hastings (MH) [Metropolis et al., 1953; Hastings, 1970]. This algorithm is also what is used by Sawhney et al. [2022] to generate sample mutations to decorrelate ReSTIR.

It is based on the concept of the Markov chain, which is essentially a sequence of samples where each next sample depends only on the previous sample in the chain. By using a Markov chain whose stationary distribution is proportional to the PDF of a point, we can efficiently generate new samples that are distributed proportionally to this PDF. We generate a new sample from a simpler distribution, dependent on the previous sample in the Markov chain, and accept this with a certain acceptance probability. This acceptance probability is dependent on the complex PDF and is defined such that the eventual stationary distribution of the Markov chain will resemble the complex PDF.

In the initialisation step, a random sample x_0 is generated. Then a number of iterations are run of the following process:

- Generate a candidate sample x'_k from the previous sample, using a proposal distribution $T(x_k \to x'_k)$. Here the transition kernel $T(x_k \to x'_k)$ represents the probability of generating the x'_k sample given the x_k sample.
- Compute the acceptance probability:

$$a(x_k \to x'_k) = \frac{C(x'_k)T(x'_k \to x_k)}{C(x_k)T(x_k \to x'_k)}.$$

  In regular MH we set the contribution function C to be equal to the target PDF $\hat{p}$.
- Set $x_{k+1} = x'_k$ based on the probability a. Otherwise x_{k+1} remains x_k.

Intuitively the ratio of the new target PDF $\hat{p}(x'_k)$ over the old target PDF results in a preference for always accepting any increase in the target PDF, and sometimes rejecting a decrease in target PDF. The ratio of the transition kernels is there to satisfy the detailed balance condition. This condition is necessary to guarantee the existence of a stationary distribution. A stationary distribution is the the probability distribution of the samples produced by the Markov chain, which is reached after a burn-in period of a number of iterations. The acceptance probability ensures this to be proportional to the target PDF $\hat{p}$. The detailed balance condition, which is needed to ensure the existence of a stationary distribution, states that the probability of the Markov chain having produced a sample x_k $(\hat{p}(x_k))$ and subsequently generating and accepting a sample x'_k $(T(x_k \to x'_k)a(x_k \to x'_k))$ must be the same as the probability of having the sample x'_k and transitioning to sample x_k. Put into a formula we get:

$$\hat{p}(x_k)T(x_k \to x'_k)a(x_k \to x'_k) = \hat{p}(x'_k)T(x'_k \to x_k)a(x'_k \to x_k).$$

To ensure that the stationary distribution is unique, the Markov chain must also be ergodic, which practically can be ensured by using a proposal distribution that can generate every possible candidate sample from every possible base sample. Put into formulas: if $\hat{p}(x_k) > 0$ and $\hat{p}(x'_k) > 0$, then $T(x_k \to x'_k) > 0$.

3.2.2 Metropolis Light Transport

The most well-known application of Metropolis-Hastings for path tracing is Metropolis Light Transport (MLT) [Veach and Guibas, 1997]. MLT uses MH to directly solve the rendering equation. It uses both large mutations to ensure that the entire domain is explored, and smaller mutations to take advantage of locally exploring the path space. Local exploration allows for more quickly finding high-contribution paths. MLT is later adapted to operate in Primary Sample Space (PSS) [Kelemen et al., 2002] rather than primary path space. This means that instead of mutating or mapping between the paths themselves, we operate on the random numbers that make up those paths. Each path then corresponds to a vector of random numbers.

Multiplexed MLT [Hachisuka et al., 2014] further improves upon the algorithm. Reversible Jump Metropolis Light Transport (RJMLT) [Bitterli et al., 2017] takes it a step further and introduces a technique for transforming light paths back into PSS. Even more recent developments include Geometry-Aware MLT [Otsu et al., 2018], Selectively Metropolised Monte Carlo light transport [Bitterli and Jarosz, 2019], Delayed Rejection MLT [Rioux-Lavoie et al., 2020], and Ensemble MLT [Bashford-Rogers et al., 2021].

What mainly separates how MH is applied in MLT versus for decorrelating ReSTIR, is that in the latter case the mutations do not drive the spatiotemporal reuse, but merely decorrelate the resampling that is already being done. Increasing the number of mutations therefore does not affect the amount of error, only how it is distributed.

Aside from MLT, Sequential Monte Carlo [Doucet et al., 2001] and Population Monte Carlo [Cappé et al., 2004] are also MCMC methods that have seen use in ray tracing [Ghosh et al., 2006; Lai et al., 2009], though not as extensively as MLT.

3.2.3 Primary Sample Space

When using the PSS for mutations instead of path space, the space they operate in gets transformed through the sampling PDF into path space. To compensate for this, the contribution function used to compute the acceptance probability for MH must now be divided by the sampling PDF. Therefore, as stated by Kelemen et al. [2002], the contribution function becomes:

$$C(\bar{u}) := \frac{\hat{p}(\bar{y}(\bar{u}))}{q(\bar{y}(\bar{u}))},$$

where $\hat{p}$ is the target PDF, q is the sampling PDF along which the path $\bar{y}$ is sampled from the random numbers $\bar{u}$. One consequence of this new contribution function is that the acceptance probability becomes flatter than it was before, resulting in more mutations getting accepted.

3.2.4 Mutating ReSTIR Paths

The ReSTIR PT paper [Lin et al., 2022] presented an improved unbiased variant of the original ReSTIR, but this requires capping the temporal confidence weight, which cannot simply be done in a scene agnostic way. Therefore it is difficult to find a cap that is large enough to sufficiently take advantage of temporal history, without introducing noticeable correlation artifacts. A potential solution is introduced by Sawhney et al. [2022]. The idea is to apply MCMC mutations in between the temporal and spatial resampling steps, so that any

bias that is carried over from previous frames is perturbed and mutated before contributions are spread out over neighbouring pixels.

To mutate the samples stored in ReSTIR's reservoirs, the reconnection vertex in the hybrid shift map is perturbed. This is a relatively cheap operation, as the reconnection vertex is already stored in the reservoirs, and only it requires retracing the two rays connecting this vertex to the rest of the path. Figure 2 shows how the reconnection vertex is perturbed.

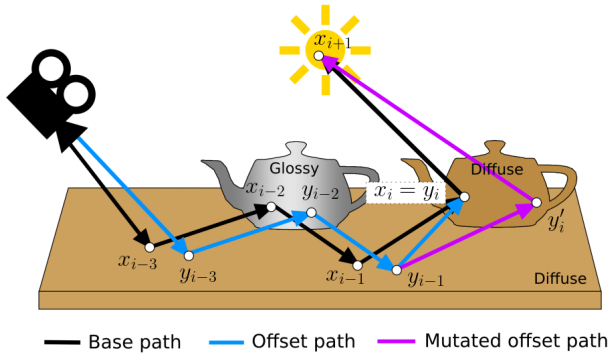


Figure 2: A mutated hybrid shift map. The reconnection vertex y_i is perturbed into y'_i.

In order to actually obtain the perturbed reconnection vertex, Sawhney et al. [2022] make use of the Primary Sample Space [Kelemen et al., 2002]. Specifically, they perturb the random numbers used for sampling the direction ω_{i-1} from y_{i-1} towards the reconnection vertex y_i. These random numbers are recovered by inverting the sampling procedure as per Bitterli et al. [2017].

