An Exploration of Improving Sampling within Monte Carlo Ray Tracing using Adaptive Blue Noise

Declan Russell[1]

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of Masters by Research

September 2018

[1]e-mail:DRussell@Bournemouth.ac.uk

Copyright Statement

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and due acknowledgement must always be made of the use of any material contained in, or delivered from, this thesis.

Abstract

In this report we demonstrate that strategically choosing sampling points with an intelligent use of adaptive blue noise sampling methods can drastically reduce the computation time required in the rendering process. We explore the state of the art in blue noise sample generation and explore new ways it can be used within the rendering process. Monte Carlo ray tracing is a vastly adopted image syntheses technique used ubiquitously across commercial and academic applications. It is capable of creating very high fidelity physically plausible images with computation that it unrestricted by dimensionality unlike most analytical approaches. Although capable of producing a high quality images Monte Carlo ray tracing often still needs to compute millions, if not billions of samples to produce a fully converged, noise free image. Tracing these samples comes at a cost and can lead to large computation times. Although we can reduce the cost of tracing rays with more optimized acceleration structures or naively throwing more hardware at the problem these are overshadowed by the improved quality gained via improved strategic sampling. Strategic sample placement has been proved to improve convergence rate of Monte Carlo ray tracing requiring fewer samples, and therefore decrease computation required to produce comparable results in quality. We explore the current literature on sampling methodologies and compare their implementation, performance and limitations and show that their sampling quality is inferior to adaptive blue noise. We will focus on applying the use of adaptive blue noise sampling within four dimensions of the rendering pipeline specifically. Firstly, we present a technique for generating primary ray samples that adaptively samples the image plane. We use a blue noise algorithm that adapts based on pre-existing information about the scene to increase the sampling frequency within areas of interest. Secondly we look at filter importance sampling, a technique that seems to be becoming ever more popular in rendering, and how we can use adaptive blue noise to generate higher quality sampling distributions than what is possible with the currently used methods of importance sampling. Next we will explore importance sampling BxDFs by generating samples over the hemisphere. Finally we will conclude with a brief discussion on some future ideas about direct light sampling of arbitrarily defined mesh lights. A challenge that is faced by all professional rendering software as efficient sampling methodologies are not well defined. Although not all our results in this report are entirely conclusive we believe that we have brought attention and provided promising results that build the foundations to an under researched area of knowledge that could help solve practical rendering problems faced by professional graphics.

Contents

Chapter 1

Introduction

"Random number generation is too important to be left to chance"
– Robert Coveyou

Ray tracing is a very popular, if not the most popular method to use in the field of image synthesis. This technique is the foundation of all production renderers, whether they are used for visual effects, architecture or even product design. In its most simplistic terms ray tracing is the act of simulating how light propagates through some domain of arbitrary dimension to produce and image. This is achieved by tracing rays into our domain that bounce around when interacting with objects within said domain. While traversing the ray path through the domain we integrate the accumulated radiance gained at each point of the path to produce a resultant pixel colour. This technique often requires millions, if not billions of rays to produce a high quality noise free result. The ray tracing algorithm is very computationally expensive and remains the most time consuming process of the image creation pipeline, often taking hours to produce a single image. When used in animation that typically requires a minimum of 24 frames per second this computation time can accumulate to months depending on how complex the scene is. For example the popular animated movie Monster's University created by Pixar took 100 million CPU hours to render the film to its final form. To address these huge render times, production houses require large server arrays known as render farms which cost millions to buy and operate simply to generate these images in a feasible amount of time. The central premise of this report is that strategically choosing where these rays sample with an intelligent use of adaptive blue noise sampling methods can drastically reduce the computation time required in the rendering process. Higher quality samples will reduce convergence time and therefore reduce the number of rays required to trace to create a fully converged image.

Brute force, unbiased Monte Carlo ray tracing is the most fundamental of ray tracing technique. Its unbiased nature can be relied upon to produce consistent correct results. First created in the 1950s by a group of researchers including John von Neumann and Stanislaw Ulam lead by Nicholas Metropolis the Monte Carlo method is a statistical approach to solve deterministic many-body problems. Monte Carlo integration, named by John Von Neumann in reference to a casino in Monaco called the Monte Carlo. Typically being a casino a lots of chance games are played at the Monte Carlo. Similarly Monte Carlo integration is the process of using "chance" to generate random samples of some arbitrary function and averaging the results to approximate value of the integral.

In 1979 Turner Whitted created the groundbreaking technique now famously known as "Ray Tracing" (Whitted, 1980). The technique of simulating rays that

propagate through some defined domain to accurately evaluate the global illumination and render a two dimensional image of a three dimensional scene. Monte Carlo integration is used ubiquitously across many signal reconstruction methods and it is one of the core pillars of ray tracing. It is used as the main method across the various dimensions of ray tracing to integrate the global illumination in various points in the scene, hence the name "Monte Carlo Ray Tracing". It has a very attractive property of being a integration method that is indifferent of dimensions, making Monte Carlo integration an exception to the curse of dimensionality which restricts other more analytical integration methods. It is this property that allows traditional ray tracing to be a practical method for generating images given that most if not all scenes that we want to render lie in multidimensional space. At the time of its creation Monte Carlo Ray Tracing was not deemed to be a practical rendering technique due to its large computational overhead that would easily take hours to render very simple scenes even at small resolutions. In fact at the time Turner Whitted estimated that it would require a Cray super computer assigned to each pixel to achieve real time ray tracing speeds. Technology has made significant advances since and now Monte Carlo Ray Tracing is the most popular technique for generating high quality physically plausible high fidelity images. We now see Ray Tracing being ever increasingly adopted in new disciplines such a visual effects, video games, architecture and product design to improve the the physically correctness and quality of images produced.

However, regardless of its vast adaption traditional Monte Carlo Ray Tracing is not a *Silver Bullet* and is very easy to create scenarios in which it will perform very poorly. Due to the unbiased nature of Monte Carlo ray tracing it can take a large number of samples to converge in scenes with occluded light sources. For example a room that is lit by a single light source in an awkward location (see figure 1.1a) or an occluded light source like the sun which is placed outside a window (see figure 1.1b).

(a) (b)

Figure 1.1: An interior scene rendered with the unbiased unidirectional path tracer Arnold. (a) Is lit with a single quad light facing upwards and is very close to the ceiling. (b) Is lit from a light source representing the sun from outside the windows. Images courtesy of Iliyan Georgiev

With the light source being heavily occluded means that the probability of hitting a light source by the rays fired from the camera, which are guided around the scene at random, is very low. It can take many bounces for the rays to reach the light source which in tern will add more computation time. Furthermore, the more bounces a ray has, the contribution to the final pixel colour becomes progressively less. In fact most Monte Carlo path tracers utilize Russian Roulette (see Pharr and Humphreys

(2010, pg 680)) which is an optimization to terminate ray paths that become too long when their contribution falls below a given threshold.

There has been a vast amount of research extending Monte Carlo path tracing to reduce the number of rays needed to trace and improve the convergence rate in such scenes. Once such popular technique is a generalization of traditional path tracing known as Bidirectional Path Tracing (see Veach (1997)). Bidirectional Path Tracing can achieve substantially lower variance with the same computation as traditional path tracing (see figure 1.2). The basic principle is as the name suggests and traces ray paths in two directions. One path forward from the camera and one backward from the light. We can then connect these paths with some path mutation. For example if we hit a diffuse surface, we can assume that this surface distributes rays evenly across the lobe and therefore we can simply pick a direction towards the last intersection of the light path.

(a) (b)

Figure 1.2: Two images rendered with the same number of samples. (a) Uses unidirectional path tracing and (b) uses bidirectional path tracing. Images courtesy of Iliyan Georgiev

However, although this technique works well for interior scenes it is easy to construct a scene where bidirectional path tracing will loose its benefits. One example would be an exterior scene where the camera and light source are obscured by difficult geometry creating challenges in connecting the paths and therefore diminishing the advantages of tracing rays from both directions.

There is a group of techniques that use caching in one form or another compute an estimate contribution of samples eliminating the expense of actually tracing rays for them. The simplest of these techniques is irradiance caching (Ward et al., 1988). The general idea is based upon the observation that unlike direct lighting that can change rapidly from one point to another, indirect lighting changes more gradually. Irradiance caching will store previously traced samples into some sparse data structure. From this sparse set of samples we can interpolate the illumination to compute a fairly accurate representation of the indirect illumination contributing to our path. Another, similar method is Photon Mapping (Jensen, 1996) which is part of the family of particle tracing techniques. In its simplest way you can think of Photon Mapping is that as a pre-computation, photons are scattered from light sources. We then record the energy and position deposited on the surface the packet lands on in some sparse data structure like a kd-tree. Finally, much like irradiance caching we can then use these recorded samples when tracing rays from the camera

to accelerate the illumination computations. This technique is particularly good at handling glossy and diffuse reflections as well as caustics. Building on Photon Mapping is a rather elegant solution named Vertex Connection and Merging (Georgiev et al., 2012). This technique combines both Bidirectional Path Tracing and Photon Mapping and has shown impressive results that prove more robust than bidirectional or Photon Mapping alone (see figure 1.3).

(a) (b)

Figure 1.3: Two images rendered with the same number of samples. (a) Uses bidirectional path tracing and (b) uses vertex connection and merging. Images courtesy of Iliyan Georgiev

Finally, we have Metropolis Light Transport (Veach and Guibas, 1997). This is an interesting techniques based upon Metropolis Sampling (Metropolis et al., 1953) which we will cover more in depth later in this report. Metropolis Light Transport will mutate ray paths overtime to such that when an "important" path is found, i.e. a path that contributes a large amount of illumination to the scene, successive paths will tend towards similar areas within the sampling domain. In practice however this technique has been deemed difficult to implement and often results in artifacts such as flickering in animations due to its unpredictably biased nature. Notice in figure 1.4 the incorrect illumination caused by paths of high illumination being over sampled.

Figure 1.4: Metropolis Light Transport rendered in the same time as figures 1.2 and 1.3. Notice the erroneous artifacts produced in the mirror due to correlations in sampling. Image courtesy of Iliyan Georgiev

These biased techniques however bring their own caveats which may prove undesirable. Introducing bias can introduce error and artifacts in the final result as you can see from figure 1.4. Error estimates are often not well defined for most bias rendering techniques results and although an image may look converged it may still have error. Unlike its unbiased counterpart, if a rendered image does have error or artifacts, increasing the sampling rate of biased techniques will not necessarily eliminate them.

From the discussion in this section we can identify that there is not yet a single *silver bullet* solution to how to correctly ray trace a scene. It is evident however, that the core difference between techniques is the way in which we sample a scene. Similarly to the quote stated by Robert Coveyou at the beginning of this chapter, the way in which we generate samples in ray tracing is too important to be left to chance and can cause drastic impact to the quality of the contribution that sample has on the results produced. In fact it has been proved that stratified sampling can improve the rate in which an image can be resolved with ray tracing. Though it is likely impossible to sample a scene with a 100 percent efficiency due to the infinite dimensionality and complexity the sampling domain, it does mean that the way to generate samples is still, and likely will forever be very much an unsolved problem. In this report we take inspiration from the current state of the art and investigate how to improve sample generation within various different dimensions of ray tracing. We will begin with an outline of the existing work in the field of sample generation, there implementations and limitations. Most notably, we will focus on blue noise sampling patterns which are in general considered a very attractive sample distribution. Blue noise, named due to its power spectrum resembling that of blue light signal, is a type of noise that has properties that are very attractive for generating samples for numerical integration. The blue noise profile is empty in the low frequency regions of its power spectrum which when used for sampling guarantees that samples are evenly distributed and avoids sample clustering. Furthermore blue noise remains stochastic which is a property that aids in removing aliasing in images. However blue noise is often neglected in practical applications in favour of Quasi-Monte Carlo sampling techniques due to the challenges in extemporaneously generating progressive blue noise samples efficiently. In this report, we suggest that there is still a place for blue noise sampling within practical applications. Specifically we will discuss four areas starting with primary ray sampling followed by filter importance sampling and finally some discussions around BxDF sampling and light sampling.

Chapter 2

Background and Related Work

2.1 Probability Theory

Random variable X is the result of some random process to generates numbers. Random numbers can either be discrete (e.g. the roll of a die) or continuous which take on real values $\mathbb{R}$. The *Cumulative Distribution Function* (CDF) is the probability of random value X being less than or equal to some selected variable x:

$$\mathsf{cdf}(x) = P\{X \leq x\} \tag{2.1}$$

The *Probability Density Function* (PDF) is the probability of choosing some variable x. The PDF is equivalent to the derivative of the CDF:

$$\mathsf{pdf}(x) = \frac{d}{dx}\mathsf{cdf}(x) \tag{2.2}$$

PDFs are always non-negative and integrate to 1 over their domains. The *expected value* $E[f(x)]$ of function $f(x)$ is the average value over the function with distribution $\mathsf{pdf}(x)$ over some domain D defined as:

$$E[f(x)] = \int_D f(x)\mathsf{pdf}(x)dx \tag{2.3}$$

It is this that is the basis of how Monte Carlo integration computes the expected values of arbitrary integrals.

2.2 The Monte Carlo Estimator

As mentioned previously the basic Monte Carlo estimator uses random sampling of a function to compute the estimate or expected value of an integral. To introduce the estimator we will use an example in the one-dimensional case. Suppose we wish to integrate function $f(x)$ over the domain $[a, b]$:

$$F = \int_a^b f(x)dx \tag{2.4}$$

Given a set of N uniform random variables $X_i \in [a, b]$ the Monte Carlo operator defines the expected value of the estimator:

$$F = \frac{b-a}{N}\sum_{i=1}^{N} f(X_i) \tag{2.5}$$

$E[F_n]$ is in fact equal to the integral. This can be easily shown in just a few steps. Notably the PDF to our random variables X_i must be equal to $1/(b-a)$, since the PDF must integrate to one over the domain $[a, b]$. Therefore with some manipulation we prove this statement in the following equation:

$$\begin{aligned} E[F] &= E[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)] \\ &= \frac{b-a}{N}\sum_{i=1}^{N} E[f(X_i)] \\ &= \frac{b-a}{N}\sum_{i=1}^{N} \int_a^b f(x)\mathsf{pdf}(x)dx \\ &= \frac{1}{N}\sum_{i=1}^{N} \int_a^b f(x)dx \\ &= \int_a^b f(x)dx \end{aligned} \tag{2.6}$$

We can relax the restriction to uniform random numbers with a small generalization. Having this generalization is very important as choosing an appropriate PDF to draw samples from to solve an integral has a large impact on reducing variance in Monte Carlo integration.

$$F = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{\mathsf{pdf}(X_i)} \tag{2.7}$$

Monte Carlo integration can very simply be extended to multiple dimensions. We simply select our random samples X_i from a multidimensional PDF and the same estimator is used. For example in a three dimensional integral using uniform variables where $X_i = (x_i, y_i, z_i)$ our estimator simply becomes.

$$F = \frac{(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)}{N}\sum_i f(X_i) \tag{2.8}$$

where our PDF is a constant value

$$\mathsf{pdf}(X) = \frac{1}{(x_1 - x_0)}\frac{1}{(y_1 - y_0)}\frac{1}{(z_1 - z_0)} \tag{2.9}$$

However due to the nature of Monte Carlo integration the error reduced per sample decreases with a rate of $O(\sqrt{N})$ where N is the number samples performed. This means that to reduce the error by half we must quadruple the number of samples. Therefore to solve the integral to a high level of precision large numbers of samples may have to be taken leading to large computation times. While other integration methods converge significantly faster in one dimension than Monte Carlo, these techniques become exponentially worse with increased dimensions where Monte Carlo is naturally indifferent to dimensions. Making it the appropriate choice for such problems.

