Techniques on Mining Effective Parallelism For High-Performance Blue-Noise Sampling
（高性能ブルーノイズサンプリング並列処理技術）

by

WANG Tong
王桐

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on June, 7, 2019
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology
in Computer Science

Thesis Supervisor: Reiji Suda　須田礼仁
Professor of Computer Science

ABSTRACT

Stochastic sampling is a core concept for many applications across different fields in signal processing, computer graphics, and computer vision. The sampling process reduces a continuous signal to discrete-time signals, which makes signal processing possible on computers. It is widely accepted that a sample pattern with a blue noise spectrum appears to be better in capturing underlying features of the continuous signal. A blue noise sample pattern means that the power spectrum of the Fourier transform of a sample set appears to have blue noise feature: that it reveals low energy during lower frequency domain while extending to a higher energy level during high-frequency domain. Such sample patterns can help applications avoid aliasing, achieve better visual quality. It also serves as the core algorithm in stippling and halftoning, when the underlying density map is used. Among which the Poisson-disk sampling pattern is proved to be one of the best choices in blue noise sampling pattern. This specific sampling pattern can help the sampling process achieve a high sampling rate while maintaining relatively good quality.

In this thesis, high-performance generation methods of blue noise sampling pattern, especially Poisson-disk sampling will be studied and presented. The contribution of this thesis comes in three-part: the first part is a novel method named KD-tree based Randomized Tiling (KDRT) to generate maximal Poisson-disk sample patterns on Euclidean space in multiple dimensions, with state-of-the-art generation rate while maintaining good sampling quality. The second part will focus on how to utilize the KDRT's generated pattern to apply the planar Poisson-disk sample pattern to sampling mesh surfaces, which is also a very important application in computational geometry and computer graphics. A progressive sample projection (PSP) method based on parallel ray-tracing will be used, which pushed the parallelism and performance of Poisson-disk sampling on the mesh surface to a new level. In the third part, a novel relaxation-based algorithm optimizing over constrained distances will be proposed to further optimize the blue noise feature of generated samples, which will be called the distance constrained relaxation (DCR). A new second-order function will be used to alleviate problems in parallel computing of relaxation-based blue-noise samples. while a relaxation based synthesis method based on distance constraints will also be presented as well, in order to satisfy applications requiring sampling with a certain final state.

Maximal Poisson disk sampling is one of the blue noise sampling patterns that can benefit various applications mentioned above. Previous methods often fail to provide enough sampling rate for usage in real-time applications or suffer from a trade-off on sampling quality. To address this problem, we propose a novel tile-based stochastic sampling method, that can provide tremendous improvements in performance (measured by sampling rate) over literature while maintaining the maximal property that most applications could benefit from. The core idea is to clip samples randomly from a predefined sample set, then tile all clips to the sample domain. To eliminate sample conflicts from this operation, a divide-and-conquer method will be applied to all clips of the tile. The method is a two-step process: divide sample space into clips randomly with KD-tree based pattern, then conquer the clips together with our proposed conquering algorithm. We also propose a generally applicable parallelization method for any other maximal Poisson-disk generation method, developed from the conquering algorithm.

Secondly, sampling in Non-Euclidean space is of crucial importance in computer graphics, especially considering sampling 2D manifold, which mostly would be represented as 3D space mesh surfaces. It is often the most important pre-processing step in applications such as re-meshing, texture synthesis, point-based learning and point-based reconstruction. Most previous methods can hardly provide enough performance for real-time use, or suffer from a painful trade-off between quality and sampling rate. Meanwhile, methods that can achieve a relatively good sampling rate are often not memory bounded or require atomic operations during parallel processing, which greatly harmed their parallelism. In the second part of the thesis, we developed a straight forward progressive sample projection method for high-performance sample generation on mesh surfaces. It can generate massive Poisson-disk samples on mesh surfaces in a very short time by projecting blue noise sample patterns from 2D planar space onto meshes. This parallel scheme can exploit full parallelism of SIMD hardware such as GPU without deep recursion or atomic operations, which are often required by other methods. The effec-

tiveness of the method can guarantee most usage in real-time applications, while also being progressive with memory usage bounded, thus being flexible for both performance and quality demanding work. The method can be easily applied to adaptive sampling as well, which could be crucial in applications such as stippling.