To apply the new mutated sample to the reservoir, the reservoir's contribution weight needs to be adjusted. The new contribution weight is updated as follows:

$$W(x_k) = \frac{\hat{p}(x_0)}{\hat{p}(x_k)} W(x_0),$$

where x_0 is the base sample already stored in the reservoir, x_k is the final mutated sample, and $\hat{p}$ refers to the target function.

Since the choice for the Metropolis Hastings starting sample x_0 is now the result of resampling done by ReSTIR, the burn-in period of Metropolis-Hastings can be avoided [Veach, 1998].

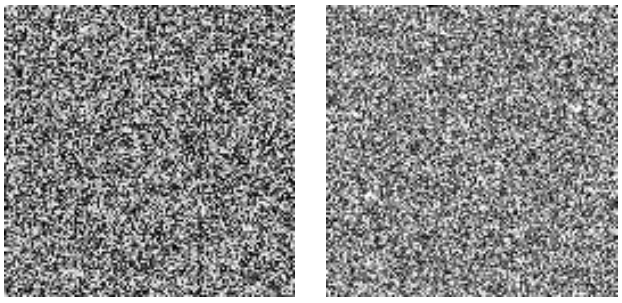
Each frame the decorrelated ReSTIR algorithm performs the following steps:

- First do the initial resampling, tracing a number of paths and storing the final selected sample in a reservoir.
- Perform temporal resampling. For each pixel combine its initialised reservoir with the reservoir of the previous frame of the pixel in that previous frame that corresponds to the current one. Which pixels correspond to which can be determined via motion vectors.
- In the next phase Metropolis Hastings is used to mutate the samples held in the reservoirs from the previous step.
- Then a spatial resampling pass is performed, where for each pixel a number of random neighbouring pixels are selected, to merge their reservoirs. Doing multiple iterations of this effectively increases the radius of pixels whose reservoirs are combined.
- Lastly the samples that are remain as the selected samples in the reservoirs are used to perform the final shading and compute each pixel's color.

3.3 Blue Noise

Blue noise refers to high frequency noise, as opposed to the more common white noise which encompasses all frequencies. This means that blue noise will see less clumps of similar values close to each other, instead similar values will be spread out and differing values are positioned close together. This is said to be more visually pleasing to the human eye. One explanation for this could be that the photoreceptors in our eyes are blue noise distributed [Yellott, 1983]. However, no clear definite explanation or justification has been put forward as of yet. Blue noise is also more easily filtered out by a denoiser, as a simple high frequency blur pass like a gaussian blur can remove high frequency noise.

The strength of a blue noise pattern can be visualised by computing the frequency of the pixel intensities with a discrete Fourier transform. If the noise in the image is indeed blue noise, then the DFT should show a lack of low frequencies, as seen in Figure 3, which is exactly what defines blue noise.



(a) A white noise mask (b) DFT of a white noise mask

(c) A blue noise mask (d) DFT of a blue noise mask

Figure 3: 128x128 noise masks and their respective power spectra. Blue noise lacks lower frequencies, which are plotted in the center of the DFT image. Images taken from Chizhov et al. [2022]

Blue noise patterns in computer graphics are commonly seen in two variants, sample point distributions and random noise masks. The former refers to a collection of points that

have random positions while still being relatively uniformly distributed over the space. The latter is more relevant in regards to ray tracing and can be used to generate pseudo-random numbers such that the noise that ends up in the ray traced image has a blue noise error distribution. Blue noise in this context was originally introduced for digital halftoning, where blue noise was applied to fuzzy the borders between colors in quantized images with a limited number of colors available [Ulichney, 1987; Lau and Arce, 2001]. Specifically the void and cluster algorithm for generating blue noise masks has been influential for the use of blue noise in ray tracing Ulichney [1993].

While having been applied in rendering before [Mitchell, 1991], blue noise dithered sampling (BNDS) [Georgiev and Fajardo, 2016], inspired by its use in digital halftoning, has recently sparked more interest in the application of blue noise in ray tracing. BNDS works through the use of blue noise masks, which are two dimensional masks consisting of blue noise distributed random vectors. These masks are tiled across the pixels of the screen and each vector is used as a source of random numbers to construct a path for the corresponding pixel. This essentially correlates the pixel estimates so that nearby pixels will differ from each other more than they would in a white noise mask.

Heitz and Belcour [2019] devise a method that scales better to higher sample counts by building on the concept of permuting sampling sequences. Ahmed and Wonka [2020] then make further improvements by taking inspiration from error diffusion. Chizhov et al. [2022] propose a method that unifies the two approaches of Georgiev and Fajardo [2016] and Heitz and Belcour [2019] into a formalized perceptual error framework. Salaün et al. [2022] then derives a bound for this perceptual error and propose a multi-class framework that can be applied to minimize it, resulting in a blue noise error distribution.

One drawback of these methods is that they neglect real time temporal noise. When taking time as a Z axis, a simple spatial blue noise error distribution still shows a white noise distribution along this Z axis. Wolfe et al. [2021] solve this problem with spatiotemporal blue noise masks. They also show that ensuring a temporal blue noise distribution is especially beneficial for temporal antialiasing or other filtering methods.

4 Methodology

To test the effectiveness of the blue noise addition, we first need to acquire a base implementation of the ReSTIR algorithm with MCMC decorrelation, as presented by Sawhney et al. [2022]. This implementation can then be built upon by incorporating blue noise techniques. The blue noise implementation can then be evaluated compared to the base implementation and compared to a noiseless reference. The noiseless reference can be obtained by running an unbiased and consistent path tracer for a sufficiently long time, so without any potential bias introduced by ReSTIR or MCMC techniques.

4.1 Interpreting the Decorrelation of ReSTIR

Paths can finish in two different ways, either through next event estimation (NEE), or by hitting a light source 'naturally' and escaping the path.

While not explicitly defined for indirect illumination, we adapt the formula for the sampling PDF q for a path segment as:

$$q(y_k \to y_{k+1}) := p_\rho(\omega_{k-1}, \omega_k) g(y_{k+1}).$$

Here p_ρ is the PDF for the BSDF ρ, and ω_k is the unit vector from vertex y_k to y_{k+1}. The geometry term g at y_k is defined as:

$$g(y_k) := \frac{|cos\ \theta|}{|y_k - y_{k-1}|^2},$$

where θ is the angle between ω_{k-1} and the geometric surface normal at y_k. Note that the geometry term at y_{k+1} is used for the PDF at y_k

We will refer to the reconnection vertex, whose position will get changed by the mutations, as y_i. The vertices that are one step in the path closer to the camera and one step further will be refered to as y_{i-1} and y_{i+1} respectively. In Figure 4 we show a schematic drawing of the relevant part of a given path.

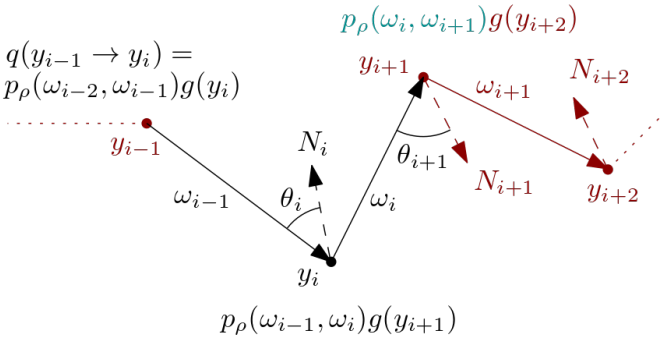


Figure 4: A schematic of the variables used to compute the sampling PDFs. The parts of the path that remain unaffected by the mutation are tinted red. In green is the factor that is ignored if y_{i+1} is the final vertex.