2.3 Sample Generation

The process of generating samples is one that has been wildly researched. Given the emphasis on blue noise that is present in this report, in the following section we will categorise the literature in two categories. With blue noise sampling techniques existing in one classification and then all other techniques falling under the classification of low discrepancy sampling techniques.

2.3.1 Low Discrepancy Sampling

The first and most basic strategical sampling method theorized is that of Jittered sampling. This is the process of applying random permutations to uniform sample distributions. Latin Hypercube Sampling (LHS) was first described by McKay et al. (1979) and involves dividing our n-dimensional sample space into M axis-aligned hyperplanes. Samples are generated within a selection of these hyperplanes abiding by the rule that only one sample can be taken within its relative hyperplane. This creates samples which are evenly distributed when projected onto each M axis-aligned hyperplane. However in the 2 dimensional case samples can suffer from clustering. An abstraction of this method which holds higher efficiency is Orthogonal Array Sampling B. Owen (1992). Although simple to implement this again suffers from poor samples distribution in multiple dimensions. Results of these methods in multiple dimensions have been compared to random sampling and have been criticized as giving no real performance or precision improvements. Further developments in this decade take the form of N-Rooks Shirley (1991). This method involves placing a jittered sample in each cell of the diagonal of a grid size nxn where n is the number of samples, over our sample domain. The x and y coordinates are then randomly shuffled. However N-Rooks distribution generated is deemed poor and a small improvement over pure random sampling. This was improved in 1994 by Chiu with Multi-Jittered sampling Chiu et al. (1994). Chiu superimposes a grid of N by N subcells for $N = n^2$ onto the N-Rooks grid. A jittered sample is then placed in each cell where the N-Rooks condition must also be met in the subcells. Chiu's method begins by placing samples in an ordered canonical arrangement. From here each dimension is in turn shuffled. This method preserves the properties of N-Rooks and proves to be an improvement to jitter. However even this technique still suffers from a degree of sample clustering.

More recent developments in low discrepancy sampling include Hammersmith, (0-2) sequences Pharr and Humphreys (2010), random Halton sequence Okten et al. (2012) and the most popular to be used in production renderers, the Sobol sequence Bratley and Fox (1988). These sequences generate well-spaced quasi-random samples in large numbers of dimensions. This is very useful in production rendering as it is very common that multiple dimensional samples must be generated to take into account sampling such dimensions as screen space, time and lens space. Although these sequences fit well within the Latin Hypercube Pharr and Humphreys (2010) and are very efficient, they can often generate regularities in their distributions and in some dimensions can become very poorly stratified. Further reading on this subject can be found in Veach (1997); Jarosz (2008).

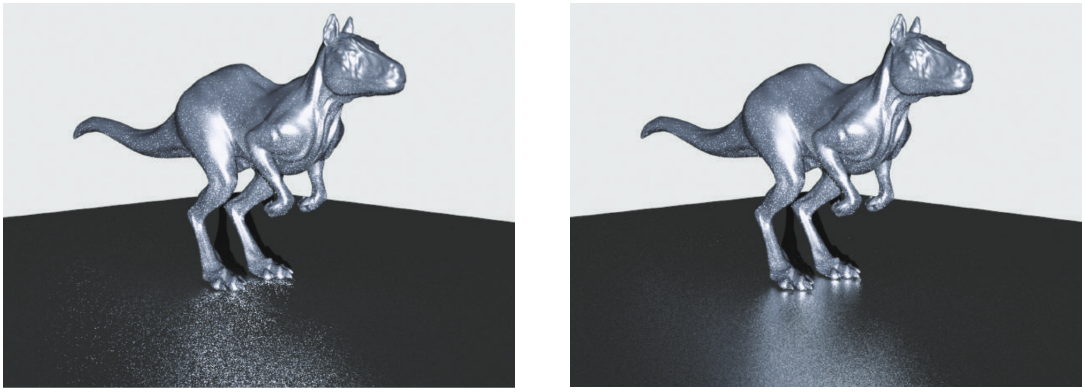
2.3.2 Blue Noise Sampling

There are a number of methods used for creating various profiles of blue noise with the most simple being the dart throwing algorithm (Cook, 1986; Mitchell, 1987).

Similar to rejection sampling, dart throwing is the process of randomly generating samples until the generated sample fulfils some defined minimum distance from all previously generated samples. This is a brute force technique with $O(N^2)$ complexity that becomes exponentially more costly as you increase size of the sample set generated. Various other techniques include Lloyd's relaxation (Lloyd, 1982), Poisson disk sampling (McCool and Fiume, 1992) and Capacity Constrained Voronoi Tessellations (CCVT) (Balzer et al., 2009). These techniques also tend to have a high level of computational cost and are also limited in the sense they can only produce a single blue noise profile. Later developments have achieved more efficient methods reducing computational complexity to $O(N \log N)$, see Cohen et al. (2003); Jones (2006); Du and Emelianenko (2006) and Dunbar and Humphreys (2006). Later Bridson (2007) and White et al. (2007) presented very efficient methods of complexity $O(N)$ with the use of acceleration structures to accelerate the look up of previously generated samples. Although efficient, these techniques to our knowledge are sequential and not parallelizable, therefore limiting performance. Further research has been made into parallelizable blue noise generation techniques to further improve sample generation speed such as Wei (2008). In 2015 Jiang et al. (2015) proposes an interesting technique that uses a smoothed particle hydrodynamics (SPH) simulation to creating blue noise. This method gives a level of flexibility by allowing the user to vary blue noise profiles by changing a single parameter. Furthermore this technique gives potential to high levels of performance being based upon SPH which can be easily parallelized to run efficiently even on large scales. In 2016 Reinert et al. (2016) provides a technique that produces high quality blue noise samples that retain their blue noise distribution when projected onto lower dimensional subspaces. Finally, during the later stages of creating this report, Per et al. (2018) proposes a technique for generating progressive sampling patterns from any arbitrary technique including blue noise using hole filling. Although this method is not efficient enough for online sample generation Per's results are very interesting and a great step forward in creating progressive blue noise sampling patterns.

2.4 Importance Sampling

Importance sampling is a very powerful variance reduction technique used to improve the efficiency of the Monte Carlo estimator. In fact, it is a crucial technique required to create a practical renderer. In order to integrate over our surface we must draw random samples over our domain to be used in our Monte Carlo estimator. The density in which we sample areas of our function can have a very large effect on the efficiency of the Monte Carlo estimator. The process of varying the density in which we sample a function is known as importance sampling. Importance sampling stems from the idea that if the evaluation of a sample at some given position of a function returns zero then this is considered a wasted sample in the Monte Carlo estimator because it provides no contribution to the accumulated result. In rendering such a situation would make results noisier which in turn will lead to more samples required to produce a resolved image. It has been proven that the best practice is to use a sampling distribution that is proportional to the integrand, see figure 2.1. Essentially, we want some function $f(\xi)$, where ξ is a random uniformly distributed number, that maps ξ to a distribution proportional to the function we are sampling. In the following sections will briefly discuss the currently used methods for achieving this mapping. For more in depth explanation of these techniques we refer to (Pharr and Humphreys, 2010, pg 643).



(a) (b)

Figure 2.1: 2.1a Using a standard uniform stratified sampling pattern that produces much more variance than 2.1b which applies importance sampling from a distribution based on the BRDF. Images taken from Pharr and Humphreys (2010)

Inversion Method The inversion method is made up of four steps. Firstly we are required to compute our *cumulative distribution function* (CDF). This can be obtained by integrating the *probability density function* (PDF) of the function we wish to sample. We will notate this $P(x) = \int_0^x p(x')dx'$ where $P(x)$ is our CDF and $p(x')$ is our PDF. Secondly, we compute the inverse of this CDF $P^{-1}(x)$ and generate uniformly distributed random number ξ. Finally, we compute our importance sample X_i where $X_i = P^{-1}(\xi)$.

This method works well and, although requires extra design thoughts when implementing importance sampling, it is a fairly simple method to use in most cases. However, the inversion method does assumes that it is possible to compute this analytic solution. Sometimes it may not be possible to integrate the PDF function in order to compute the CDF or perhaps it may not be possible to invert our CDF.

Rejection Method As mentioned in the previous section it may not be possible to create an analytic solution to generated samples that correspond with our integral. The *rejection method* can create samples only requiring the information provided by the value of our sampled function and the probability selecting that value. Given that we are drawing samples from function $f(x)$ with PDF $p(x)$ that satisfies $f(x) < cp(x)$ where c is some scalar constant. The rejection method goes as follows:

```
1 loop forever:
2   sample X from p's distribution
3   if ξ < f(X)/(cp(x)) then
4     return X
```

We repeatedly choose pairs of random variables (x, ξ) until the equation is satisfied. Therefore having a higher probability of acceptance within the higher regions of $f(x)$. This technique is essentially dart throwing which has the nice characteristic of working for any arbitrary function. However this techniques efficiency is dependent on how closely our distribution $cp(x)$ fits $f(x)$. Leaving functions with poorly approximated PDFs greater amounts of wasted computation when searching for samples.

Metropolis Sampling Metropolis sampling Metropolis et al. (1953) is a very powerful technique which can generate samples proportionally to a functions value. In our sampling domain Ω and our function to be sampled f, after selecting our first

sample $X_0 \in \Omega$ each successive sample X_i is generated using a random mutation to X_{i-1} to compute a proposed sample X'. Selecting a mutation strategy is a subject of its own so we will not go into this in the report but rather assume we already have one. See (Pharr and Humphreys, 2010, pg 694) for more on selecting mutation strategies. Given that we have our mutation strategy that proposes changing from given state X to X' we must be able to compute a tentative transition function $T(X \to X')$ that gives the probability density of the mutation transitioning to X'. With this transition function it is also possible to define an acceptance probability $a(X \to X')$ of accepting the proposed mutation.

$$a(X \to X') = min(1, \frac{f(X')T(X' \to X)}{f(X)T(X \to X')}) \tag{2.10}$$

If the transition probability is the same in both ways this simplifies to,

$$a(X \to X') = min(1, \frac{f(X')}{f(X)}) \tag{2.11}$$

Pharr and Humphreys (2010) presents a basic metropolis sampling algorithm to look like so,

```
X = X0
for i = 1 to n
  X' = mutate(X)
  a = accept(X, X')
  if (random() < a)
    X = X'
  record(X)
```

However, although this technique has proved quite successful at sampling proportional to a function, sequential samples are often correlated making it difficult to guarantee a even sample distribution across our integral. This introduced bias can lead to error and artifacts as seen in figure 1.4. Notice the correlations that are apparent in the mirror due the biased nature oversampling paths that lead to light sources. Metropolis sampling notoriously difficult to implement and one of the rarer methods used in practical image synthesis applications.

2.5 Generating Adaptive Blue Noise Samples with Smoothed Particle Hydrodynamics

The following chapters in this report will bare heavy use upon Jiang et al. (2015) technique generating blue noise samples using smoothed particle hydrodynamics. Therefore in this section I will give some brief theory behind smoothed particle hydrodynamics and Jiang's adaptation of the technique to generate adaptive blue noise samples.

Smoothed particle hydrodynamics (SPH) is a widely adopted method used in fluid simulation. SPH serves as an approximation to solve *Navier Stokes* equations 2.12 which provide a mathematical model of most fluids occurring in nature.

$$\rho \frac{du}{dt} = -\nabla p + \mu \nabla^2 u + f \tag{2.12}$$

The Navier Stokes equations looks at solving the acceleration of a fluid $\frac{du}{dt}$ by accumulating the pressure force $-\nabla p$, viscosity $\mu \nabla^2 u$ and other external forces f.

SPH was first used for fluids by Monaghan (1994) and uses a collection of approximations to solve each of these terms. In these terms a scalar A_i which represents an attribute of a particle, at location x_i is approximated with a set of neighbouring points j at location x_j using a symmetric smoothing kernel W_{ij}. Where h is the smoothing length of our kernel.

$$A(x) = \sum_j A_j \frac{m_j}{\rho_j} W(x - x_j, h) \tag{2.13}$$

$$\nabla A(x) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(x - x_j, h) \tag{2.14}$$

Where $\int W(x,h)dx = 1$ and $W(x,h) = 0$ when $||x|| > h$. Following these approximations allow us to solve for our unknown forces to solve for the acceleration of our fluid.

$$\rho_i = \sum_j m_j W_{ij} \tag{2.15}$$

$$-\nabla p = -m_j \sum_j m_j (\frac{p_i}{\rho_i} + \frac{p_j}{\rho_j}) \nabla W_{ij} \tag{2.16}$$

$$p = k(\rho - \rho_0) \tag{2.17}$$

Where p is our pressure calculated from our density ρ along with a user defined rest density ρ_0 and gas constant k.

Using the approach presented in Jiang et al. (2015) it is possible to generate various blue noise profiles which each have their own advantages and disadvantages when it comes to noise and aliasing. Jiang conveniently designs her algorithm such that the blue noise profiles generated can be controlled simply by varying a single input parameter. This parameter, known as the *density difference* is the difference between the rest density ρ_0 and the actual density $\overline{\rho}$ which is calculated as the mean density of all particles. To control the *density difference* which for our use we will name $\rho_\triangle$, at every step in the simulation we must calculate $\overline{\rho}$ and set ρ_0 such that it is lower than $\overline{\rho}$ by the user defined $\rho_\triangle$ simply expressed by equation 2.18.

$$\rho_0 = \overline{\rho} - \rho_\triangle \tag{2.18}$$

The simulation is considered converged when particle displacement $||x' - x||$ is less than ϵ where x' is the updated point position. ϵ is set as $\epsilon = 0.01d$ where d is the average distance between adjacent particles.