And finally, for some more demanding work like low aliasing rendering, re-meshing, etc., the Poisson-disk sampled set need further optimization to achieve a better blue-noise quality level. Applications such as object placements require that a precise number of samples should be synthesized instead of a specified Poisson-disk radius. For such conditions, a relaxation-based method will be needed to optimize an Energy function defined on the sample points that can further improve blue noise quality. We developed a novel method to exploits more parallelism from existing relaxation based algorithms by assigning each sample with an "activeness" value calculated from their incident forces received from neighbors. The activeness will dynamically re-distributing tension forces between pairs of current point with its neighbor. The algorithm takes incident forces of each sample as a way to measure whether the sample is stuck in a local balance and cannot be further improved with normal parallel calculation. This mechanism can dynamically detect and resolve such local balance and help the iteration to achieve effective parallelism. We showed that the proposed method can achieve a better global variance than other strategies. We also proposed to use distance constraints, or the edges of Delaunay triangulation of the point cloud to do blue-noise synthesis in this way. A simplified algorithm utilized k-nearest neighbor is also proposed, which is much more GPU friendly.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Professor SUDA, for his consistent, illuminating and invaluable guidance during my doctoral research, also being so generous in offering me chances to attend international conferences and all the resources for undertaking research activities.

Thanks to all reviewers of my Ph.D. dissertation and thanks for your valuable review comments.

I deeply appreciate Nomura-san, our hard-working secretary, for her kindness and patience. Thanks for all the staff in the office of the Computer Science department and their assistance during my study. I would also take the opportunity to thank all group members for their support and help.

Last, my thanks would go to my beloved parents for their loving consideration, constant support and great confidence in me all through these years. Thanks for my wife Sun Yaping, your supports are the most valuable gifts in my life, like the brightest star above the darkest ocean during the longest journey.

This work is supported by Research funding of IST, University of Tokyo. Thanks, the University of Tokyo, for all great school facilities and amazing courses. Thank you, Tokyo, a city that is really fascinating to live in.

Contents

Chapter 1

Introduction

In this chapter, we first explain the background of sampling, especially on why stochastic sampling is introduced. Then we'll discuss the advantages of using blue-noise sampling techniques for demanding sampling applications, as well as why performance matters. Finally, we will introduce some related literature.

1.1 Sampling and Stochastic Sampling

A necessary preliminary process to process continuous analog signals captured from nature on a digital computer is sampling. For example, the smallest unit that most digital display device can show is a pixel, so image processing is inherently a sampling process ([Cook, 1986]) that made a computer capable to process continuous 2D signals. In this section, we will introduce basic backgrounds in sampling and stochastic sampling, which will lead towards discussions of how to define a better sampling pattern.

1.1.1 Regular Sampling

It is well known in signal processing fields that sampling is the process of getting scaled delta functions from the continuous function by multiplying signal with Dirac Comb. Signals can then be reconstructed from the samples with a proper reconstruction kernel. See figure 1.1 for more illustration.

Dirac comb function $III(t)$ is a combination of regular interlaced periodic Dirac functions added together for sampling: $III_T(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT) = \frac{1}{T} III(\frac{t}{T})$. T being the period of the cycle. Using this periodic Dirac comb function to sample continuous function is called *regular sampling*, and is commonly