Since both vertex y_{i+1} and y_{i+2} are unaffected by the mutation, we do not need to recompute the geometry term for y_{i+2}. However, if y_{i+2} and by extension ω_{i+1} exist, we do need to recompute $p_\rho(\omega_i, \omega_{i+1})$, since this is affected by ω_i. As such the total sampling PDF that needs to be recomputed is:

$$q_m = p_\rho(\omega_{i-2}, \omega_{i-2}) g(y_i) p_\rho(\omega_{i-1}, \omega_i) g(y_{i+1}) p_\rho(\omega_i, \omega_{i+1}),$$

where the trailing $p_\rho(\omega_i, \omega_{i+1})$ is left out if y_{i+2} does not exist. We can use this as the sampling PDF for the contribution function, as the sampling PDF that remains constant for the rest of the path cancels itself out in the ratio of the contribution function used for computing the Metropolis Hastings acceptance probability. The same holds true for the target PDF.

Sawhney et al. [2022] show that the ratio of the transition functions that is used for the MH acceptance probability is as follows:

$$\frac{T(\bar{u}' \to \bar{u})}{T(\bar{u} \to \bar{u}')} = \frac{|cos\ \theta'|}{|cos\ \theta|} \frac{|y_{i+1} - y_i|^2}{|y_{i+1} - y_i'|^2} \frac{p(\omega'_{i-1}, \omega'_i)}{p(\omega_{i-1}, \omega_i)} \frac{p(\omega'_i, \omega_{i+1})}{p(\omega_i, \omega_{i+1})}.$$

Here θ is the angle between ω_i and the surface normal at y_{i+1}.

We need to choose one of the BSDF components when inverting a path segment to the random numbers used to generate it. To do this, one option is to keep using the same

sampling strategy as is used for the base reconnection vertex, which is stored in the reservoir. This ensures consistency and minimises the chance of error. Another option is to use a reversible jump, as presented by Bitterli et al. [2017], to guess what sampling strategy is most plausible. This might be beneficial as it is another more effective way to mutate the path's sampling strategy beyond mutating the random number responsible for choosing the sampling strategy. We have evaluate our implementation using the first option to keep the mutation process as simple as possible, though we ended up implementing both options.

4.2 Blue Noise

To adjust the decorrelation algorithm to achieve a more high frequency error distribution closer to blue noise, we follow the the remarks made by Sawhney et al. [2022] in their future work section. They suggest that the mutations might be able to optimise for blue noise by also taking neighbouring pixel values into consideration when computing its acceptance probability, preferring mutations that introduce differing sample values.

To compute when a mutation will contribute to a blue noise error distribution, we take inspiration from Georgiev and Fajardo [2016]. They create blue noise sample masks by minimising a blue noise energy function of the entire mask. They define this as the total sum of blue noise energy for each pair of pixels i and j, which in turn is defined as follows:

$$E(i,j) = exp\left(-\frac{||i_c - j_c||^2}{\sigma_c^2} - \frac{||i_s - j_s||^{d/2}}{\sigma_s^2}\right),$$

where i_c and j_c are the integer pixel coordinates, i_s and j_s are the sample values, and σ_c and σ_s are Gaussians that are respectively set to 2.1 and 1. d refers to the dimensionality of the sample space. Whereas Georgiev and Fajardo used this energy function for creating a blue noise sample mask, we use it directly on the pixel light intensities. We convert the pixel color to scalar intensity, so that we can compute the blue noise energy over a 1-dimensional sample space.

Georgiev and Fajardo apply simulated annealing to swap pixels around while minimising this energy function. Instead, when deciding whether to accept a mutation, we compute the average blue noise energy with a number of neighbouring pixel within a given box radius. We compute this both for the base pixel luminance and the mutated pixel luminance.

These two energy values can then be used in the standard simulated annealing acceptance probability as formulated by Kirkpatrick et al. [1983]:

$$a_{bn} = \begin{cases} 1, & \text{if } e' < e \\ exp\left(-\frac{(e'-e)}{T}\right), & \text{otherwise} \end{cases}$$

Here e and e' are the base and mutated energies, respectively. T refers to the temperature, which usually decreases as simulated annealing progresses, in order to gradually reduce the chance of accepting increases in the energy function. Since our algorithm needs to produce blue noise in real time, we fix to a low value in an effort to greedily approach a blue noise error distribution.

To actually integrate this blue noise acceptance probability into the existing acceptance probability that targets the target PDF, we replace the ratio of the contribution function with a weighted average of this ratio and a_{bn}:

$$a(x_k \to x'_k) = \left(a_{bn} w_{bn} + \frac{C(x'_k)}{C(x_k)}(1 - w_{bn})\right) \frac{T(x'_k \to x_k)}{T(x_k \to x'_k)},$$

where w_{bn} is the weight for preferring blue noise, in the range of $[0, 1]$. We leave the transition kernels out of the weighted average, as these only function as a way to guarantee the detailed balance condition.

5 Implementation

The source code of ReSTIR PT is publicly available, whereas the code for the additional decorrelation via MCMC mutations is not. Therefore this decorrelation feature must be implemented by ourselves. This open source ReSTIR implementation makes use of GPGPU and is built with Falcor [Kallweit et al., 2022], which is a popular real-time rendering framework made by NVIDIA that aims to abstract away many operations relevant to graphics. While this might speed up the implementation process once the framework is learned, it also might mean less freedom in the optimisation of the code. An alternative might be reimplementing ReSTIR PT in a less complex framework like Lighthouse 2 and working more directly with CUDA. However, considering both time constraints and convenience, we opt to simply move forward with the existing implementation in Falcor.

5.1 ReSTIR PT

ReSTIR PT is implemented as a render pass for Falcor, which are organised in separate projects within the Falcor's Visual Studio solution. Each frame the render pass executes its kernels to update the pixel buffer.

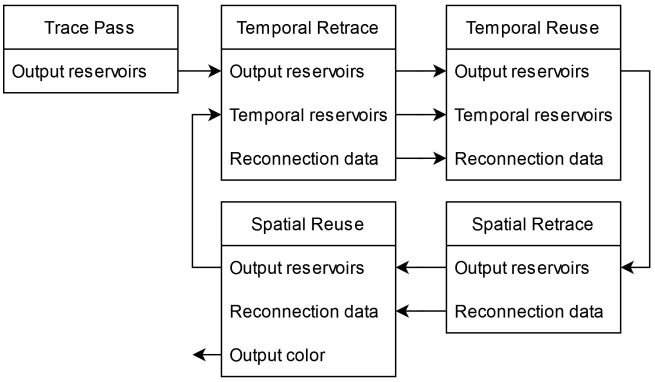


Figure 5: An overview of the different kernel passes used in ReSTIR PT.