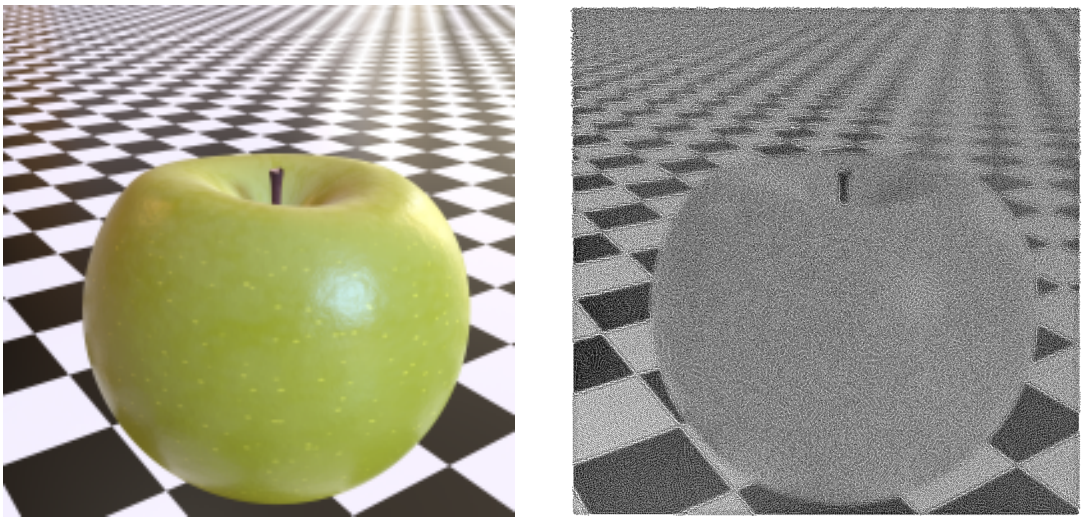
This method also supports adaptive sampling as seen in figure 2.2b. This is achieved by applying a distance field scale $s(x)$ to sample properties. This scalar field could be based upon any arbitrary property. An example could be in the use of image stippling one would want high densities of particles in darker regions of an image. Therefore you could use the scalar $s(x) = 1/\sqrt{I(x)}$ where $I(x)$ is the image intensity at location x. We use this to warp the distance between samples when computing our SPH calculations. The distance between 2 particles now becomes:

$$\widetilde{s}(x_i, x_k) = \frac{2(x_i - x_j)}{s(x_i) + s(x_j)} \tag{2.19}$$

This is then used in our smoothing kernel described in equations 2.15 and 2.16 as demonstrated in equation 2.20.

$$W_{ij} = W(\widetilde{s}(x_i, x_j), h) \tag{2.20}$$

Therefore as we increase our scale $s(x)$ our smoothing kernels will cover larger distances therefore causing an increase in pressure around this particle pushing neighbouring particles away resulting in particle placement becoming more sparse.



(a) Original apple image (b) Stippled apple image

Figure 2.2: Adaptive blue noise samples used to recreate 2.2a through image stippling 2.2b with our scaler s in our size function is as follows $s(x) = 1/\sqrt{I(x)}$ where $I(x)$ is the intensity of our image at location x.

2.6 Implementing the Adaptive Blue Noise Generator

Our adaptive blue noise generator is an implementation of the SPH based technique by Jiang et al. (2015). We have chosen this technique due to its speed and ability to easily adjust the blue noise profile generated. We use a weakly compressible SPH solver is roughly based on the work presented by Priscott (2010). Weakly compressible fluids were sufficient for our implementation as we do not require any additional forces such as gravity acting on the particles to generate our blue noise. We parallelize the solver using NVidia's parallel computing API CUDA using similar approaches to the methods described by Harris (2008) and Hoetzlein (2014). The solver in its simplest sense is outlined with the pseudocode snippet 2.1.

```
generate random particles
while not converged do
  for all i do
    hash particles to grid
  for all i do
    find neighbourhoods N_i(t)
  for all i do
    compute density rho_i(t) (Eq. 2.15)
  compute rest density rho_0 (Eq. 2.17)
  for all i do
    compute pressure p_i(t)  (Eq. 2.16)
  for all i do
    compute forces F
  for all i do
    compute new velocity v_i(t+1)
```

```
    compute new position x_i(t+1)
    map particles to domain
```

Listing 2.1: Adaptive blue noise solver pseudocode

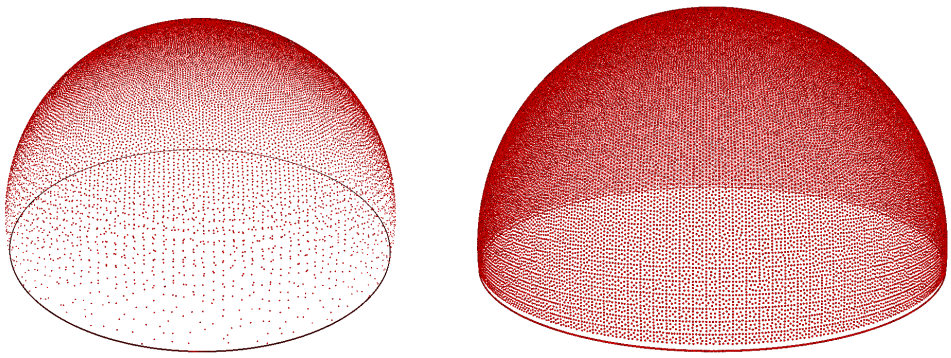
We initialize the samples randomly within the given sampling domain. The random number generation for these samples can be as simple as white noise. However, though not always possible, we have found improved convergence speeds if the samples can be generated evenly spaced or fairly proportional to the function the adaptive heuristic is based on as samples are initialized closer to the final result. The remaining functions are all implemented as CUDA kernels that run in parallel on the GPU on all samples. Following Harris (2008) we accelerate the neighbourhood search using a hash grid and use some intelligent sorting to simplify access patterns. Our hash function simply assigns the hash cell index to the sample with equation 2.21. This works well enough for our need but more elegant spatial hashes have been explored and can further improve performance with more regular memory access patterns that can improve caching.

$$hash(x, y, z) = x + y * gridDim.x + z * gridDim.x * gridDim.y \tag{2.21}$$

The density and pressure of the particles are computed with equations 2.15 and 2.16 respectively using the adaptive soothing kernel in equation 2.20. The rest density is computed at every time step with 2.18 and is used to control the blue noise profile that is generated. To compute the new positions we simply use Newton's second law of motion $a = \frac{F}{m}$ and integrate to solve for the new velocity and position. In our implementation we use a fixed time step with Leapfrog integration which although is only second order accurate is very stable. Though this worked well for our results in some scenarios when forces in the simulation become particularly large this can still produce instabilities. This could be improved with other methods such as adaptive time stepping and or more robust integration methods. Finally if needed we map particles back to their sampling domain. For example this could mean mapping samples back to the surface of the hemisphere such as in section 5.

However we found in our tests that depending on the scaling function used discontinuities can occur in the simulation. These would happen when the cell size in the has table was not sufficient enough to cover the neighbourhood of particles required after the scaling function is applied. When using an acceleration structure you partitioned space into a subset of equally divided space. When using an adaptive distance scale per particle this causes bandwidths of our smoothing kernels to vary relative to every particle making it difficult to select an appropriate cell size for the hash table without compromising performance. If a kernel bandwidth is larger than our hash subdivisions this can lead to under sampling neighbour particles which leaves areas of low densities resulting in particle clustering. On the other hand if our kernel bandwidth is to too small this can cause areas of high pressure. Furthermore boundaries are a difficult problem in SPH simulation as they can lead to density discontinuities causing particles to cluster at the boundaries. Examples of these issues can be seen in figure 2.3.

To help avoid these discontinuities we propose some possible solutions, however currently these solutions are by no means entirely reliable and require some experimentation to achieved the desired effect. In some difficult scenarios that do not require fast sample generation and can be precomputed we simply brute force the simulation without the acceleration structure.



(a) (b)

Figure 2.3: Distance scaling functions that cause discontinuities when used with an acceleration structure. In (a) the scaling function causes low densities, causing particles to pool at the bottom. In (b) the scaling function causes high pressure regions, causing grid-like patterns.

In section 4.1 we use the ghost particles method as presented by Schechter and Bridson (2012) which builds boundaries of a given simulation out of particles themselves. These samples behave the same way as other samples in the simulation with the one exception of having a fixed position. This solution was sufficient for our results but can prove difficult to implement correctly when adaptive smoothing functions are used. Particles in the boundary must be placed proportionally to the adaptive heuristic along the given boundary to avoid discontinuities between the boundary and the particles within the domain.

When generating samples for filter importance sampling in section 4.2 we generate samples on a two dimensional unit square which has four distinct boundaries. These boundaries can cause discontinuities in density therefore we remove the boundaries by turning the unit cube into a continuous surface. This is done by simply mirroring the samples from each side of the unit square to the opposite side of the square. In practice this works well because generally filtering kernels are symmetrical and tend to zero at their tails.

In section 5 in which we generate samples over a hemisphere we avoid areas of low density around the boundary by simply removing the boundary. For example if we are running our simulation over a hemisphere we simply run it over a sphere instead such that our surface becomes continuous, see figure 2.4. We then simply use particles generated on one of the hemispheres of our sphere as our sample positions. In fact, we can reuse the samples on the second side of the hemisphere as more sample batches to avoid wasting the computation used.

Secondly we currently limit our scaling function between minimum and maximum boundaries. This helps avoid creating kernel bandwidth that are too small thus causing particles to cluster at the boundaries of our hash. For example,

$$\begin{aligned} s(x_i) &= a(I(x_i)) + c_{min} \\ a &= c_{max} - c_{min} \end{aligned} \tag{2.22}$$

Where c_{min} and c_{max} are the minimum and maximum values respectively that we wish our scaling function to be limited to.

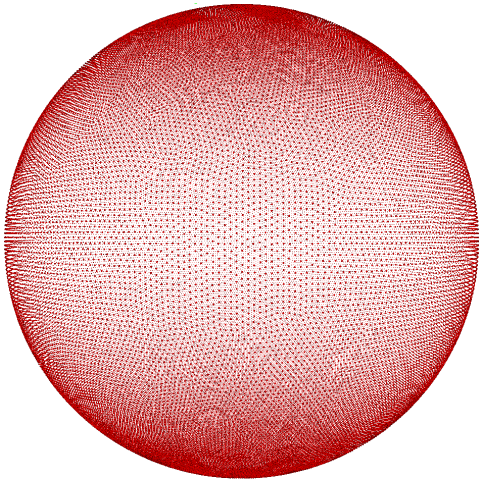


Figure 2.4: Diffuse samples generated over a sphere

2.7 Related Work

The importance of sampling within the context of rendering was first pioneered by the works of Dippé and Wold (1985) and Cook (1986). Dippé and Wold (1985) identified that the problem with regular grid shaped sampling patterns is that as we are point sampling there are large areas that are never sampled. The regular structure of these patterns causes coherent regular errors in the form of aliasing. On the other hand irregular sampling pattens where we randomly vary the space between samples, have a random chance of sampling any region creating unstructured, incoherent *noisy* error. These irregular patterns prove more visibly pleasing as it mimics the random receptor patterns within the human retina. These early works only explored the use of two sampling patterns, Jitter and Poisson Disk. Shirley (1991) was the first to use discrepancy as a measure of quality to evaluate sampling patterns. In its simplest sense, measuring the discrepancy is done by splitting the sampling domain into n regions and comparing the number of sample points to the volume of that region. In general, to achieve a good quality discrepancy a fraction of the volume should have roughly the same fraction of the total number of sample points inside it. However discrepancy alone is not necessary a good metric. Sampling patterns can still achieve low discrepancy while still exhibiting sampling clustering in their distributions. Mitchell (1991) further pursues the idea of improving sampling in n-dimensional space with stochastic sampling with the use of N-Rooks sampling. Mitchell (1996) has provided interesting analysis on the consequences of stratified sampling within ray tracing. In his studies he concludes that for a single sample stratified sampling is no better than uniform random sampling. However as the number of samples per pixel increases the stratified sampling will converge to the mean asymptotically faster. He also observes that the improvement a sampling distribution may have is also linked to the nature of the image under the area of the pixel. Though in the worst case stratified sampling will be no worse than uniform random sampling. They state that if a finite number of edges pass through the area of a pixel that the variance of the mean to be lower by a factor of $N^{\frac{1}{2}}$ with strategic sampling and a factor of N if the image is smooth under the pixel area. Heck et al. (2013) focus on the spectral properties of sampling patterns and observe how the different oscillations in their power spectrum have an affect on the aliasing artifacts in resulting images. Heck et al. (2013) characterizes the perfect sampling distribution will ideally have a high Nyquist frequency, i.e. the size of the

low-frequency region in its Fourier power spectrum and low oscillations. However, they unfortunately discover that with a high Nyquist frequency also comes high oscillations. This is the case because as we maximize the space between samples this will lead to forming more regular structured patterns. Thus leaving the design of a sampling distribution to have a trade off between these properties, or in other words a trade off between noise and aliasing.

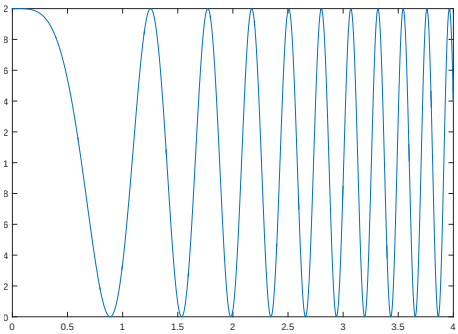
Chapter 3

A Comparison of Sampling Techniques

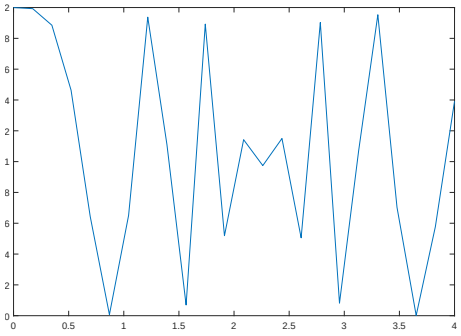
In section 2.2 we outlined the Monte Carlo estimator which we use in rendering to integrate our image function. As mentioned in this section the Monte Carlo estimator uses random samples of the integrand function to estimate the result of the integral. It turns out that the distribution of these samples can make a great difference in the performance of the estimator. In this section we will give some insight to how and why using strategic sampling strategies can have an important role in improving error that lies in our estimator.

Aliasing is one of the biggest practical problems in signal reconstruction. A function not sampled at a significantly high sampling rate can lead to a loss of data and the reconstruction of a signal not accurately representing the original function as seen in figure 3.1.

To avoid loss of data, sampling theorem tells us that to avoid such aliasing we must sample at least at a rate of twice the maximum frequency apparent within the signal. This sampling frequency is known as the *Nyquist frequency*. However although it may seem obvious to sample every signal at the *Nyquist frequency*, it is not always that easy. The majority of signals in rendering may not have a band limit, and therefore it is impossible to sample at a high enough rate to avoid loss of data.