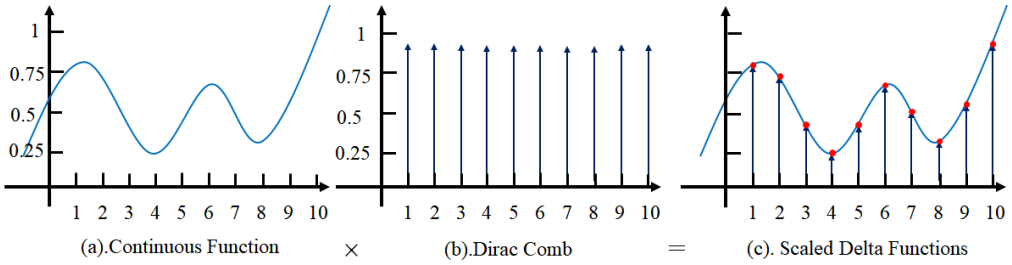


Figure 1.1: Sampling is the process of acquiring discrete representation from a continuous function for digital systems. The function $f(x)$ in (a) will be multiplied by (b) the Dirac comb function (or sampling function $III(t)$). The result is (c), which is an infinite sequence of scaled delta functions that represent the function's value at each sample point.

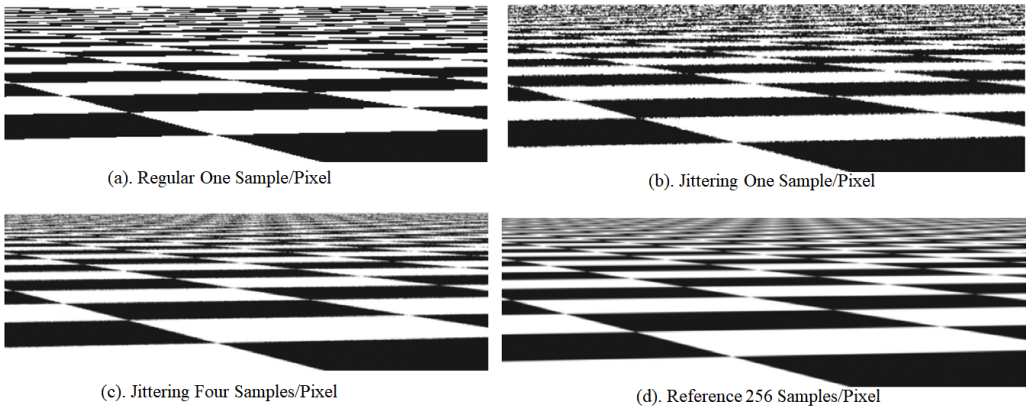


Figure 1.2: In 2D, we can see that regular samples can cause serious aliasing that's often not visually acceptable. While with a simple random jitter applied to the samples, the aliasing problem can be significantly alleviated. (a). Sampling the checkerboard image with regular sampling, one sample per pixel. (b). Sampling the checkerboard image with the jittering sampling, one sample per pixel. (c). Sampling the checkerboard image with the jittering sample, four samples per pixel. (d). Referenced image of the checkerboard, rendered with 256 random samples per pixel.

used in 1-dimensional signal processing algorithms. See detailed explanations in [Lim, 1990] and [Russ, 2016].

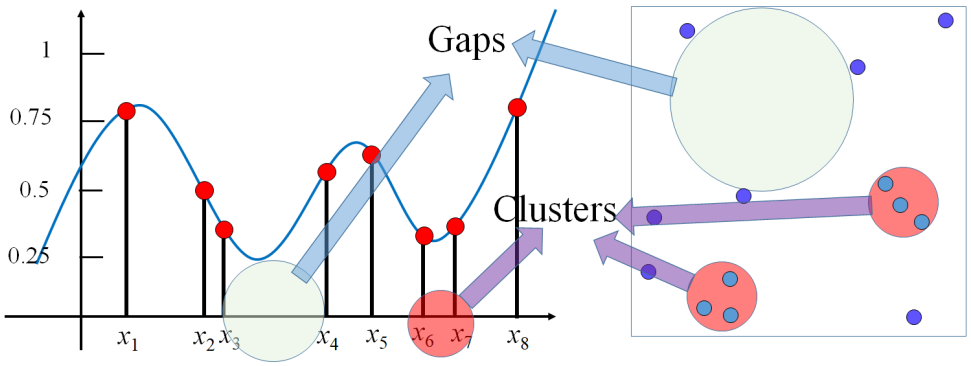
One problem of using such a sampling pattern is that it will cause **aliasing**, which will cause strong artifacts during the reconstruction process, thus considered not favorable. Here is a brief explanation of the aliasing artifacts: The samples generated from the sampling process will be used for reconstruction in a digital system. As we know that sampling is multiplication in the time domain, thus will be equivalent to convolution in the frequency domain. To preserve all information of the signal being sampled, it is required that the sampling frequency should be high enough to interleave the bandwidth of the signal being sampled in the frequency domain. If the sampling frequency is not high enough, the convoluted band-limit function being sampled will be overlapping each other's frequency spectrum. This overlapping will cause information loss after reconstruction.