First the main trace pass kernel gets executed, which samples and traces a new light path for each pixel. These samples are used as the initial samples stored in the reservoir structs, as shown in Figure 1. These reservoirs also store information for potential reconnection vertices (rcVertex) corresponding to the selected sample, which can later be used for reconnection shifts. Note that in this implementation, the final vertex

Algorithm 1 Reservoir struct for ReSTIR PT (simplified for clarity)

```
struct PathReservoir
{
    float M;       // temporal confidence weight
    float w_sum;
    float weight;
    bool specularRc;
    bool specularPrev;
    bool transmissiveRc;
    bool transmissivePrev;
    bool rcVertexLength;
    bool pathLength;
    bool lastVertexNEE;  // whether the path ended via NEE
    float3 F;            // cached integrand / target PDF
    float lightPdf;      // NEE light PDF
    float sourcePdfPrev; // scatter PDF for y_{i-1}
    float sourcePdfRc;   // rc vertex scatter PDF
    float geomRc;        // geometry term at rc vertex
    uint initRandomSeed; // saved random seed at first bounce
    uint rcRandomSeed;   // saved random seed after rc Vertex
    // position and surface normal of the reconnection vertex
    HitInfo rcVertexSd;
    // incident direction on reconnection vertex
    float3 rcVertexWi;
    // sampled irradiance on reconnection vertex.
    // oscar: the total thp starting with rcNext
    float3 rcVertexIrradiance;
}
```

on a light source will never actually be marked as a reconnection vertex.

The initRandomSeed variable is necessary for the random replay shift, whereas the rcRandomSeed variable is used in case we want to retrace the part of the path from a temporal sample that is reconnected to. This is mainly useful to do when rendering a dynamic scene with moving objects that might affect the lighting measured in previous samples.

The reservoir also stores several flags, for example for whether the bounces at y_{i-1} and y_i were specular. It also stores the length of the path and the number of vertices before the selected potential reconnection vertex[1]. Whether the path was finished via NEE is also stored as a flag.

After the reservoirs have been built, it is followed up by a temporal retrace pass. This pass uses both the newly created reservoirs and a buffer of 'temporal' reservoirs from the previous frame. It traces a random replay path up to the reconnection vertex, and stores some reconnection data needed to reconnect this path to an older sample. As shown in Algorithm 2, this data includes the position and outgoing direction from y_{i-1} to y_{i-2} and the target PDF for the path up to the reconnection vertex.

Algorithm 2 Reconnection data used for reconnection shift in ReSTIR PT

```
struct ReconnectionData
{
    float3 prevPosition; // position of y_{i-1}
    float3 prevWo;       // outgoing direction from y_{i-1} to y_{i-2}
    float pathTargetPdf; // target PDF up to rc vertex
}
```

This information is passed to the subsequent temporal reuse pass, which attempts to finish the hybrid shift by now using the reconnection data to reconnect the temporal replay path to the temporal reservoir. Therefore the temporal reservoir is shifted towards the domain of the new reservoir.

Now that the temporal resampling is done, the algorithm moves on to a spatial retrace pass. Once again this pass performs the random replay part of the hybrid shift, but now we use several neighbouring pixel's reservoirs rather than a temporal reservoir. The same goes for the subsequent spatial reuse pass, except in this pass the resulting shifted and merged reservoirs are finally used to compute each pixel's color.

5.2 Reproducing the Decorrelation

To implement the decorrelation of this algorithm, Sawhney et al. [2022] add a decorrelation pass in between the temporal reuse and spatial retrace passes. Since correlation artifacts occur when the same sample is spread around via spatial resampling, decorrelating right before helps diversify those samples.

Figure 6 shows how the decorrelation pass relates to the temporal and spatial resampling passes. It needs to use the reconnection data (Algorithm 2) saved by the temporal reuse pass to have access to y_{i-1} without having to retrace the path.

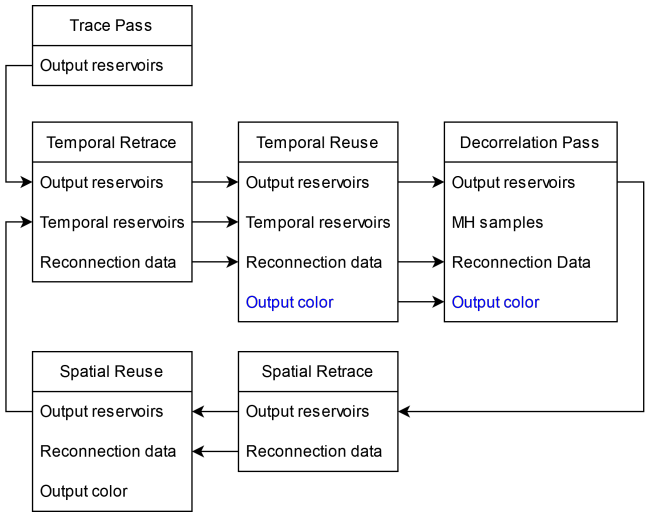


Figure 6: An overview of how the decorrelation pass fits in between the ReSTIR PT passes. The output color in blue is only used for blue noise.

In Algorithm 3 we show the way the mutation algorithm works on a higher level. It starts an initialisation step, followed by a number of iterations of computing a mutation and determining whether to accept it, and then finally applying the mutation to the reservoir. Separating the computation of the mutated sample and the computation of the acceptance probability is important in regards to the cohesion of the program, and facilitates implementing the blue noise acceptance probability later on.

5.2.1 Initialisation

The initialisation mainly consists of recomputing some characteristics of the existing path. We define two different structs, as shown in 4: SampleMHInit for variables that

[1] In the actual ReSTIR PT implementation, path length excludes both the first vertex (that directly connects to the camera), and last vertex (on a light source).

Algorithm 3 High level code structure for mutating the reservoirs.

```
Input : Reservoir r and the number of MH iterations
Output : The potentially mutated reservoirs
function metropolisHastingsPT(r, iterCount)
    s_b = ∅ // Base sample
    s_m = ∅ // Mutated sample
    accept = false
    initialiseMH(s_b, sampleMHInit, r)
    for iterCount iterations:
        s_m = computeSampleMut(s_b, sampleMHInit, r)
        accept = acceptSampleMut(s_b, s_m, sampleMHInit, r)
        if accept
            s_b = s_m
    if accept
        updateReservoirMut(s_b, sampleMHInit, r)
```

are only initialised once, and SampleMH for the variables that get updated by the mutation. This way we can use two instances of the SampleMH struct for the base and the mutated values.

Algorithm 4 The SampleMHInit and SampleMH structs which respectively contain the variables that remain constant and the variables that are changed by the mutation.

```
struct SampleMHInit {
    float3 startingTargetPdf;
    float startingSourcePdf;
    bool rcIsSecondToFinalVertex;
    bool rcVertexNEE;
    float3 rcNextPos;
    // position and outgoing direction of y_{i-1}
    ShadingData rcPrevVertexSd;
}

struct SampleMH {
    float4 samplePSSPrev;
    float sourcePdfPrev;
    float sourcePdfRc;
    float sourcePdfNext;
    float sourcePdf;
    float3 targetPdf;
    float3 rcPrevVertexWi;
    float3 rcVertexWi;
    bool isTransmissionPrev;
    bool isTransmissionRc;
    bool isSpecularPrev;
    bool isSpecularRc;
    bool isSpecularNext;
    // position, surface normal, and outgoing direction of y_i
    ShadingData rcVertexSd;
    // position, surface normal,  and outgoing direction of y_{i+1}
    ShadingData rcNextVertexSd;
}
```

After ensuring that a valid y_i exists, we first initialise the shading data, which includes a vertex's position, surface normal, and outgoing direction, for the reconnection vertex (y_i), and for both the vertex before and after that (y_{i-1} and y_{i+1} respectively). We obtain information for y_{i-1} from the reconnection data[2] (Algorithm 2), and for y_i from the reservoir.