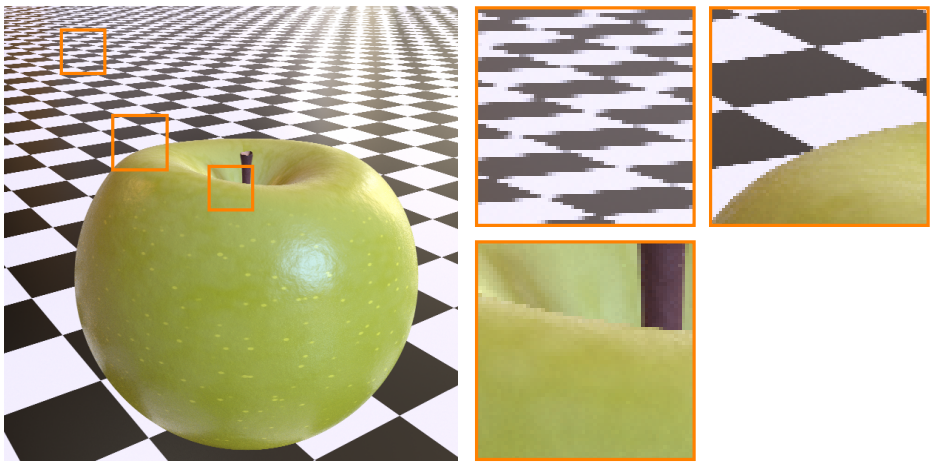
(a) The function $y = 1 + cos(4x^2)$



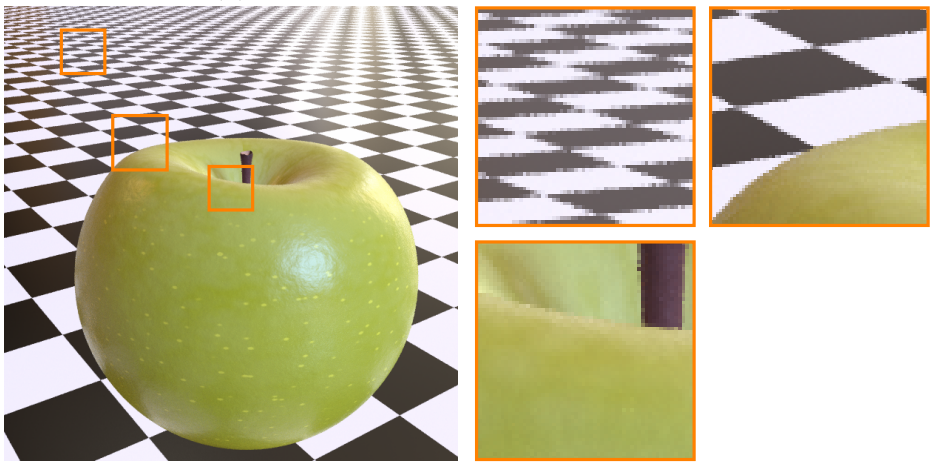
(b) The function $y = 1 + cos(4x^2)$ sampled at the interval 0.25

Figure 3.1: Example of aliasing caused by high frequency information lost via an inappropriate sampling.

Although we may not always be able to sample at the required frequency to avoid



(a)

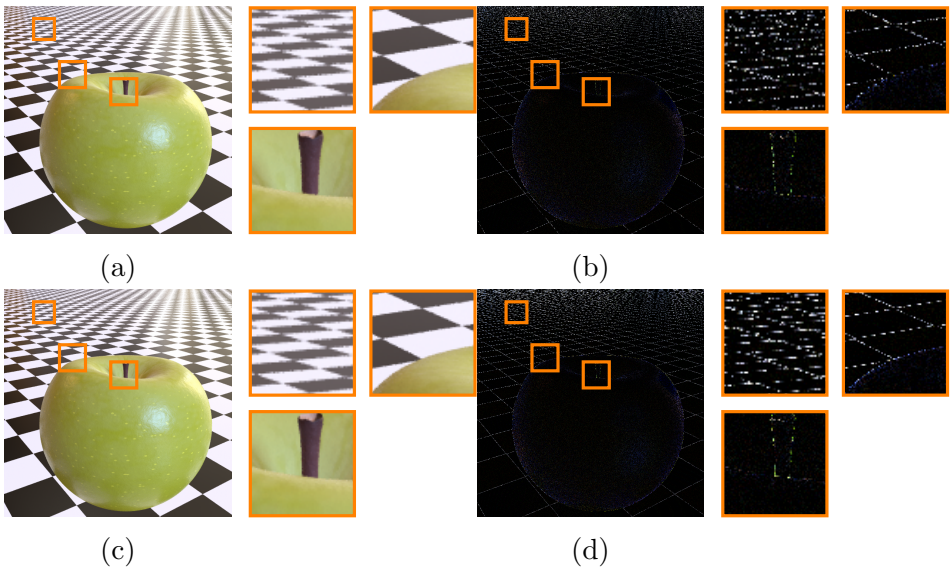


(b)

Figure 3.2: 3.2a Downscaling of an image using 16 uniform samples per pixel and 3.2b using 16 non-uniform samples per pixel

aliasing there are a number of ways we can approach sampling signals to reduce the visual impact of the aliasing. These are suitably known as *anti-aliasing* techniques. Firstly, non-uniform sampling is where we vary the space between samples. Although still incapable of obtaining correct results in unlimited band frequencies it tends to mask aliasing artifacts as noise which is much more visually pleasing. This is demonstrated in figure 3.2. Notice the sharp stair like edges of the chequered squares and the edges of the apple using the uniform sampling in contrast to the non-uniform sampling where these edges are blurred.

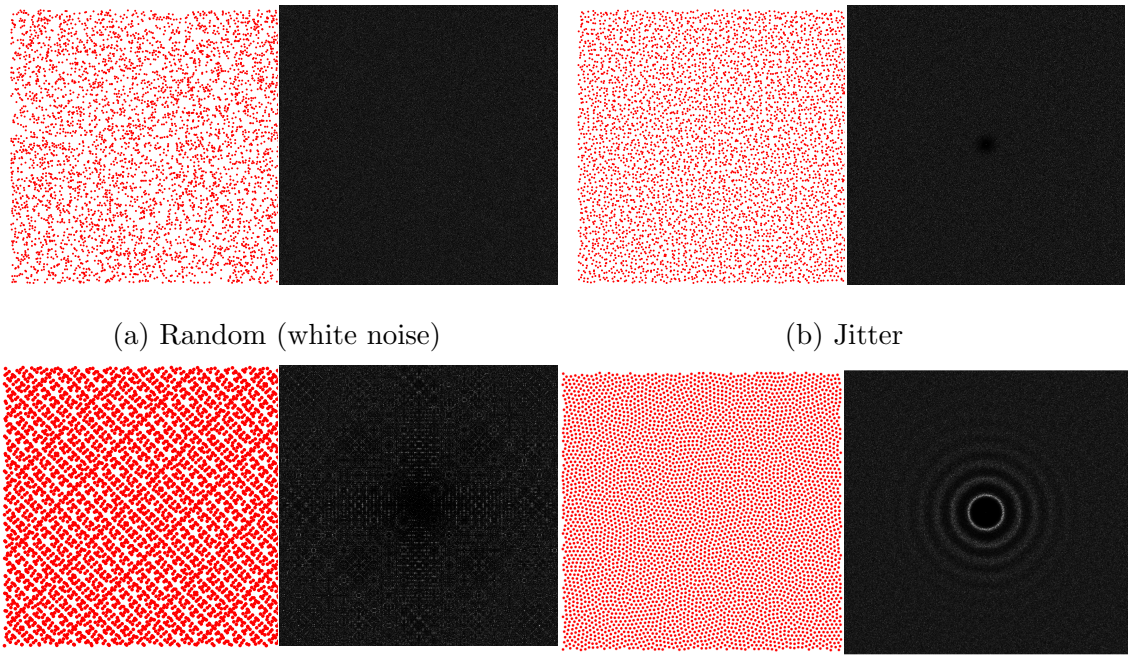
Although using random non-uniform samples proves more visually pleasing than using uniform samples the quality of these distribution can still make a big difference. In the simplest case of selecting our samples at random we can often see clustering in our distribution, see figure 3.4a. This is an unattractive property as it can cause sparse unsampled areas in our distribution leading to the loss of important information about our integrand. Ideally we want to evenly evaluate samples over the sampling domain as much as possible while still maintaining a level of stochasticity. Strategic sampling methods look to avoid this with the goal of generating stochastic and yet evenly distributed samples. These include low discrepancy techniques such as jitter (figure 3.4b), Quasi-Monti Carlo sequences such as Sobol (figure 3.4c) and blue noise (figure 3.4d). We give an example of how strategic sampling can improve sampling quality in figure 3.3 which shows a clear reduction in error.



(a) (b)

(c) (d)

Figure 3.3: 3.3a Downscaling of an image using 16 random samples per pixel and 3.3c using 16 strategic blue noise samples per pixel. Figures 3.3b and 3.3c show the error of each pixel multiplied by 10. The reduction in error is roughly 17.7%



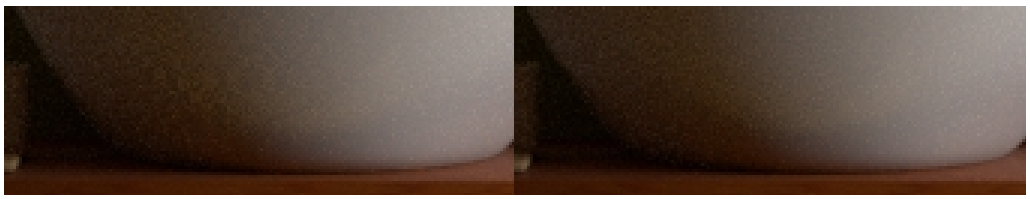
(a) Random (white noise) (b) Jitter

(c) Sobol sequence (d) Blue noise

Figure 3.4: Various anti-aliasing sampling distributions and their Fourier transform power spectrum generated with Wei and Wang (2011)

The distributions in figures 3.4c and 3.4d of the Sobol sequence and a blue noise distribution respectively are of particular interest in this report. Sobol sequence, a member of the Quasi-Monte Carlo sequences family is a very popular choice of sampling distribution in practical applications. Quasi-Monte sequences such as the Sobol sequence have some very attractive properties. They are easy to implement and in the most case produce good quality results. Most importantly however, they can extemporaneously generate progressive samples with a very high number of dimensions. Meaning that the number of samples does not need to be predetermined

as you can generate new samples on the fly. Blue noise sampling, named due to its power spectrum resembling that of the signal generated by blue light, is a technique that is popular across several areas of graphics such as image synthesis and geometry processing. Blue noise is a close approximation to Poisson disk sampling which guarantee that no two samples are generated closer than a specified distance. This is exceptionally desired trait for samples to have in image synthesis techniques as studies have shown that the rods and cones within the human eye are distributed in a similar way. This sampling distribution produces stochastic samples that are evenly spread in the special domain, giving us an even distribution over our domain while also reducing aliasing issues masking them as noise which as mentioned previously appears more visually pleasing. However, current methods for generating blue noise samples are typically limited by the curse of dimensionality and typically not well adapted to generating samples on the fly. Where blue noise particularly shines is in the quality of the distribution. Comparing blue noise to Quasi-Monte Carlo techniques such as Sobol as see in figure 3.5 you can see that unlike blue noise the Sobol sequence can suffer from structured artifacts due to its rigid distribution. In fact some dimensions of the Sobol sequence are very poorly stratified which you can see in figure 3.6.



(a) (b)

Figure 3.5: Comparison of two images rendered with the same number of samples. 3.5a uses the sobol sequence 3.5b uses blue noise samples.

(a) (b)

Figure 3.6: Examples of some of the poorly stratified Sobol sequence dimensions. Notice the regular patterns and sample clustering. 3.6a Sobol dimensions 11 and 12 3.6b Sobol dimensions 19 and 20

Although these methods prove as a good foundation for sampling, in the general case we still need to generate enough samples to match the Nyquist frequency. This

may lead to wastefully over sampling lower frequency areas of our integrand that require less samples. Ideally we would want to vary our sampling rate such that we adaptively increase the number of samples in high frequencies and decrease in low frequencies. This is known as *Importance Sampling*. If we can identify the regions in which frequencies exceed the *Nyquist* limit then we can increase the sampling density of these areas without the expense of having to increase the sampling rate across the entirety of our domain. For more information on sampling theory and Monte Carlo integration see Pharr and Humphreys (2010); Jarosz (2008); Veach (1997).

Chapter 4

Applications of Adaptive Blue Noise within Image Plane Sampling

In this chapter we explore practical uses for adaptive blue noise sampling in the image plane in Monte Carlo ray tracing. Firstly we will look at generating adaptively placed primary ray samples across the image plane. These are the first rays that originate from the camera and are projected into the scene. Secondly we will look at how blue noise sampling can be used to improve the anti-aliasing technique known as filter importance sampling.

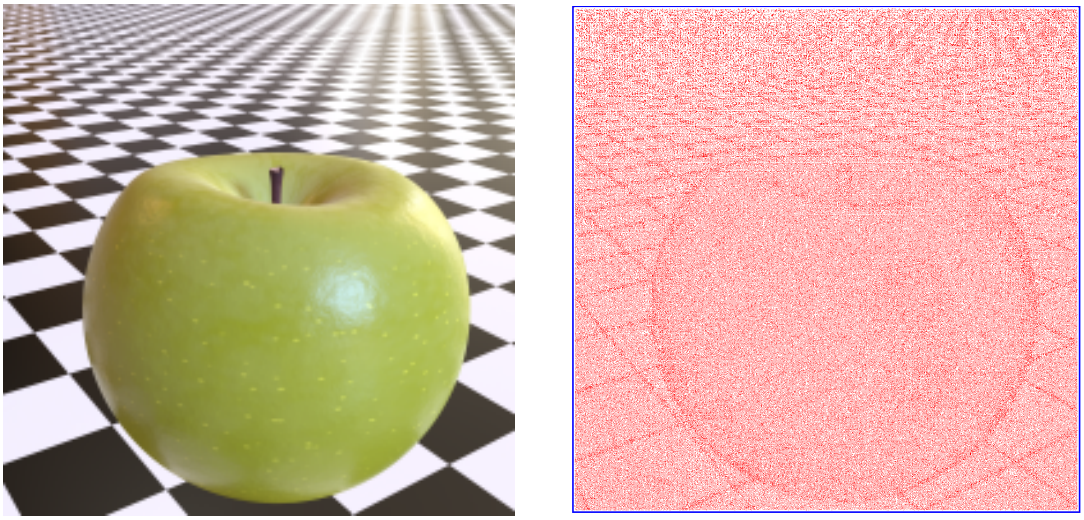
4.1 Adaptive Primary Ray Sampling

Primary ray samples can be thought of as the first dimension of sampling in our Monte Carlo ray tracer. This is the generation of rays from our camera into our scene. We are essentially trying to integrate the radiance that enters our camera from our arbitrarily defined scene into a discrete grid of colours, i.e. the pixels which make up our image. These pixels can otherwise be thought of as point samples of our image function which in turn can be thought of as some arbitrarily defined N-dimensional integral of radiance. We can generate primary ray samples by distributing points across a 2-dimensional plane. This plane varies in size depending on properties of the camera such as field of view, the resolution of the image plane and the shape of the camera. We then use the distribution of samples across the image plane as the ray directions from our camera to sample the scene. Finally we integrate these samples with the Monte Carlo estimator (section 2.2) to solve how much radiance enters the camera at each pixel.