This phenomenon will also be recognized in the 2D signal (image) sampling as well. The error introduced by aliasing in 2D will be revealed as zigzagging artifacts from a visual perspective, which is often not acceptable for human vision. See Figure 1.2 for examples of aliasing artifacts in 2D sampling, as well as how the artifacts can be alleviated.

1.1.2 Stochastic Sampling

In Figure 1.2 (b) and (c), we get an inspiration that even a slight change in the regular position of regular samples (by the process of random jittering) can greatly reduce the aliasing artifacts caused by under-sampling. Such randomized sampling technique is often referred to as *stochastic sampling*.

An early pioneer that proposed the concept of applying stochastic sampling in computer graphics is [Cook, 1986]. The work stated that in most computer applications including rendering and modeling, one of the best ways to avoid aliasing is to do stochastic sampling instead of sampling at regularly spaced locations.



(a). 1-dimensional stochastic sampling (b). 2-dimensional uniform sampling

Figure 1.3: Problem in randomized sampling.

It turns out that aliasing is a much more severe problem than noise, while with such proper stochastic sampling methods applied, aliasing can be replaced by high-frequency noise with correct average intensity, which is much more friendly to human vision.

Still, the noise should be alleviated as well. From 1.2 we observed that random sampling suffered from under-sampling and high-frequency noise. This is because that random samples often introduce big gaps where signals are under-sampled, as well as space with sample clusters which over-sample the domain, see Figure 1.3 for illustration. An ideal sampling pattern should have unbiased randomness with an evenly scattered feature to benefit from both sampling strategy. Such a sampling strategy that can generate evenly distributed random samples is often called *blue-noise sampling*.

1.2 Blue-Noise Sampling

Blue-noise is a naming convention following the color of noise [Gonzalez et al., 2002]. But empirically as a tradition that blue-noise sample pattern is characterized as samples with low energy in low frequency, then promptly grow on energy with a pretty sharp transition which equivalently means that sample density at approximately the average neighboring distance being very high. The energy then drops to a white noise level beyond. Generally speaking, blue-noise samples are evenly distributed, but also randomly chosen. The discrepancy ([Niederreiter, 1988]) of a blue-noise sample set can be kept low, while the randomness is kept to avoid aliasing caused by regularly sampled artifacts.

In [Cook, 1986], they proposed Poisson-disk sampling as a good example of such a nonuniform distribution of samples that is first being discussed by Yellott [Yellott, 1983] when he did the study on distributions of cone cells of human eyes.

Poisson-disk samples are randomly sampled in the domain, but each sample will be away from another sample than a minimum specified distance d. Figure 1.4 shows that the human eye captures nature to its "pixels" (cone cells), and the cells of human retina arranged in a Poisson-disk sampling pattern to precisely and effectively capture lights from nature.

It is not only staying on a bionic level, but people also proved that sample patterns with a good *blue-noise* property are often suitable to be utilized in applications to control aliasing and depress noise. In this case, we can please the human eye with the best sampling quality.