To further initialise these structs, we need more information than is originally stored in the reservoirs in the base implementation of ReSTIR PT. We show the variables we add to the reservoir struct in Algorithm 5. For one, the vertex y_{i+1} is not stored in the reservoir, as it is not relevant to the reconnection shift, which connects y_{i-1} to y_i. However, we do need y_{i+1} to reconnect it to the mutated y_i'. Therefore, just as is done by Sawhney et al. [2022], we store it in the reservoir, so that we can use it to initialise its shading data. We make sure to save it during the initial path construction, whenever we hit a surface while the previous vertex is an rcVertex. One caveat with this is that we encounter different surface hit data when we perform NEE, as opposed to simply hitting a triangle mesh surface. If y_{i+1} is indeed a point on a light source, sampled through NEE, then we only need to know this position and its surface normal. Therefore we store both of these as separate float3 variables, on top of the HitInfo we save if y_{i+1} is a regular surface, as shown in Algorithm 5. There would be room for future optimisation here if necessary, since we never use both of these at the same time.

In addition to this, we also need store the incoming direction ω_{i+1} from y_{i+1} towards y_{i+2}[3], since we need it to recompute the sampling PDF for y_{i+1}. We would not need to do this if we only needed to compute the diffuse component of this PDF, since it only depends on ω_{i+1}, which is unaffected by the mutation. However, the specular component depends on how the outgoing direction relates to the incoming direction, so it does require us to store ω_{i+1}. To save this variable in the reservoir, we update it during the path tracing process whenever handle a vertex hit while the previous vertex was the reconnection vertex. One again we need to make an exception for NEE, if we have sampled the light from y_{i+1}. In this case we can simply store the direction towards the light when it gets sampled through NEE.

Note that we might be deviating here from Sawhney et al. [2022], as they only mention storing the offset vertex y_i in addition to what is already stored in the reservoir for ReSTIR PT. On the other hand they do mention that ω_{i+1} is required for computing the transition kernels, while also implementing the decorrelation algorithm as a separate block from the larger ReSTIR algorithm.

Algorithm 5 Variables added to reservoir for decorrelation

```
struct PathReservoir // 128 Bytes
{
    ...
    // position and surface normal of y_{i+1},
    // if not sampled via NEE
    HitInfo rcNextHit;
    // position of y_{i+1} as the point on the light source that
    // is selected via NEE
    float3 posNEE;
    // If y_{i+1} is sampled via NEE, then this is the
    // normal on the light source. Otherwise this is the
    // incident direction ω_{i+1} on y_{i+1}.
    float3 rcVertexNextWiOrN;
}
```

We can then use the data for these three consecutive vertices to initialise additional data about this part of the path. First we take the path segment from y_{i-1} to y_i and invert it into the four random numbers used to generate it.

We also convert the path segment from y_{i+1} to y_{i+2} to PSS, in order to determine whether to treat it as a specular bounce

[2] An implementation-specific exception to this is when the reconnection vertex length is 1, in which case y_{i-1} would be the first vertex directly connected to the camera. In this case we need to use the primary shading data, which is the shading data corresponding to this very first vertex, and always gets passed separately from the reconnection data.

[3] While the code is largely unoptimised due to time constraints, one minor, potentially superfluous optimisation we made here is combining the incoming direction and the NEE light surface normal for y_{i+1} into one variable, since we only use the latter if y_{i+1} is sampled through NEE, in which case there is no incoming direction left.

or not. We can use the cachedJacobian for $p_\rho(y_{i-1})$, $p_\rho(y_i)$ and $g(y_i)$. An exception is if y_{i+1} is sampled through NEE, in which case we need to recompute $q(y_i)$. We do need to (re)compute $\hat{p}(y_{i-1})$, $\hat{p}(y_i)$, $g(y_{i-1})$ and $p_\rho(y_{i+1})$. We multiply the partial target PDFs for each vertex together to get a total target PDF, and multiply the sampling PDFs together for a total sampling PDF as well.

5.2.2 Computing the Mutation

After initialisation, we can perform a number of iterations of metropolis-hastings. One such iteration starts with computing the integrand for the mutated reconnection vertex. To do this, we first apply a gaussian mutation to the four random numbers in PSS we have saved, corresponding in path space to the path segment from y_{i-1} to y_i. We immediately convert the mutated numbers back to path space to obtain the mutated direction for the bounce from y_{i-1}. We also then compute the throughput and sampling PDF for this bounce. If the mutated PSS numbers do not correspond to a valid path, or the sampling PDF is nearly equal to 0, we cancel the mutation and skip an iteration. The restriction on the sampling PDf is to prevent floating point errors. Next we trace a ray in the mutated direction to find y_i'. We then initialise the shading data for this mutated reconnection vertex. In the following step we trace an additional ray to evaluate the visibility of the segment between y_i' and y_{i+1}. We also invert this segment to PSS, in order to determine whether to treat it as a specular bounce.Then we can compute the mutated throughput and sampling PDF for this path segment and for the one following it. We compute the updated total target PDF and sampling PDF. We cancel the mutation if either of these is below some epsilon, to prevent floating point errors.

5.2.3 Accepting the Mutation

Once the difference in throughput and sampling PDF caused by a mutation has been computed, we can determine whether we want to accept the mutation or not. We rewrite the formula for the acceptance probability to be slightly different from Sawhney et al. [2022], to prevent floating point inaccuracies.

$$a(u \to u') = \frac{p(y(u'))}{p(y(u))} * \frac{q(y(u))}{q(y(u'))} * \frac{T(u' \to u)}{T(u \to u'}$$

If we accept the mutation based on this probability, we replace the base SampleMH struct instance with a copy of the mutated instance.

5.2.4 Updating the Reservoir

If we have accepted a mutation in one of the iterations, we update the reservoir.

We multiply the reservoir's weight by the ratio of the mutated target PDF over the base target PDF, after first converting both of these to grayscale scalar values. Note that we do not compute the target PDF for the entire path, but only for the part of the path that is affected by the mutation. The constant part of the target PDF cancels itself out when computing the ratio, so it can be ignored.

We similarly multiply the cached integrand by the ratio of the base over the mutated target PDF.

We also update each part of the cachedJacobian and the flags for whether the bounces at y_{i-1} and y_i are specular.

5.2.5 PSS Inversion

We have implemented functions to invert a sampled direction to the random numbers used to sample it. We have mostly based the process on the inverse mappings as presented by Bitterli et al. [2017], though the concept of switching between path and sample space goes back to Kelemen et al. [2002]. Since we want to support BRDFs that consist of both specular and diffuse components, we implement Bitterli et al's reversible jump.

The actual sampling method of diffuse reflections is based on concentric mapping, the inverse for which we adapt from Shirley and Chiu [1997]. As we weren't able to find an inverted method for GGX sampling [Walter et al., 2007], which was used for specular/glossy reflections, we derived this ourselves.

5.2.6 Direct lighting

We also implemented the decorrelation algorithm for ReSTIR DI before working with ReSTIR PT, however the structure and implementation of the ReSTIR DI algorithm turned out to be so different from the ReSTIR PT implementation that we opted to shift our focus solely to the latter. The effect of the mutations on ReSTIR DI will naturally be less strong, as the paths are per definition much less complex.

5.3 Blue Noise

When optimising for blue noise, we need to take neighbouring samples into consideration. Specifically we need to know the light intensity of each sample to calculate the blue noise energy of the selected neighbours. To prevent having to recompute this for every neighbour, we need access to a precomputed array of the light intensity of each pixel. As portrayed in Figure 6, we solve this by computing the output color in the temporal reuse pass, and passing this on to the decorrelation pass.