As mentioned previously in section 3, generating an appropriate sampling distribution can significantly improve the performance of Monte Carlo integration. For optimal sampling efficiency we need to sample our function proportionally to the frequency of our function. Adaptive sampling is popular method to do this in most production renderers. This is the process of tracing more samples in pixels with a higher variance. Generally this is achieved by having some user set variance threshold and continuously sampling pixels until they reach this threshold. However, this is suboptimal as your adaptive sampling is bound by discrete resolution of the image. For example, you could very easily over sample areas of low frequency under a pixel simply if there is a small area of high frequency within that pixel.

In our implementation we use the technique proposed in Jiang et al. (2015) to

create our blue noise sampling distribution due to its speed and flexibility, however other techniques comparable of generating adaptive blue noise could also be used. We simulate the total number of samples that we will need for our integration at the same time across a 2-dimensional plane. We have found that to solve boundary discontinuities, using ghost particles in our boundary worked better than the originally proposed method. This involves creating a boundary out of particles around the borders of our simulation. These particles are taken into account in our SPH simulation, causing high pressure when particles move to close and therefore push particles away. This border is displayed in blue in figure 4.1.



(a) Original apple image

(b) Samples generated

Figure 4.1: Adaptive blue noise samples generated and adapted based on the variance measured when producing image 4.1a

After convergence has been achieved we randomly select a single sample, if available, for every pixel in our image plane for use in our Monte Carlo integration. Every sample selected for use in integration is then set to be frozen in our simulation. This avoids successive samples being able to be generated in the same place as previous samples taken. After samples have been traced we can use information retrieved in each pass to adapt our successive samples to better fit our image function. Currently we have investigated using the variance measured at every pixel after each pass in integration. We use this variance to adapt successive samples generated within our scene by varying our scalar distance field relative to the variance of each pixel using the equation:

$$s(x) = \frac{1}{aV(x) + c_{min}} \qquad (4.1)$$
$$a = c_{max} - c_{min}$$

Where c_{min} and c_{max} are the minimum and maximum values respectively that we wish our scaling function to be limited to and $V(x)$ is the variance at pixel location x. This means that in areas where the variance increases, so does the density of samples in these areas. We have found that these areas generally include high frequency regions such as hard edges of objects and sharp changes in textures that could appear in a scene as you can see in figure 4.1, an example of samples generated using this method. Currently we only look into using variance to adapt our particles

however this could be adapted to other properties of our scene as discussed in section 6.1 regarding future work.

4.1.1 Results

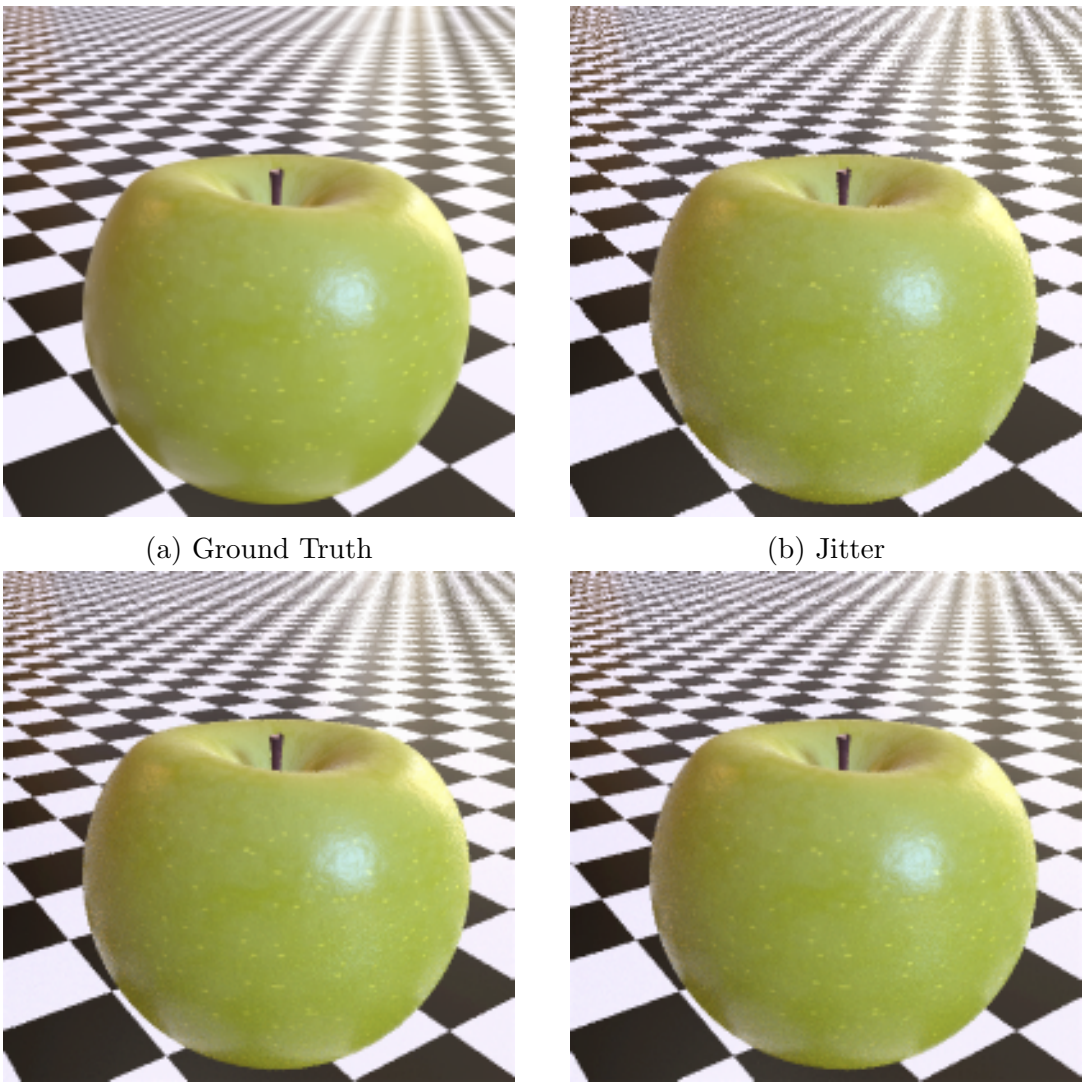
For our current experiments with our primary ray sampling generation, we use Monte Carlo integration and point sampling to scale a large resolution image to a smaller resolution image. This is essentially mimicking what we would be doing in traditional path tracing of evaluating some image function with point samples however for early experiments this is easier to implement and removes the computation of actually having to trace rays around a scene. For the results in this report we scale a 3000x3000 pixel image to a 200x200 pixel image. With our current implementation we generate 240,000 samples in approximately 10 seconds on a NVidia 970m graphics card and a Intel Core i7-4710MQ CPU @ 2.50GHz. This number of samples can be thought of as an average of 6 samples per pixel for a 200x200 pixel image.

It is evident in the results you can see in figure 4.2, blue noise is far superior to simple jitter, however there are finer differences between blue noise and our variance adapted blue noise. Our adaptive method appears to produce better results around the edges of objects. This is apparent if you focus around the edges of the apples stork displayed in figure 4.3. The blue noise sampled image in 4.3a suffers noticeably more by aliasing than our adaptive blue noise in figure 4.3c which holds truer to the ground truth in figure 4.3b. We can prove this by looking at the at the difference between the ground truth and these images which is shown in figure 4.4. It is clear that the differences around the edges of the stork are greater and more frequent just using blue noise (figure 4.4a) than the adaptive blue noise (figure 4.4b).

However due the large areas of high variance caused by the chequerboard running off into the distance in the top of the image we suffer from some sub-sampling and aliasing issues. This can be seen in figures 4.5 and 4.6. This is caused by our method identifying a very large area of high variance and adapting our particles such that they can converge closer together. Unfortunately at this point there is a lack of samples to create an even distribution around this area leading to clustering an a distribution similar to that of white noise. In fact this leads to a bias in our sampling similar to how it might in metropolis ray tracing and overall results in our adaptive blue noise sampling having 4.1% more error than the original blue noise sampling on the apple image.

4.1.2 Conclusion

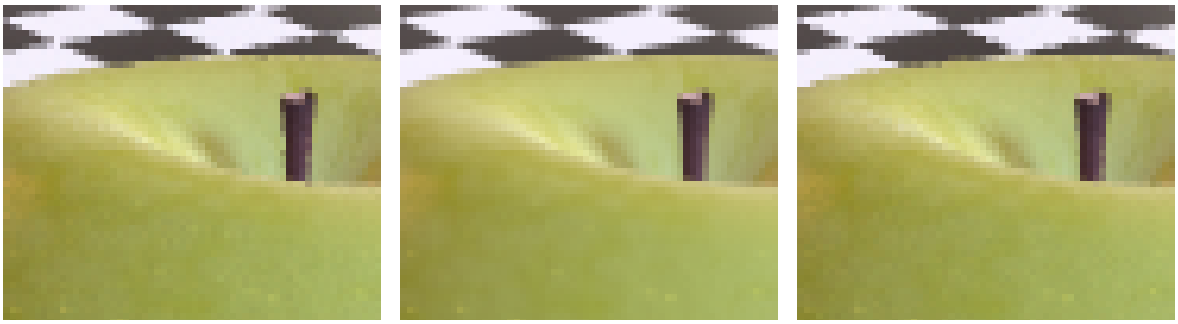
In this section we have demonstrated that in some cases our method of using adaptive blue noise for adaptive screen space sampling can improve results and reduce aliasing. These cases include hard edges of geometry as demonstrated in figures 4.3 and 4.4. However our method can also suffers in regions of high frequency due to correlations in sampling as demonstrated in figures 4.5 and 4.6. We believe that if these correlations can be avoided by perhaps smoothing such discontinuous areas of the sampled function that our technique can be a viable method for improving anti-aliasing in screen space sampling.



(a) Ground Truth (b) Jitter

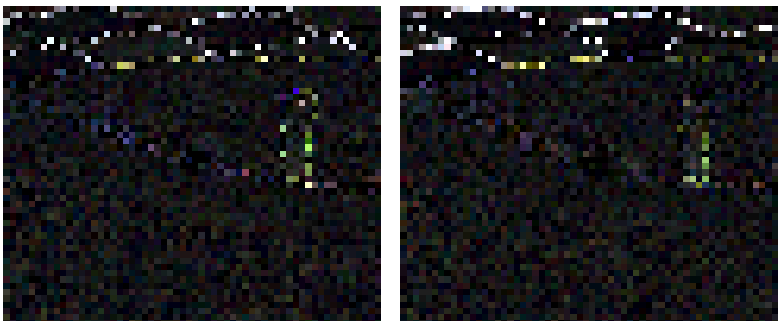
(c) Blue Noise (d) Adaptive Blue Noise

Figure 4.2: Comparison of different sampling techniques when scaling a 3000x3000 pixel image to 200x200 using 240,000 samples. The ground truth in image 4.2a was generated using 1000 random samples per pixel or 40,000,000 samples in total.



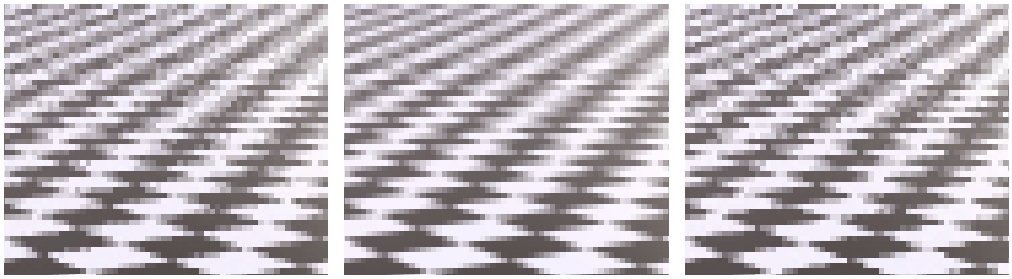
(a) Blue Noise (b) Ground Truth (c) Adaptive Blue Noise

Figure 4.3: Close up on the stork of the apple. Notice the aliasing on the edge of the stork in 4.3a compared to 4.3c.



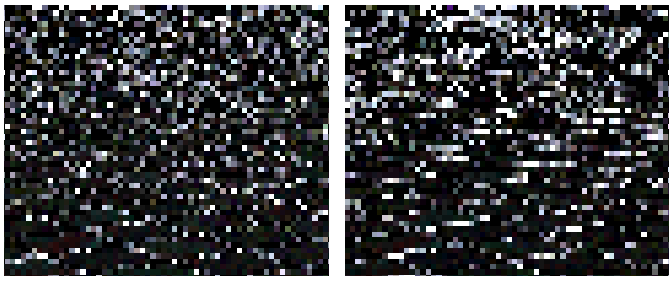
(a) Blue Noise (b) Adaptive Blue Noise

Figure 4.4: The difference multiplied by 10 between the ground truth and the close ups of the apples stork in figure 4.3. Brighter pixels mean higher difference and thus more error.



(a) Blue Noise (b) Ground Truth (c) Adaptive Blue Noise

Figure 4.5: Close up on the distant chequerboards in the apple image. Notice the increase in noise in the adaptive blue noise 4.5c im comparison to 4.5a.