Here we will introduce how Poisson-disk samples are generated. The process

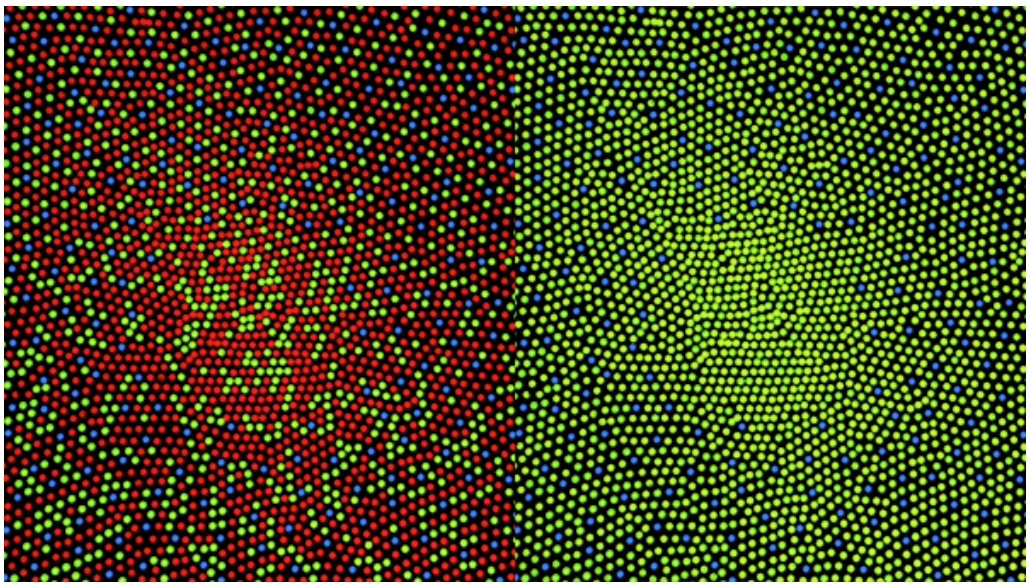


Figure 1.4: Cone cells distribution on human retina. They appear to be forming a Poisson-disk sampling pattern. The left part is an individual with normal color vision, and the right is a color blind retina. Picture courtesy of Mark Fairchild by Wikimedia Common.

of sampling scoped in this thesis can be formulated as a procedure to position samples on a geometry domain so that the samples will be distributed with a specified characteristic. Poisson-disk samples are generated base on two regulation: (1). All samples are generated with uniform random probability, and (2). No samples are closer to each other than a specified distance d. With such easy regulation, the generated sample set will appear to have a blue-noise feature. Poisson-disk sampling is the most common pattern to use as an example of synthesized blue-noise samples.

A common and easy way to generate Poisson-disk samples is called *Dart-throw*. The process is to throw darts in the sample domain, and if it has enough distance from others, keep the dart. On the other hand, if it is within the specified radius with other samples(darts), reject the dart and throw again. As a naive method, it is not an efficient way. Dart-throw prevents clusters but didn't consider gaps that samples may not evenly spread with an effective way. It is also a topic throughout the thesis, that we will propose novel methods on improving the parallel scheme of dart-throw that can make it much more efficient and reliable.

1.2.1 Generation of Blue-noise Samples

Here we will introduce some common generation methods for blue-noise sampling pattern. More specifically related literature will also be introduced in later corresponding chapters.

Dart-throw and Its Variation Methods

For decades people have been exploring various methods to generate blue-noise samples. Naive Poisson-disk sampling like dart-throw seems to be effective at the beginning of the iteration, but the time complexity will soon grow beyond to combinatorial complexity which is not acceptable. While the needs for high-