Algorithm 6 Pseudocode for handling blue noise.

```
function metropolisHastingsPT(r, iterCount, color)
    initialiseMH(s_b, sInit, r)
    for iterCount iterations:
        s_m = computeSampleMut(s_b, sInit, r)
        a_bn = computeBlueNoise(s_b, s_m, sInit, r, color)
        accept = acceptSampleMut(s_b, s_m, sInit, r, a_bn, w_bn)
        if accept
            s_b = s_m
    if accept
        updateReservoirMut(s_b, sInit, r)

function computeBlueNoise(s_b, s_m, sInit, r, c_b, n_nb)
    // Compute mutated color
    c_m = c_b * mutF / oldF * toScalar(oldF) / toScalar(mutF)
    bnEnergyBase = 0
    bnEnergyMut = 0
    for n_nb pixels in a given radius:
        // Use color and pixel location to compute bn energy
        bnEnergyBase += blueNoiseEnergy(c_b, c_nb, p_b, p_nb)
        bnEnergyMut += blueNoiseEnergy(c_m, c_nb, p_b, p_nb)
    // Obtain average bn energies
    bnEnergyBase = bnEnergBase / sampledNeighbourCount
    bnEnergyMut = bnEnergyMut / sampledNeighbourCount
    // Always accept lower mutated bn energy, sometimes higher
    bnDiff = bnEnergyMut - bnEnergy
    return bnDiff >= 0 ? exp(-bnDiff / T) : 1
```

As shown in Algorithm 6, we compute the color that the

mutated path would contribute by taking the adjustments that we would apply to the reservoir's weight and target PDF if we accept the mutation, and applying these to the base color. We can then use these two color values and compute the average blue noise energy between them and a number of their neighbours. Finally we compute the blue noise acceptance probability based on these blue noise energies.

One drawback of our implementation is that we do not update the pixel luminance in between the MH iterations, to avoid dealing with concurrency issues. This means less effectiveness for the blue noise energies, as they are not based on how a mutation changes the blue noise energy from its base sample, but compared to the very first sample that is still stored in the reservoir.

6 Results and Discussion

These base and the blue noise implementations are compared in terms of the amount of noise and the distribution of noise they produce, and how much time it takes them to produces a frame, given the same number of samples per pixel. To properly compare the ray tracers, they need to produce images of the same scene, using the same parameters for the camera and such. Furthermore, to reduce the effect of outliers, we average our metrics over a certain number of images, which we have set to 10. To obtain these images, we let the ray tracer run for 50 frames, save the pixel luminance (before it gets sent to the tonemapper), reset the reservoirs, and repeat the process until we have saved 10 images. The resulting EXR files are then imported and analysed in Matlab.

We set the temporal confidence weight cap to 50, since the benefit of decorrelated ReSTIR is being able to make more use of temporal history without the large correlation artifacts. Furthermore we have used an adapted version of the Veach Ajar scene that features surfaces that are mostly diffuse, as we encountered issues with scenes with too many strongly specular surfaces that we haven't had the opportunity to resolve. In part due to this we also closed the door slightly more, to create more hard to sample light-carrying paths, which become less common if there are more diffuse surfaces. And lastly the camera is aimed at the ceiling, where the correlation artifacts caused by hard to sample paths are more clearly visible.

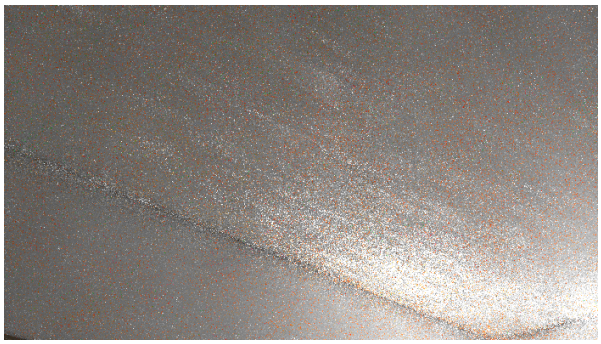
For the blue noise implementation we use a neighbour count of 8 and radius of 4. We set the blue noise weight to 1.

In Figure 7 we show a cropped example of the reduction in correlation artifacts when using the same seed. While not too noticeable, performing 5 mutations does result in less correlation artifacts in the form of streaks across the ceiling, parallel to the slit in the door through which light enters the room.

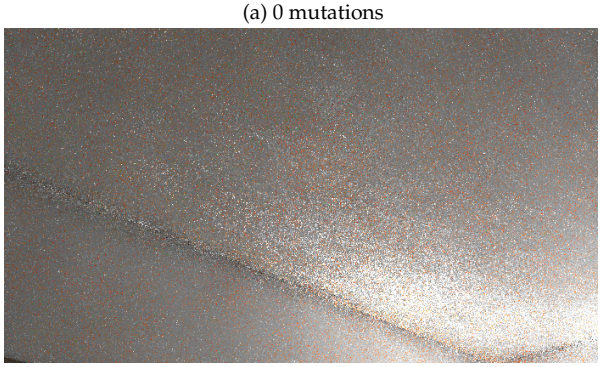
6.1 Error

The amount of error or noise can be measured with metrics such as RMSE, SSIM [Wang et al., 2004] or FLIP [Andersson et al., 2020]. RMSE is simply average difference in pixel values compared to a reference image. Specifically, RMSE is defined as:

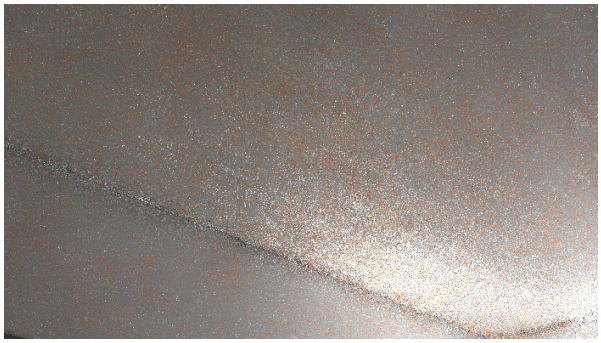
$$RMSE = \sqrt{\frac{\sum_{m=1}^{K} \left(I_m - \hat{I}_m\right)^2}{K}},$$



(a) 0 mutations



(b) 5 mutations



(c) 5 mutations (blue noise)

(d) Reference image

Figure 7: Cropped frames of the ceiling in Veach Ajar with 0 vs 5 mutations, using the same seed. Exposure is manually turned down for visibility.

Scene	Covariance 0 mutations	Covariance 5 mutations	Covariance 5 mutations (blue)	Time (ms) 0 mutations	Time (ms) 5 mutations	Time (ms) 5 mutations (blue)
Veach Ajar (ceiling)	3.5208e-04	3.0106e-04	2.938e-04	17	22	24

Table 2: Measured relative covariance with pixel radius 8, based on ReSTIR PT with a hybrid shift.

where K is the total number of images, I_m is the reference pixel luminance, and $\hat{I}_m$ is the estimated pixel luminance. SSIM is slightly more advanced and less absolute, looking at the structure between pixels as well. FLIP takes it a step even further and aims to measure the perceptual difference for humans compared to the reference image. However, the amount of noise in the image renders is usually not very relevant when evaluating the difference between renders with white noise versus blue noise, as the amount of error in the image is expected to be the same, just at different frequencies. Confirming that this is indeed the case can sufficiently be done with RMSE, though it might be interesting to examine whether FLIP picks up a difference between blue and white noise.

As shown in Table 1, we do see a slight increase in error when applying mutations, as opposed to the slight decrease reported by Sawhney et al. [2022]. This could be simply an inaccuracy due to a low measured frame count, or due to a quirk in the different scene and parameters we use. If not, then in theory we wouldn't expect a higher error if Metropolis Hastings is properly implemented.