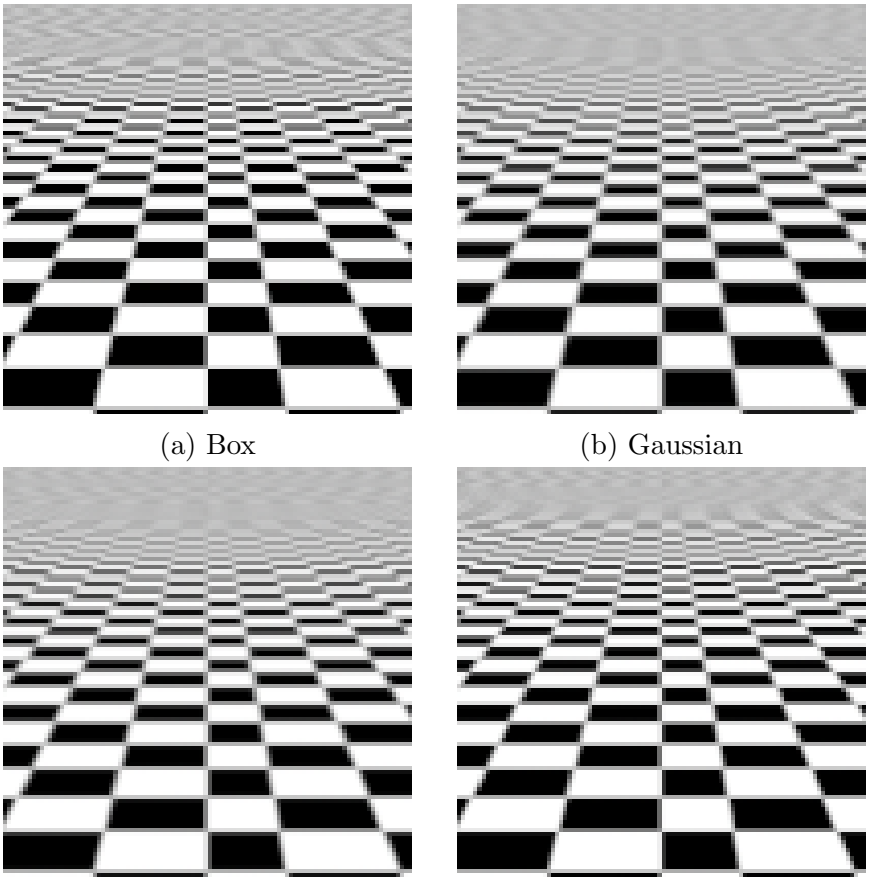
(a) Blue Noise (b) Adaptive Blue Noise

Figure 4.6: The difference multiplied by 10 between the ground truth and the close ups of the distant chequerboards in the apple image in figure 4.5. Brighter pixels mean higher difference and thus more error.

4.2 Filter Importance Sampling

Filter Importance Sampling (FIS)(Ernst et al., 2006) is another approach to achieve sample filtering. To understand FIS I will first give some background on what filtering is. Filtering is the process of accumulating camera samples to produce the final pixel color. The method used to accumulate these samples can aid in reducing aliasing between pixels in the final image. The simplest approach is known as box filtering. This means that all samples within the sampling domain, which for box filtering is our pixel, are weighted evenly and summed together. This is essentially taking the average value of all samples. However, this most basic technique can be very susceptible to aliasing artefacts. There are two elements to a filter, the weight-

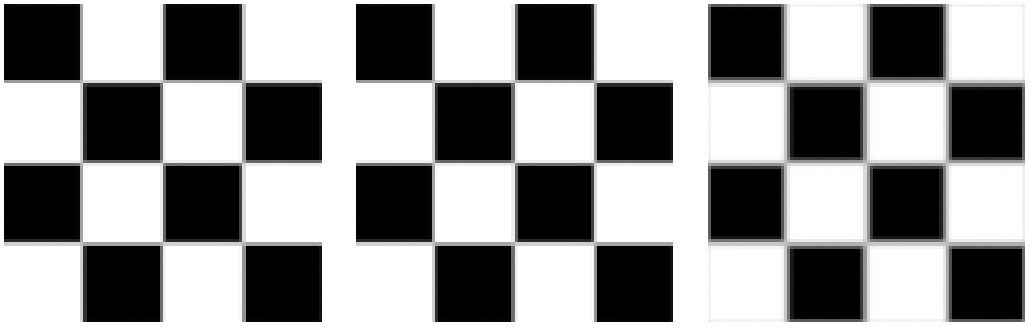
ing function and the width of the domain of which the filter acts upon. Changing the weighting function can preserve or eliminate different frequencies of noise, see figure 4.7 for comparisons and refer to (Pharr and Humphreys, 2010, pg 393) for implementation details of these various filtering functions. Changing the width of these filters will effectively serve as a blurring effect between pixels as samples are accumulated from a domain larger than just the pixel itself, see figure 4.8.



(a) Box (b) Gaussian

(c) Triangle (d) Blackman Harris

Figure 4.7: Comparison of different filter functions

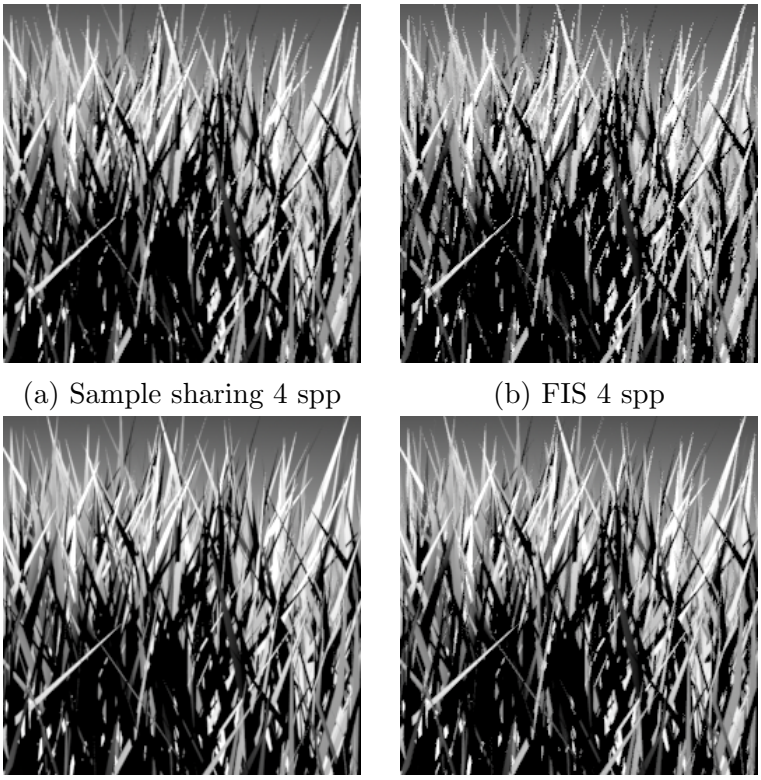


(a) width 1 (b) width 2 (c) width 4

Figure 4.8: Comparison of a Gaussian filter with various widths

However, traditional filtering brings some caveats. Padding must be added to the sampling domain to accommodate filter widths larger than that of a pixel. Samples must be shared across pixels which can be be more challenging to implement especially if each pixel is being rendered in parallel. FIS tackles this problem in a

different way. Instead of weighting samples after the render process has happened FIS will importance sample the filter when creating primary samples. Creating a a distribution that is proportionally distributed according to the filter function which removes the need for samples sharing. FIS has been proven to be an easy to implement technique which improved on traditional filtering techniques in performance and at preserving high frequency noise.



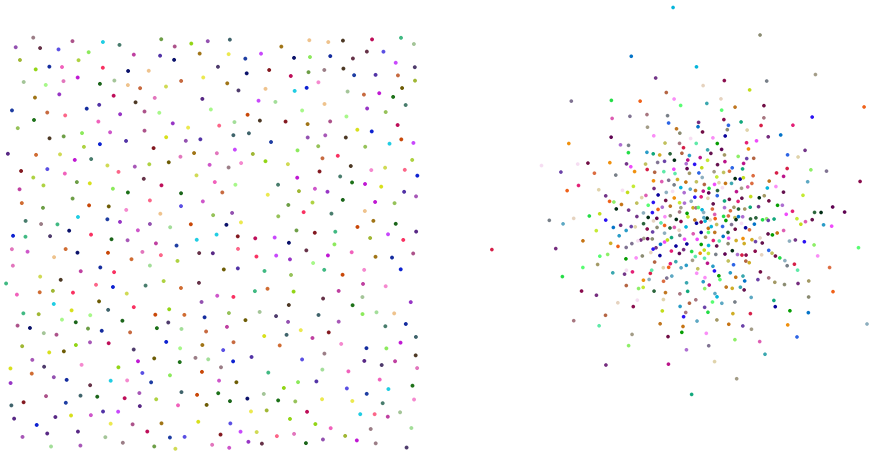
(a) Sample sharing 4 spp (b) FIS 4 spp

(c) Sample sharing 16 spp (d) FIS 16 spp

Figure 4.9: Comparison of sample sharing versus FIS using a gaussian for weights

The limitations of FIS are present twofold. Firstly FIS cannot be used on filters with negative lobes such as a sinc filter. Secondly some functions are difficult to importance sample such as a truncated Gaussian. Sampling these functions often require either the suboptimal rejection sampling or use of importance tables which although works in practice will not be able to exactly distribute samples proportionally to that of the function. Furthermore, we find that using the inversion method could actually deteriorate well stratified sampling distributions as a side effect of the warping applied to it. Let us look at the effect importance sampling a truncated gaussian has on a distribution using the inversion technique. In figure 4.10 we transform a blue noise distribution of samples into an importance sampled truncated normalised gaussian distribution using the Box-Muller transform Martino et al. (2012). Notice in figure 4.10b that although our samples are now distributed proportionally to our truncated gaussian function, as a side effect of the transformation some of the nice properties of our blue noise distribution have been lost. There is very evident clustering of samples towards the centre denser areas of the distribution. Furthermore, our samples are noticeably less evenly distributed like our original blue noise distribution with some samples very sparsely spaced from other samples.

We manage to retain these blue noise properties using adaptive blue noise. In our implementation we use the filter function to drive our adaptive blue noise dis-



(a) (b)

Figure 4.10: Blue noise sample distribution 4.10a and the resulting distribution 4.10b after being transformed with the inversion technique to distribute samples proportional to a truncated normalised gaussian function.

tribution. In our implementation using Jiang et al. (2015) SPH method where we simple use the filter function as the scaler in the size function (see equation 2.19). In figure 4.11 we use this technique to again generate samples distributed proportionally to a truncated normalised gaussian. Notice how because our samples are generated within the sampling space of our filter function rather than having some arbitrary transformation that is ignorant to the distribution, we better retain the blue noise properties of our distribution. We no longer have the issue with samples clustering like the previous inversion method and our samples are much more evenly distributed across our domain.

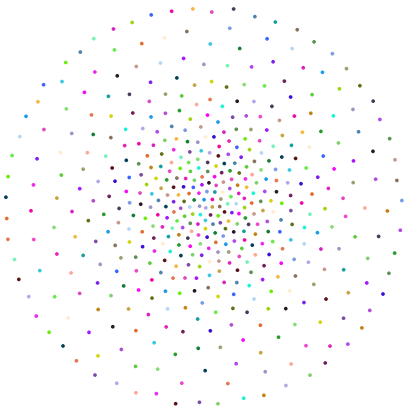


Figure 4.11: Adaptive blue noise samples generated proportionally to a truncated normalized gaussian function

4.2.1 Results

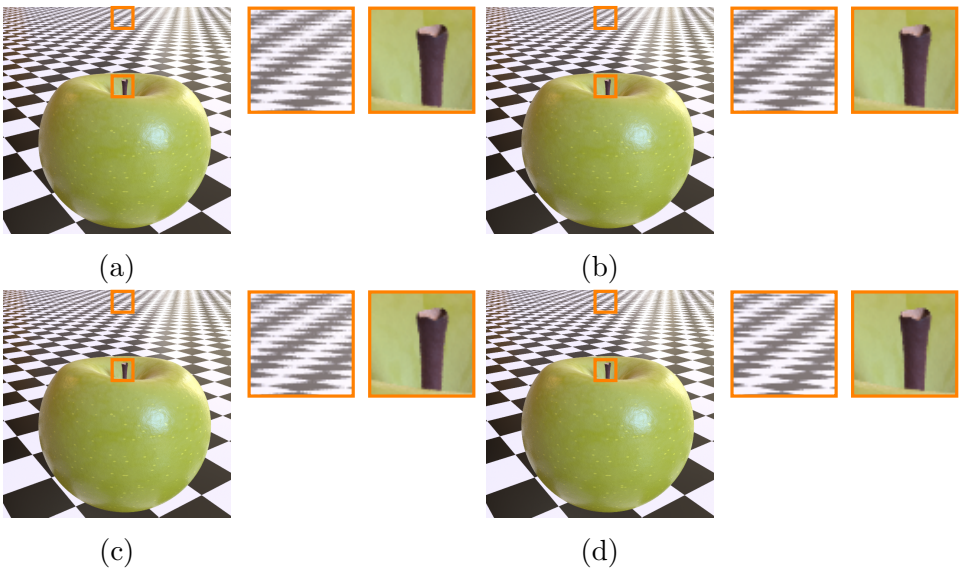
In our results we again use Monte Carlo integration and point sampling to scale a large resolution image to a smaller resolution image just like in section 4.1.1. In figure 4.15 we compare four different sampling techniques, Jitter (figure 4.12a), blue noise (figure 4.12b), blue noise transformed with the inversion technique to importance sample a truncated normalised gaussian (figure 4.12c) and finally our adaptive blue noise also adapted to the truncated normalised gaussian (figure 4.12d). For our blue noise we also use multiclass so that we can select a sub class of blue noise samples for each pixel to reduce correlation. These classes represented by different colours in figures 4.10 and 4.11. If the filter function is symmetrical like most filter functions are we can further decorrelate these samples using dithering. This is the process of applying a random amount of rotation to the sampling distribution and can vastly improve visual fidelity. See (Georgiev and Fajardo, 2016) for the current state of the art in this.

4.2.2 Conclusion

As seen in figures 4.15 using our adaptive blue noise method for generating samples for FIS achieves far superior results to all the other techniques producing visibly less noisy images. This is further shown in the root mean squared error of the images. As you can see in table 4.1 using adaptive blue noise hs significantly reduced the error when compared to the inversion technique in all the scenes that we tested. Using this technique will improve convergence rate and therefore reduce the computation needed to create a fully converged image.

Percentage perceptual mean error improvement	
Apple Scene	24.23%
Cornell Box Scene	2.25%
Bathroom Scene	0.13%
Bedroom Scene	10.25%

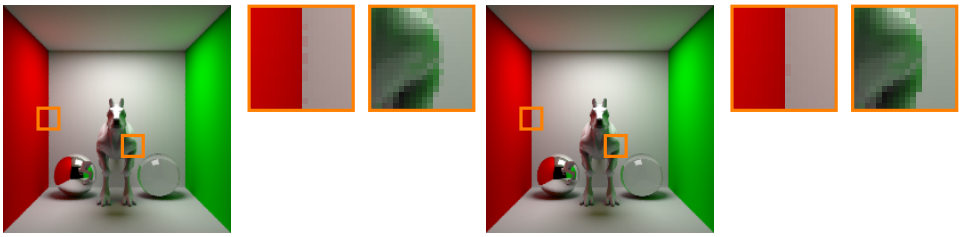
Table 4.1: Root mean squared error of results when compared to ground truth image



(a) (b)

(c) (d)

Figure 4.12: Comparison of filter importance sampling techniques using 10 samples per pixel. 4.12a Jitter no FIS 4.12b blue noise no FIS 4.12c blue noise transformed with the inversion technique to importance sample a truncated normalised gaussian and 4.12d our adaptive blue noise also adapted to the truncated normalised gaussian.