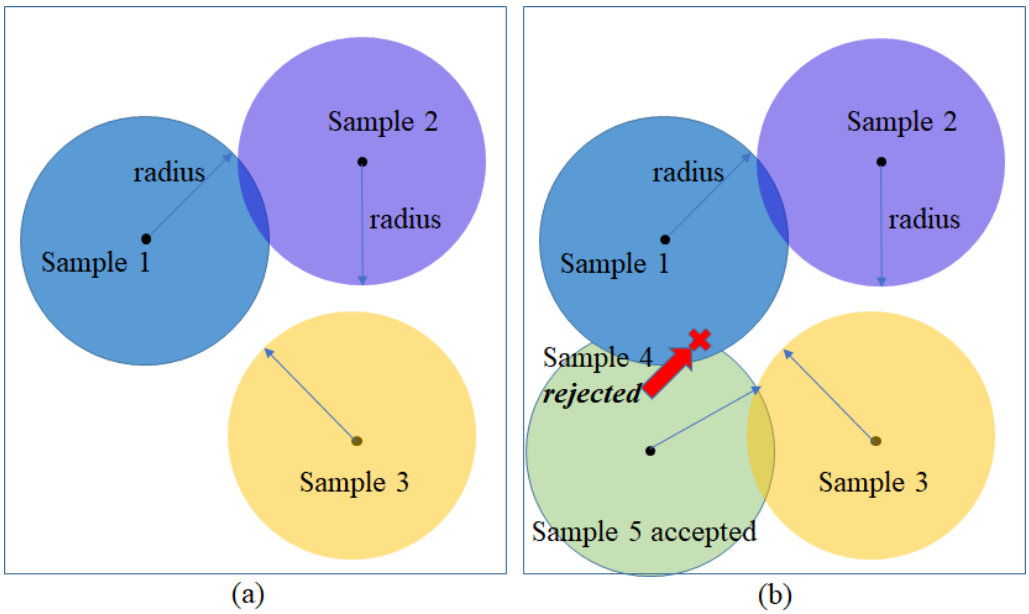


Figure 1.5: Illustration of dart-throw algorithms, the basic method to sample Poisson-disk samples. (a). Samples will be randomly placed in the sample domain, and each will be at least away from each other than a specified radius. (b) New samples will be added to the domain, if the randomly generated new samples (for example, sample 4) is within the conflict radius of a previously existed sample, then sample 4 will be rejected, until a valid sample is generated (for example sample 5). The algorithm cannot effectively insert new samples as the gaps in the sampling domain will be smaller and smaller. Various other methods have been proposed to accelerate this process, with the similar basic idea of throwing darts, and they will be grouped to the dart-throwing family.

quality blue-noise samples growing in the field, various other approaches tried to improve upon the naive dart throw methods to make it good enough for generating off-line usable samples in a reasonable time. See Figure 1.5 for a simple illustration.

Dart-throw based methods can be easily extended to 3-dimensional space or more. But the performance of testing neighboring samples will surely suffer from the curse of dimensionality.

Tile-Based Methods

Besides generating samples from the scratch, use of already generated patterns and combine them to a larger sample set will be the key idea for another family of methods. This group of methods is often referred to as tile-based methods. A famous example will be [Cohen et al., 2003]. Tile-based methods are often significantly faster than the previous one but will suffer from a noticeable bias when studying their properties, as they are not completely random, and sometimes appear to have periodic properties. Also, the pre-processing time needed for this group of methods can be significant for some applications.

Relaxation-Based Methods

Relaxation-based methods are also one of the largest group of methods that can generate high-quality blue-noise sample pattern by changing sample positions

from an initial sample set. Samples will be moved around until the sample set appears to have the blue-noise feature. One advantage is that they can generate a sample set with a fixed number of samples instead of fixed property. Another advantage is that most of the time a relaxation-based sampling method can generate the highest quality blue-noise sample pattern. The disadvantage of this group of methods is that it is an iterative process, thus will need a lot of iterations to converge. The iterative execution will harm performance so that the relaxation-based generation methods are often not suitable for most interactive applications.

We proposed three techniques that seek to exploit full parallelism and reach the highest effective sample rate in **each of these three categories** we discussed above correspondingly. In the following chapter, we will be introducing a KD-tree based randomized tiling method, which belongs to the *tile-based methods*; A progressive projection method belongs to *parallel dart-throw* that can generate Poisson-disk samples on 3D surfaces with high sampling rate, which utilized a special parallel executing scheme that avoids nested recursions and imbalanced workloads; A novel parallelism scheme in *relaxation-based methods* that increase the convergence speed of the iterative process, while seeking to converge on a better local equilibrium. Later we will discuss why we tackle from these directions, and how they are related to each other as a whole to form an effective solution to the high-performance blue-noise sampling problem.