Scene	RMSE 0 mutations	RMSE 5 mutations	RMSE 5 mutations (blue)
Veach Ajar (ceiling)	0.1019	0.1057	0.1026

Table 1: Measured root-mean-square error.

6.2 Correlation

Another relevant metric is the relative pixel covariance, used to measure correlation. This was already used by Sawhney et al. [2022] to measure the effectiveness of their algorithm at reducing correlation artifacts in ReSTIR. The covariance between two pixels i and j in an image I is given by:

$$c_{ij} = \frac{1}{K-1} \sum_{m=1}^{K} (I_{mi} - \bar{I}_i)(I_{mj} - \bar{I}_j),$$

where K is the number of images and $\bar{I}$ is the average of K images. The total covariance of the image is then computed by taking the average of for each pixel i the average covariance with all the pixels j within their neighbourhood of a box with a given radius r (which we set to 8).

As shown in Table 2, we have achieved a similar reduction in correlation artifacts as Sawhney et al. [2022] did, though it should be noted that our adjustments to the scene make the results difficult to compare one to one. Our blue noise implementation appears to achieve slightly lower covariance than the base decorrelation, but considering our the relatively small number of frames we have computed this over, this difference may very well be insignificant.

An observation we made while testing is that mutations can fall short when the correlation artifacts are caused by light coming in through a slit. In this case spatial resampling can fail to help neighbouring pixels perpendicular to the slit to find this source of light, as the light vertex becomes obstructed by the shift. If the light source is large, it could mean that there is still light reaching the neighbouring pixel via the slit, but at a different angle than can be found via a hybrid shift. At the same time, sample is easily spread out parallel to the slit. This can result in streaks of bright spots parallel to the slit, which is also slightly visible in Figure 7a. Mutations lose their effectiveness in these cases, compared to for example correlation artifacts caused by glossy bounces, as they will attempt to move the reconnection vertex around across the surface, but only moving it parallel to the slit will prevent the light source vertex from getting obstructed.

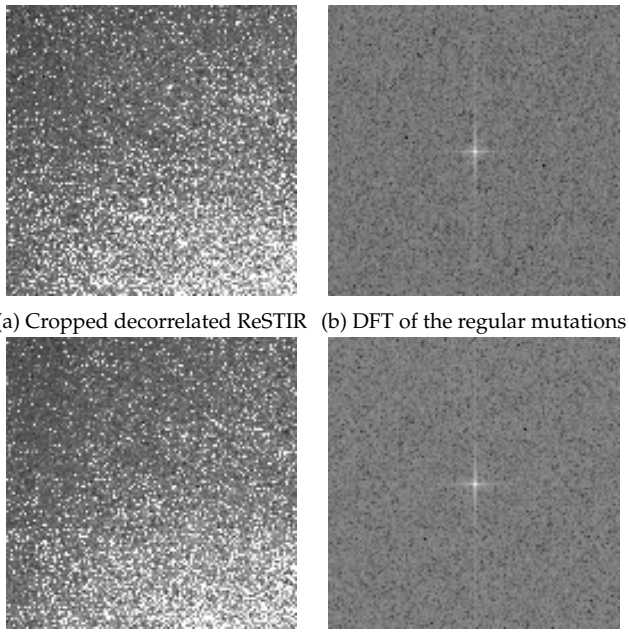
6.3 Blueness of noise

To visualise how the error is distributed in an image, we compute the discrete Fourier transform (DFT) of the image. In the case of blue noise this should result in fewer lower frequencies compared to higher frequencies. The DFT is usually visualised as a square image whose axes represent the frequency of noise, with the middle of the image as the origin. For white noise the frequencies would be random, but for blue noise the middle of the image would be darker, as this means less low frequency noise compared to high frequency noise.

This begs the question of at which point our path tracer will produce noise that we deem sufficiently blue. There unfortunately do not seem to be any studies that look into how well the improved perception of blue noise path traced images scales up as the error distribution becomes more blue. Since there seems to be no clear defined point at which perceptual improvements become negligible, we could instead simply compare a few levels of blueness to each other by varying the amount of mutations and the degree to which their acceptance probability skews towards a blue noise distribution. Unfortunately, as evident by the DFTs corresponding to frames produced by the basic decorrelation algorithm and the blue noise version shown in Figure 8, we were unable to succeed in using the mutations to produce blue noise distributed error.

This lack of blue noise may be in part because its acceptance probability becomes less effective over multiple iterations, as we do not update the pixel luminances between MH iterations.

Another factor is the structure of the render passes. The noise produced by our decorrelation pass gets fed into the spatial resampling pass, which essentially undoes the high frequency distribution. Samples that differ from each other and are nearby each other are likely to get overridden by the highest sample, thereby shifting the samples back from being negatively correlated to being positively correlated. The effect that the spatial resampling pass has on the blue noise is



(a) Cropped decorrelated ReSTIR (b) DFT of the regular mutations

(c) Cropped blue noise frame (d) DFT of blue noise mutations

Figure 8: DFTs of zoomed in part of the ceiling in Veach Ajar with 5 mutations, using the same seeds.

similar to how blue noise interacts with a denoiser; the high frequency noise gets mostly filtered out, leaving only low frequency noise. In theory this would still preferable to feeding regularly distributed error to the spatial resampling pass.

Unfortunately displaying the pixel values of the reservoirs directly after the decorrelation pass does not result in blue noise either. Moreover, as mentioned before, executing the decorrelation pass after spatial resampling would reduce its effectiveness in diversifying the samples. It is possible the mutations are simply not able to do enough to shift the error distribution from being positively correlated to being negatively correlated. As shown by Sawhney et al. [2022], the mutations contribute very little to reduce the overall error, meaning that the algorithm relies on spatiotemporal resampling to find important paths, but not on the mutations. As a result, the mutations will most often be applied to high-contribution paths, diversifying them into slightly lower-contribution paths, rather discovering high-contribution paths from less valuable paths. Because of this, using mutations to optimise for blue noise is inherently limited by for the most part only being able to do this by rejecting a sample's mutation that decreases light intensity if its neighbours are already darker. Combining this decreased effectiveness with already positively correlated samples means approaching a blue noise error distribution becomes an uphill battle.

Unfortunately due to time constraints we were unable to test if our implementation is capable of producing blue noise when executed independently from the rest of the algorithm.

7 Conclusion

Our main contribution has been reimplementing the decorrelation of ReSTIR via MCMC mutations, as put forward by Sawhney et al. [2022]. We have provided our interpretation of how their algorithm is best implemented, and have found somewhat similar results in our evaluation.

We were not able to adjust the mutations into producing more high-frequency distributed error, akin to blue noise. Despite this, it may be valuable to now know the limitations of blue noise mutations when combined with ReSTIR. MCMC mutations do not appear to be the most efficient method of creating blue noise, because they are not able to match the quality of the paths already found by ReSTIR.

ReSTIR's spatiotemporal resampling also actively works to reduce blue noise and increase low frequency noise in the form of correlation artifacts. Because of this, vying for blue noise may still be beneficial to ReSTIR, but it will be difficult to actually produce a blue noise error distribution.

8 Future Work

More analysis into exactly why our algorithm fails at producing even a hint of blue noise, and whether it produces blue noise when executed in isolation would be useful.

Ensuring that new samples are blue noise distributed, by for example using a blue noise mask, might be beneficial to our algorithm. It might again be cancelled out by the resampling, but will likely at least help combat correlation artifacts.