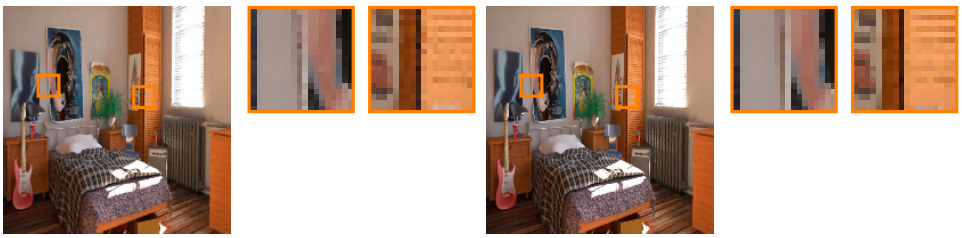


(a) (b)

Figure 4.13: Cornell box scene rendered with 10 FIS samples using 4.13a inversion technique and 4.13b our adaptive blue noise method.

(a) (b)

Figure 4.14: Bathroom scene rendered with 10 FIS samples using 4.14a inversion technique and 4.14b our adaptive blue noise method.



(a) (b)

Figure 4.15: Bedroom scene rendered with 10 FIS samples using 4.15a inversion technique and 4.15b our adaptive blue noise method.

Chapter 5

Applications of Adaptive Blue Noise within BxDF Sampling

In this chapter we show how adaptive blue noise sampling can be used to improve sampling when evaluating shading points in the ray tracing algorithm. This is the process of integrating the incoming radiance to a shading point and how it scatters according to the properties of the material associated with said surface. Physically based materials are defined with one or more BxDFs. The term BxDF encompasses a family of mathematical models that describe the way that light scatters when interacting with a surface. This family includes bidirectional reflectance distribution function (BRDF), bidirectional transmission distribution function (BTDF) and bidirectional specular distribution function (BSDF) which encompasses both BRDFs and BTDFs. During our surface integration when a ray hits a object we use the BxDF (or collection of BxDFs) associated with that object to compute several properties. Firstly the BxDF defines the illumination at the point on the objects surface and secondly how the light will be scattered. Reflectance of real world surfaces can be generalized into a mixture of four categories diffuse, glossy specular, perfect specular and retro reflective (figure 5.1).

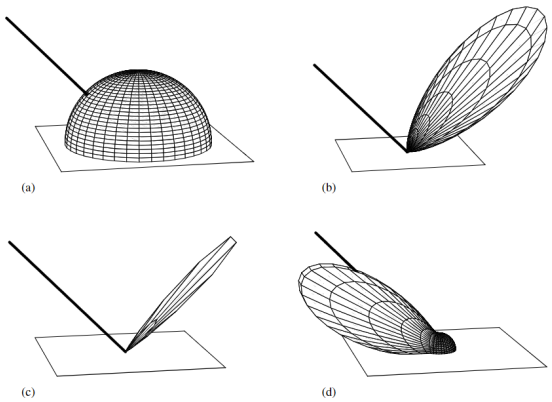


Figure 5.1: (a) Diffuse, (b) glossy diffuse, (c) perfect specular, and (d) retro reflective distribution lobes Pharr and Humphreys (2010)

The simplest distribution to model is diffuse. Diffuse scatters light equally in all directions and can be thought of as matte like materials such as paper. Glossy is a representation of specular surfaces such as plastics. Perfect specular scatters incident light in a single outgoing direction and represents mirror or glass like surfaces. Finally, retro-reflective surfaces primarily reflect light back towards incident

direction such as materials like velvet.

The most important functionality of our BxDF is to calculate the ratio of reflected radiance I given the incident direction w_i and reflected direction w_r of our light. One simple example of this is the diffuse BRDF which evaluates to the cosine of the angle between the incoming light direction and the normal of the surface N.

$$I(w_i) = w_i \cdot N \tag{5.1}$$

As we can see in equation 5.1, as the surface normal tends towards the incoming light direction our reflected radiance tends towards 1. This is shown in figure 5.2.

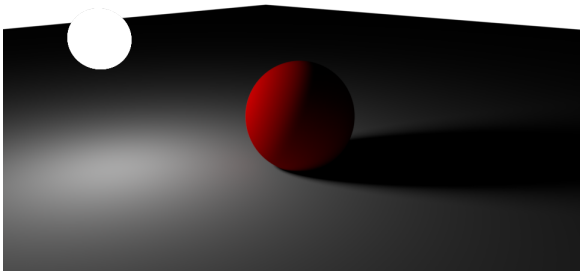


Figure 5.2: Sphere with diffuse BRDF applied.

As mentioned before our diffuse lobe scatters light equally in all directions. However due to our BRDF giving low radiance with rays that tend towards the perpendicular plane of our normal this can lead to tracing rays that ultimately give a very low contribution to the total illumination of the path. Therefore it is a common technique to improve performance by stratifying samples to trace more rays in directions that are proportional to the contribution of the BxDF.

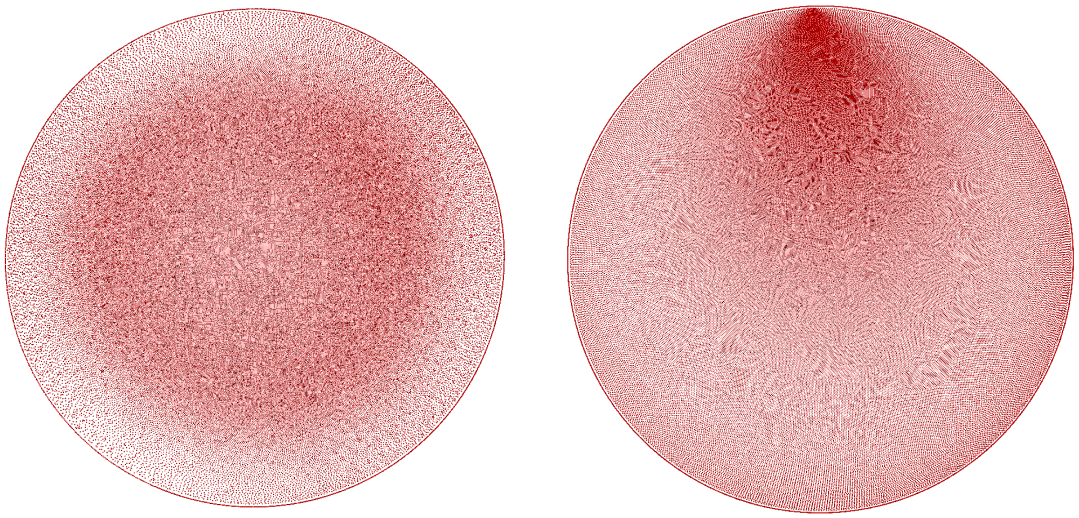
5.1 BxDF Sampling Using Adaptive Blue Noise

Our goal with this implementation is to create a technique that can produce a high quality adaptive distribution for BxDF functions that are difficult or inefficient to produce with the current standard of importance sampling outlined in section 2.4. Our implementations of our proposed technique again adopts Jiang et al. (2015) method of blue noise sampling and adapt it to use to sample BxDF models due to its unique ability to generate samples across any arbitrary surface, in this case our BxDF lobe. Our method constrains particles in our SPH simulation to lie upon the surface of an appropriate lobe shape. e.g. for the case of diffuse materials hemisphere to mimic the distribution over a BxDF lobe. Constraining our simulation can be easily accomplished by mapping particles back to the surface of the lobe after every step of the simulation. This provides an even distribution of stochastic samples across our lobe.

Without importance sampling, blue noise samples alone would give vastly inferior results than the previously mentioned techniques 2.4. Our method achieves importance sampling of arbitrary BxDFs in a very simplistic manner. As described in section 2.5 Jiang's method supports adaptive sampling through the use of some defined scaling function $s(x_i)$ to warp the relative distance between particles used in our smoothing kernel. We design this scaling function to be inversely equivalent to our BxDF. In our implementation we use equation 5.2.

$$
\begin{aligned}
s(x_i) &= a(1 - I(x_i)) + c_{min} \\
a &= c_{max} - c_{min}
\end{aligned}
\tag{5.2}
$$

Where $I(x_i)$ is our radiance intensity of our BxDF at location x_i across our lobe. This in turn results in samples becoming denser as $I(x_i)$ tends towards it maximum value of the BxDF function. Using this method of adaptive sampling gives us the ability to support any arbitrary BxDF without the need for any analytic development. An example of this can be seen in figure 5.3a where we use a simple diffuse BRDF where $I(x_i) = x_i \cdot n$ where n is the normal of our surface.



(a) Diffuse (b) Blinn

Figure 5.3: Top down view of various BxDF models approximated with our adaptive blue noise samples

5.2 Results

So far we have created a BxDF plugin for Pixar's Renderman that generates and uses adaptive blue noise samples for the diffuse BRDF model. Our current unoptimized simulation takes approximately 60 seconds to converge 100,000 samples on a NVidia 970m graphics card and a Intel Core i7-4710MQ CPU @ 2.50GHz. Due to this large convergence time this technique is not yet practical for online sample generation. In this early implementation we pre-generate samples offline and use the same sampling distribution for every shading point of integration.

5.3 Conclusion

We can see in figures 5.4 and 5.5 that with a small number of BxDF samples our adaptive blue noise method out performs traditional inversion. However as the number of BxDF samples are increased we see an increase in correlation. This causes error in our results and ultimately leads to our method being out performed by the inversion method. We believe that removing these correlations will prove to create an effective method that will outperform the inversion technique in all cases regardless of the number of samples.

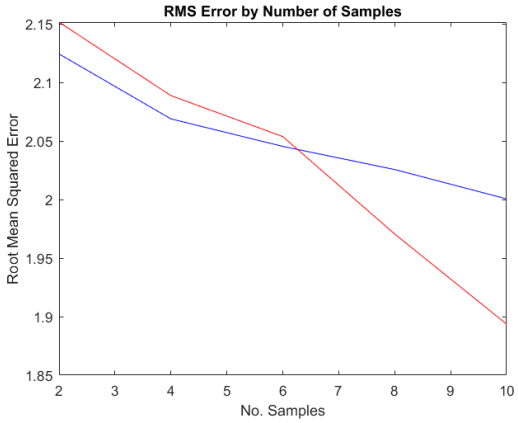
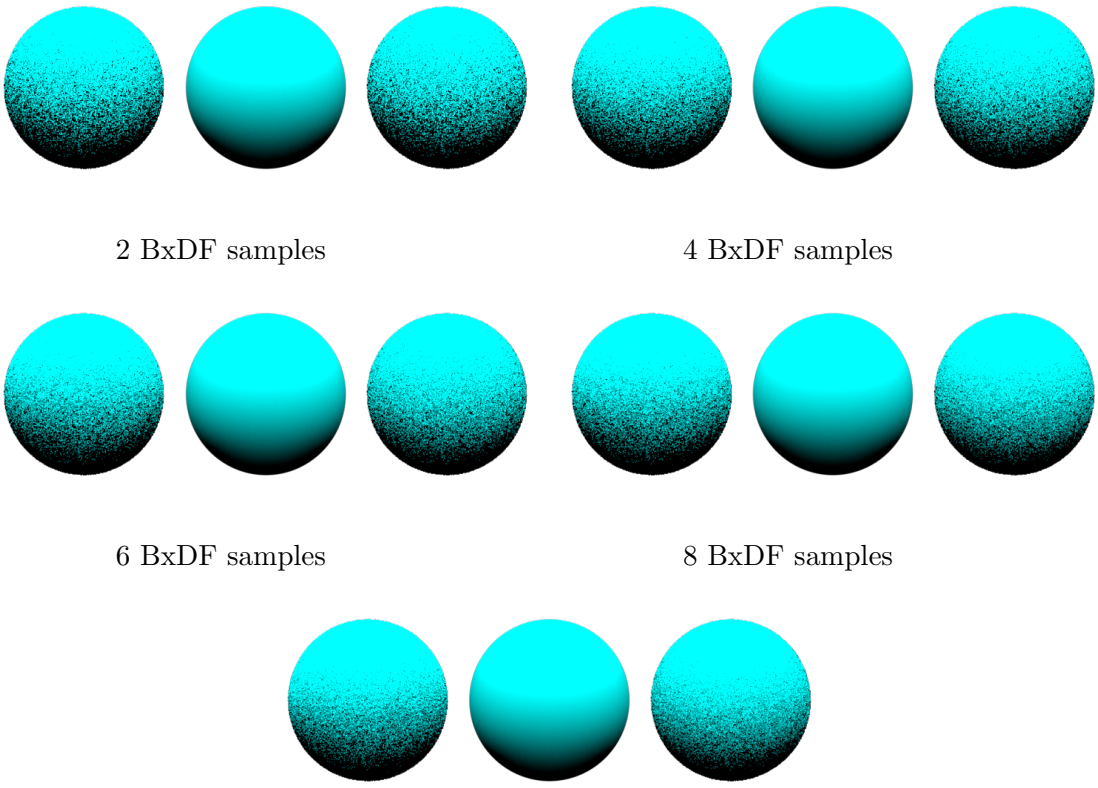


Figure 5.4: Root mean squared error of our blue noise samples (blue) and the inversion method (right) when compared to our ground truth image



2 BxDF samples

4 BxDF samples

6 BxDF samples

8 BxDF samples

10 BxDF samples

Figure 5.5: Renders of diffuse sphere produced with Pixar's Renderman using our blue noise sampling (left), the inversion method and some low discrepancy random sequence (right) and our reference image produced with 500 primary ray samples and 10 BxDF samples (centre). All renders were done using a single primary ray sample.

Chapter 6

Conclusion and Future Work

In this report we have discussed the challenges that are faced on a regular basis in the field of Monte Carlo Ray Tracing which in turn create real world problems. We demonstrated the importance of using high quality stratified sampling distributions and how their usage can be a very powerful tool in improving efficiency in the Monte Carlo estimator. Furthermore, we have shown that blue noise is a high quality sampling distribution and is a good candidate for professional graphics. We have given examples to support this argument and shown that while perhaps on the surface Quasi-Monte Carlo sequences may appear to be a very attractive choice of sample generation there are scenarios in which this can be inferior to blue noise distributions. Secondly we have discussed the limitations faced in the current methods used for importance sampling. Finally we have proposed the foundations of several new techniques that use adaptive blue noise sample distributions to address some of these problems. We believe that initial results of these techniques look promising in providing higher quality sampling which will improve convergence rates and therefore reduce computation and render times. However it is evident that there is still much future work to be taken to make them practical in real world applications.

As expected, we have not solved the problem of sampling in this report. Sampling is still an open problem which will likely indefinitely be improved. There is a vast amount of further research that can be conducted even on the techniques proposed in this report. In the rest of this section we will discuss our ideas on how our research could be progressed in future work.