1.2.2 Problem with Blue-Noise Sampling

Blue-Noise Sampling has a good reputation of being a good sample pattern that can avoid aliasing problem and alleviate high-frequency noise. But the generation of good blue-noise samples with fast sampling rate is not easy. Especially when there are quality requirements. Traditional Poisson-disk sampling methods relying on dart-throw are prohibitively expensive. Parallel execution of throwing darts also suffer from low parallelism, thus being unfriendly to be executed on SIMD hardware such as on GPU ([Ying et al., 2013], [Wei, 2008], [Bowers et al., 2010]). A good implementation of such parallel dart-throw methods can reach a maximum of 1 to 2 million samples per second, which is still often below the bar for real-time applications.

There are fast tile-based algorithms, but they suffer from pre-generation and complex implementation problems ([Cohen et al., 2003],[Ahmed et al., 2015]). Such a solution also generate periodic bias that is not favorable for many applications ([Kopf et al., 2006]). The most relaxation-based algorithm, on the other hand, can generate high-quality samples by morphing an initial sampling set, but the problem is that these iterative algorithms often need hundreds of iterations to get to convergence states ([Ahmed et al., 2016]). Blue-noise sampling can ensure a high-quality stochastic sampling quality in most applications, but the most serious problem preventing its wide usage in various applications will still be its performance (or sampling rate, samples generated per second). Because of these observations, the main focus of this thesis will be tackling the performance issue that prevents using the high-quality blue-noise sampling from some important applications.

1.3 What the Thesis Trying to Solve

Here we'll discuss the problem that the thesis trying to solve, with a brief introduction on the scope of the thesis as well. In the following chapters, we will

focus on mining effective parallelism in three aspects to push the sampling rate of blue-noise samples to the limit. Interactive applications or even real-time applications that previously may only be able to use deterministic low-discrepancy sequences will now have chances to do stochastic sampling with a much more reasonable and high-quality blue-noise sample pattern with methods introduced through this thesis.

The following are the reasons that the thesis tries to tackle the problem in these three parts:

Firstly, we need a base algorithm that can generate the most commonly used 2D blue-noise sampling pattern with extremely fast performance. We also need assurance of the 2D blue-noise pattern so that it can be further used in other sampling strategies as an input. To achieve this goal, we try to develop a tile-based algorithm named KD-tree based Randomized Tiling (**KDRT**). To simplify the implementation and avoid too much pre-processing, we choose to tile the sampling domain using building blocks clipped from one single pattern filled with maximal Poisson-disk sampling without the extra necessity of pattern set. We also managed to deal with conflicts and gaps by extra gap detection, rather than leave them untackled as in [Kalantari and Sen, 2012]. The final results are ensured to have maximal-coverage property, which is a quality assurance needed for further usage of such pattern by the following methods.

Secondly, we consider how such a 2D planar space sampling can benefit sampling blue-noise pattern on mesh surfaces. Surface sampling is of great importance for many computer graphics applications such as texture synthesis, point-based learning, and mesh reconstruction (re-mesh). Trying to do blue-noise sampling with high-performance sampling rate on mesh surfaces is challenging. As we already developed the generation algorithm for 2D planar space, thus we use this advantage to further improve upon surface sampling using a new technique called Progressive Sample Projection (**PSP**). This method will use the maximal Poisson-disk samples generated from the previous KDRT method and project the sample from three orthogonal planes in space onto the mesh surfaces. Redundant samples will then be eliminated from the surface. Samples will be properly sampled on the surface with such projection and elimination process, and more iterations can be applied to make the process progressive.