As such, it might also be interesting to combine the findings of Heitz and Belcour [2019] concerning reordering pixel values of the previous frame to achieve blue noise, with the spatiotemporal reuse of ReSTIR PT. Especially applying shift maps to the reordered pixel samples rather than simply swapping them around in image space might be enough to provide an alternate way of reducing correlation artifacts in ReSTIR. This reordering step might then somehow be merged with the spatial resampling step to minimise the performance hit.

References

Ahmed, A. G. M. and Wonka, P. (2020). Screen-space blue-noise diffusion of monte carlo sampling error via hierarchical ordering of pixels. *ACM Trans. Graph.*, 39(6).

Andersson, P., Nilsson, J., Akenine-Möller, T., Oskarsson, M., Åström, K., and Fairchild, M. D. (2020). Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2).

Bashford-Rogers, T., Santos, L. P., Marnerides, D., and Debattista, K. (2021). Ensemble metropolis light transport. *ACM Trans. Graph.*, 41(1).

Bitterli, B., Jakob, W., Novák, J., and Jarosz, W. (2017). Reversible jump metropolis light transport using inverse mappings. *ACM Trans. Graph.*, 37(1).

Bitterli, B. and Jarosz, W. (2019). Selectively metropolised monte carlo light transport simulation. *ACM Trans. Graph.*, 38(6).

Bitterli, B., Wyman, C., Pharr, M., Shirley, P., Lefohn, A., and Jarosz, W. (2020). Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4).

Cappé, O., Guillin, A., Marin, J. M., and Robert, C. P. (2004). Population monte carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929.

Chao, M. T. (1982). A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656.

Chizhov, V., Georgiev, I., Myszkowski, K., and Singh, G. (2022). Perceptual error optimization for monte carlo rendering. *ACM Trans. Graph.*, 41(3).

Doucet, A., Smith, A., de Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Information Science and Statistics. Springer New York.

Fajardo, M., Hanrahan, P., Jensen, H., Mitchell, D., Pharr, M., and Shirley, P. (2001). State of the art in monte carlo ray tracing for realistic image synthesis.

Georgiev, I. and Fajardo, M. (2016). Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, New York, NY, USA. Association for Computing Machinery.

Ghosh, A., Doucet, A., and Heidrich, W. (2006). Sequential Sampling for Dynamic Environment Map Illumination. In Akenine-Moeller, T. and Heidrich, W., editors, *Symposium on Rendering*. The Eurographics Association.

Hachisuka, T., Kaplanyan, A. S., and Dachsbacher, C. (2014). Multiplexed metropolis light transport. *ACM Trans. Graph.*, 33(4).

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

Heitz, E. and Belcour, L. (2019). Distributing monte carlo errors as a blue noise in screen space by permuting pixel seeds between frames. *Computer Graphics Forum*, 38.

Kajiya, J. T. (1986). The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 143–150, New York, NY, USA. Association for Computing Machinery.

Kallweit, S., Clarberg, P., Kolb, C., Davidovič, T., Yao, K.-H., Foley, T., He, Y., Wu, L., Chen, L., Akenine-Möller, T., Wyman, C., Crassin, C., and Benty, N. (2022). The Falcor rendering framework. https://github.com/NVIDIAGameWorks/Falcor.

Kelemen, C., Szirmay-Kalos, L., Antal, G., and Csonka, F. (2002). A simple and robust mutation strategy for the metropolis light transport algorithm. *Comput. Graph. Forum*, 21.

Kirk, D. and Arvo, J. (1991). Unbiased sampling techniques for image synthesis. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 153–156, New York, NY, USA. Association for Computing Machinery.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Lai, Y.-C., Liu, F., and Dyer, C. R. (2009). Physically-based animation rendering with markov chain monte carlo.

Lau, D. and Arce, G. (2001). *Modern Digital Halftoning, Second Edition*. Signal Processing and Communications. Taylor & Francis.

Lin, D., Kettunen, M., Bitterli, B., Pantaleoni, J., Yuksel, C., and Wyman, C. (2022). Generalized resampled importance sampling: Foundations of restir. *ACM Trans. Graph.*, 41(4).

Lin, D., Wyman, C., and Yuksel, C. (2021). Fast volume rendering with spatiotemporal reservoir resampling. *ACM Trans. Graph.*, 40(6).

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. 21(6):1087–1092.

Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341.

Mitchell, D. P. (1991). Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 157–164, New York, NY, USA. Association for Computing Machinery.

Otsu, H., Hanika, J., Hachisuka, T., and Dachsbacher, C. (2018). Geometry-aware metropolis light transport. *ACM Trans. Graph.*, 37(6).

Ouyang, Y., Liu, S., Kettunen, M., Pharr, M., and Pantaleoni, J. (2021). Restir gi: Path resampling for real-time path tracing. *Computer Graphics Forum*, 40(8):17–29.

Rioux-Lavoie, D., Litalien, J., Gruson, A., Hachisuka, T., and Nowrouzezahrai, D. (2020). Delayed rejection metropolis light transport. *ACM Trans. Graph.*, 39(3).

Salaün, C., Georgiev, I., Seidel, H.-P., and Singh, G. (2022). Scalable multi-class sampling via filtered sliced optimal transport. *ACM Trans. Graph.*, 41(6).

Sawhney, R., Lin, D., Kettunen, M., Bitterli, B., Ramamoorthi, R., Wyman, C., and Pharr, M. (2022). Decorrelating restir samplers via mcmc mutations.

Shirley, P. and Chiu, K. (1997). A low distortion map between disk and square. *Journal of Graphics Tools*, 2(3):45–52.

Talbot, J., Cline, D., and Egbert, P. (2005). Importance Resampling for Global Illumination. In Bala, K. and Dutre, P., editors, *Eurographics Symposium on Rendering (2005)*. The Eurographics Association.

trends.google.com (2023). Google trends. https://trends.google.com/trends/explore?q=ray%20tracing&date=all.

Ulichney, R. (1987). *Digital Halftoning*. Cambridge, Ma.

Ulichney, R. A. (1993). Void-and-cluster method for dither array generation. In Allebach, J. P. and Rogowitz, B. E., editors, *Human Vision, Visual Processing, and Digital Display IV*, volume 1913 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 332–343.

Veach, E. (1998). *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA.

Veach, E. and Guibas, L. J. (1997). Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 65–76, USA. ACM Press/Addison-Wesley Publishing Co.

Walter, B., Marschner, S. R., Li, H., and Torrance, K. E. (2007). Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, page 195–206, Goslar, DEU. Eurographics Association.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Wolfe, A., Morrical, N., Akenine-Möller, T., and Ramamoorthi, R. (2021). Scalar spatiotemporal blue noise masks. *CoRR*, abs/2112.09629.

Yellott, J. (1983). Spectral consequences of photoreceptor sampling in the rhesus retina. *Science (New York, N.Y.)*, 221:382–5.

Appendices

A Code

The source code for this project is available publicly at: https://git.science.uu.nl/vig/mscprojects/blue-noise-distributed-mcmc-decorrelation-of-restir.

B Reflection

It was unfortunate that the leap from ReSTIR DI to ReSTIR PT was as big as it was, partly due to the algorithms being structured differently, so in hindsight it might have been better to focus on only one of the two.

Furthermore, the majority of the time was spent on implementing the decorrelation algorithm, leaving little to none for the handling of blue noise. This also resulted in time constraints concerning the evaluation and analysis of the results.