6.1 Primary Ray Sampling

With the currently proposed method the issue with sample clustering is one that must be addressed to make this technique viable in practical applications. In its current state this method will actually deteriorate as the samples per pixel increases and likely introduce error through bias in the result much like metropolis ray tracing. Work to improve this could either be through simulating progressively more samples at each iteration to normalize the SPH distribution or somehow normalizing the contribution already traced samples have upon the new samples generated. If this problem can be solved we can experiment by using more interesting scaling functions to drive the scaling function used to adapt the samples. So far we have only looked at the use of variance as our scaling function. However, it could be beneficial to use other either prior known properties or information gathered during rendering of our scene to aid adapting samples to interesting regions. This could include geometric

information for edge detection, surface information such a albedo to lower samples of dark objects that absorb most indirect light or if you want to focus on areas that contribute more radiance in a scene we could use some pre-baked information such as occlusion maps to minimise samples sent to areas where light is not apparent.

6.2 BxDF Sampling

The current state of the proposed BxDF sampling method is not practical with the correlations caused by using the same sampling pattern. There are two ways in which this could be solved. The first, and most naive way would be to simply optimize the implementation to make the generation of samples on the fly practical. Currently as mentioned in section 5.2 our simulation takes 60 seconds to converge. This is not a viable computation time when we could have thousands of shading points across our surface that will each require different uncorrelated distribution of samples. A simple approach to improving performance is to first try to find the most optimal parameters used to drive the SPH simulation to reduce convergence time. This however is likely to be a very ad hoc process as there are many variables that contribute to the speed of convergence. Furthermore one set of chosen parameters may be optimal for one set size of N samples but not for another. Another approach could be to have multiple online simulations running concurrently. Using these simulations we would select the simulation that has generated samples with conditions closest to our current shading point. For example with Blinn we could compare half angle vectors and use this to select the simulation converged with the most similar vector. This simulation would then become the starting point of convergence rather than starting from white noise, meaning that samples would be closer to convergence as soon as the simulation begins.

If the correlation problem can be solved efficiently further areas of of interest would be to research into including other scene properties such as light positions in our scaling function. Therefore making our method also a viable approach for generating multiple importance samples. This is unachievable with the traditional inversion method and hard to achieve with Metropolis without causing correlations in successive samples. For more on multiple importance sampling refer to (Pharr and Humphreys, 2010, pg 690).

6.3 Direct Light Sampling

Direct lighting is a crucial optimization to allow path tracing to be a practical technique. It is the process of evaluating emission from light sources of a scene from a shading point on a surface. As seen in figure 6.1 due to the low probability of indirect sampling alone hitting a light source it would take an extremely high sample count to converge the image.

Sampling lights requires selecting a random point upon the light and tracing "shadow rays" which test if the path from the shading point to this point on light source is occluded. If so we can conclude that this path does not contribute any light to the aforementioned shading point. By increasing the number of samples we take of a light source, we increase the accuracy of the approximation of radiance contributed from the light source. However, with every ray traced comes added computation. Furthermore as previously mentioned in section 1 sampling lights efficiently is challenging and is a broad area of research in itself.

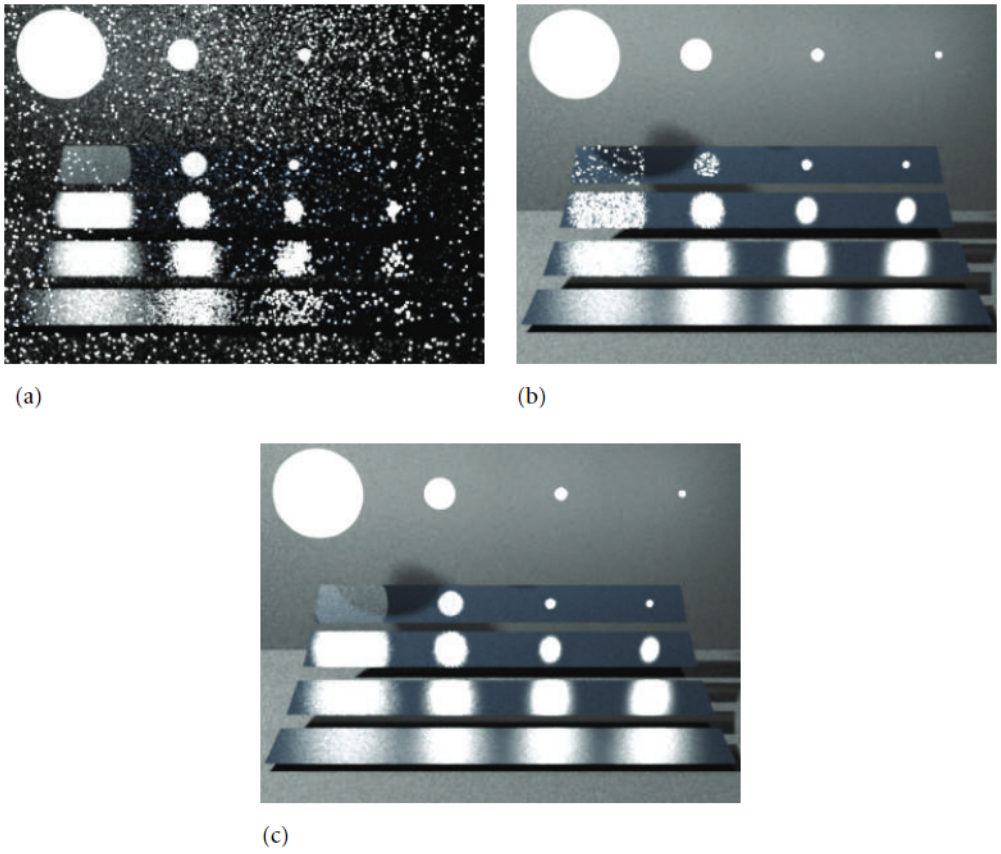


Figure 6.1: (a) Indirect lighting (b) Direct lighting (c) both combined with multiple importance sampling. Image from (Pharr and Humphreys, 2010, pg 857)

Much of the literature available has been applied to lights of simple geometric shapes such as quad lights. Kok and Jansen presents a method to reduce the number of occluded shadow rays traced. This is done by adaptively subdividing quad lights based on the success of a shadow ray and increasing the probability of sampling successful regions. In more recent developments, Solid Angle have made some impressive contributions in regards to light sampling. Presenting techniques for quad (Ureña et al., 2013), ellipses (Guillén et al., 2017) and spherical light sampling (Urea and Georgiev, 2018). These techniques improve on existing methods with higher quality importance sampling of the light source. Results have been shown to be almost noise free results with direct lighting as they exactly importance sample the light source and prove particularly effective within participating media. However, these techniques are optimized for each shape and do not generalize to lights of other shapes. Mesh lights are a phenomenon that is proving ever more popular in its use in rendering. This is the case where an entire mesh of arbitrary shape and number polygons is treated as a light. The ambiguity of how this mesh is defined makes these lights difficult to sample. A naive solution to this is to treat each polygon as its own unique light source. However, with a basic light loop this could lead to evaluating many thousands if not more lights at each shading point which would be very inefficient. Portsmouth (2017) presents an unbiased method of efficient barycentric point sampling on meshes that uses rejection sampling (see 2.4). However, the rejection method, given its random nature cannot guarantee how many samples will need to be taken to produce an accepted result which could prove costly in practice. This method is also limited to triangle shaped polygons which although is generally acceptable given that it is common practice in most

renderers triangulate geometry as a pre-process regardless, it does mean that the memory footprint of the defined mesh will increase. Estevez and Kulla (2017) treats each polygon as a unique light source however builds an adaptive tree acceleration structure upon them. This tree is sorted by the importance of a polygon such as the emission produced. A subset of polygons will then be selected to sample rather than the entire set. While this proves effective in practice it is possible to created a scenario where this method becomes sub optimal. For example, if a single polygon of the mesh light is significantly brighter than the rest however is completely occluded. This would cause higher likelihood of this polygon being sampled even though it would not contribute and light to the scene.

Given that Jiang et al. (2015) method can effectively generate samples on arbitrarily defined geometry we believe that this would prove as a good method for mesh light sampling. Furthermore it would be interesting to investigate how adaptive sampling could be used to improve mesh light sampling. A simple idea would be to adapt the samples to importance sample the illumination of the light if the illumination across the light is non-uniform. Other ideas could be to use some preprocessed data to identify areas of the geometry that are completely occluded either by the mesh itself or by other objects in the scene.

Acknowledgements

Firstly I would like to thank my supervisors Richard Southern and Ian Stephenson for all their advice and support in making this thesis a reality. Secondly I would like to thank all the guys at Solid Angle who have taught me more than I ever thought I could learn about production rendering. Finally I would like to thank all my family and friends for keeping me sane after countless hours of staring at dots on a screen and my cats for occasionally deciding to not sit on my keyboard to allow me to work.

Bibliography

B. Owen, A. (1992). Orthogonal arrays for computer experiments, integration and visualization. 2:439–452.

Balzer, M., Schlömer, T., and Deussen, O. (2009). Capacity-constrained point distributions: A variant of lloyd's method. *ACM Trans. Graph.*, 28(3):86:1–86:8.

Bratley, P. and Fox, B. L. (1988). ALGORITHM 659: implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 14(1):88–100.

Bridson, R. (2007). Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA. ACM.

Chiu, K., Shirley, P., and Wang, C. (1994). *Graphics Gems IV*. Academic Press Professional, Inc., San Diego, CA, USA.

Cohen, M. F., Shade, J., Hiller, S., and Deussen, O. (2003). Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294.

Cook, R. L. (1986). Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72.

Dippé, M. A. Z. and Wold, E. H. (1985). Antialiasing through stochastic sampling. *SIGGRAPH Comput. Graph.*, 19(3):69–78.

Du, Q. and Emelianenko, M. (2006). Acceleration schemes for computing centroidal Voronoi tessellations. *Numerical Linear Algebra with Applications*, 13(2-3):173–192.

Dunbar, D. and Humphreys, G. (2006). A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics*, 25(3):503.

Ernst, M., Stamminger, M., and Greiner, G. (2006). Filter importance sampling. In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 125–132.

Estevez, A. C. and Kulla, C. (2017). Importance sampling of many lights with adaptive tree splitting. In *ACM SIGGRAPH 2017 Talks*, SIGGRAPH '17, pages 33:1–33:2, New York, NY, USA. ACM.

Georgiev, I. and Fajardo, M. (2016). Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, pages 35:1–35:1, New York, NY, USA. ACM.

Georgiev, I., Křivánek, J., Davidovič, T., and Slusallek, P. (2012). Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192:1–192:10.

Guillén, I., Ureña, C., King, A., Fajardo, M., Georgiev, I., López-Moreno, J., and Jarabo, A. (2017). Area-preserving parameterizations for spherical ellipses. *Comput. Graph. Forum*, 36(4):179–187.

Harris, M. (2008). Cuda fluid simulation in nvidia physx. *SIGGRAPH Asia 2008*, Parallel Computing for Graphics: Beyond Programmable Shading.

Heck, D., Schlömer, T., and Deussen, O. (2013). Blue noise sampling with controlled aliasing. *ACM Trans. Graph.*, 32(3):25:1–25:12.

Hoetzlein, R. (2014). Fast Fixed-Radius Nearest Neighbors: Interactive Million-Particle Fluids. *GPU Technology Conference (GTC)*.

Jarosz, W. (2008). *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. PhD thesis, UNIVERSITY OF CALIFORNIA, SAN DIEGO.

Jensen, H. W. (1996). Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK. Springer-Verlag.

Jiang, M., Zhou, Y., Wang, R., Southern, R., and Zhang, J. J. (2015). Blue noise sampling using an SPH-based method. *ACM Transactions on Graphics*, 34(6):1–11.

Jones, T. R. (2006). Efficient Generation of Poisson-Disk Sampling Patterns. *Journal of Graphics, GPU, and Game Tools*, 11:27–36.

Kok, A. J. F. and Jansen, F. W. Adaptive sampling of area light sources in ray tracing including diffuse interreflection. *Computer Graphics Forum*, 11(3):289–298.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Martino, L., Luengo, D., and Miguez, J. (2012). Efficient sampling from truncated bivariate gaussians via the box-muller transformation. 48:1533–1535.

McCool, M. and Fiume, E. (1992). Hierarchical poisson disk sampling distributions. In *Proceedings of the Conference on Graphics Interface '92*, pages 94–105, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal Chemical Physics*, 21(6):1087–1092.

Mitchell, D. P. (1987). Generating antialiased images at low sampling densities. *ACM SIGGRAPH Computer Graphics*, 21(4):65–72.

Mitchell, D. P. (1991). Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph.*, 25(4):157–164.

Mitchell, D. P. (1996). Consequences of stratified sampling in graphics. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 277–280, New York, NY, USA. ACM.

Monaghan, J. (1994). Simulating free surface flows with sph. *J. Comput. Phys.*, 110(2):399–406.

Okten, G., Shah, M., and Goncharov, Y. (2012). Random and Deterministic Digit Permutations of the Halton Sequence. *Springer Proceedings in Mathematics and Statistics*, 23(January 2009):609–622.

Per, C., Andrew, K., and Charlie, K. (2018). Progressive multi-jittered sample sequences. *Computer Graphics Forum*, 37(4):21–33.

Pharr, M. and Humphreys, G. (2010). Physically Based Rendering. *Physically Based Rendering*, (April):738–871.

Portsmouth, J. (2017). Efficient barycentric point sampling on meshes. *CoRR*, abs/1708.07559.

Priscott, C. (2010). 3D Langrangian Fluid Solver using SPH approximations. Master's thesis, Bournemouth University.

Reinert, B., Ritschel, T., Seidel, H.-P., and Georgiev, I. (2016). Projective blue-noise sampling. *Comput. Graph. Forum*, 35(1):285–295.

Schechter, H. and Bridson, R. (2012). Ghost sph for animating water. *ACM Trans. Graph.*, 31(4):61:1–61:8.

Shirley, P. (1991). Discrepancy as a quality measure for sample distributions. In *In Eurographics '91*, pages 183–194. Elsevier Science Publishers.

Ureña, C., Fajardo, M., and King, A. (2013). An area-preserving parametrization for spherical rectangles. In *Proceedings of the Eurographics Symposium on Rendering*, EGSR '13, pages 59–66, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Urea, C. and Georgiev, I. (2018). Stratified sampling of projected spherical caps. *Computer Graphics Forum*, 37(4):13–20.

Veach, E. (1997). Robust Monte Carlo Methods for Light Transport Simulation. *Dissertation at the Department of Computer Science of Stanford University*, 134(December):759–764.

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, pages 65–76, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Ward, G. J., Rubinstein, F. M., and Clear, R. D. (1988). A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pages 85–92, New York, NY, USA. ACM.

Wei, L.-Y. (2008). Parallel poisson disk sampling. *ACM Trans. Graph.*, 27(3):20:1–20:9.

Wei, L.-Y. and Wang, R. (2011). Differential domain analysis for non-uniform sampling. *ACM Transactions on Graphics*, 30(4):1.

White, K. B., Cline, D., and Egbert, P. K. (2007). Poisson disk point sets by hierarchical dart throwing. In *2007 IEEE Symposium on Interactive Ray Tracing*, pages 129–132.

Whitted, T. (1980). An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349.