Finally, we consider how the sample quality can be further improved by a novel relaxation-based algorithm. We take a further step in developing a parallel relaxation-based method based on a distance constraint. Most relaxation-based algorithms stay at 20K samples per second performance, which is not acceptable. We developed distance-constraint relaxation (**DCR**) method, which also used a more advanced second-order information named activeness, to help improve the convergence time and sampling rate. We will make further explanations to these methods in following chapters. But here we can preview the relationship of these techniques and locate our methods with our most focused parameter: quality and performance (generation speed), in a big picture. See Figure 1.6 for details. The algorithm in the figure comes from the following work: Naive Dart-throw ([Cook, 1986]), Best Candidates([Mitchell, 1991]), Annulus darts([Bridson, 2007]), CVT(Centroidal Voronoi Tessellation, [Lloyd, 1982]), Wang Tiles([Cohen et al., 2003]), Flat Quad-tree([Ebeida et al., 2012]), PixelPie([Ip et al., 2013]), Weight-Elimination([Yuksel, 2015b]), Phase Group based Elimination ([Bowers et al., 2010]), Priority-based Elimination Elimination([Ying et al., 2013]), FPO(Farthest Point Optimization, [Schlömer et al., 2011]), CCVT(Capacity-Constraint Voronoi Tessellation, [Balzer et al., 2009]), CapCVT(Capacity-Constraint Centroidal Voronoi Tessellation, [Chen et al., 2012]),

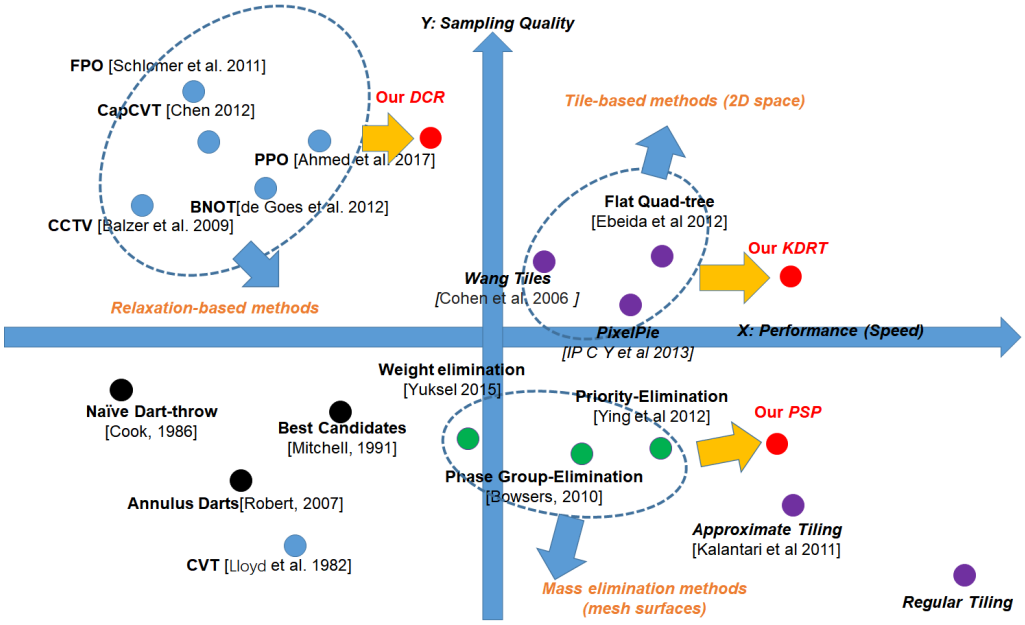


Figure 1.6: We place our techniques discussed in the thesis based on the axis of quality and performance (speed) on the points in the graph. Our focus is the performance, but also try to keep the quality on-par or better than existing previous methods. This figure shows how we placed our methods in the field.

BNOT(Blue Noise through Transport Optimization, [De Goes et al., 2012]), PPO(Push-Pull Optimization, [Ahmed et al., 2016]).

We have discussed above on the problem we try to solve (performance), and the relationship between our three techniques, as well as where we can put the methods in the big picture of other families of methods and how they are related. We further illustrate the relationship of our techniques as in Figure 1.7. The relationship between these techniques is quite straight forward. We developed KDRT for fast generation of 2D planar space sampling. The sample pattern will be further used as the input for the second PSP method for surface sampling. Our DCR method contributes to tuning the results generated from the other techniques for better quality, while it can also be used to generate high-quality blue-noise samples from scratch.

To evaluate the sample set, a research tradition would be to use the sampling strategy's power spectrum as a way to do the analysis, which will be referred to as *Spectral Analysis*. The following section will introduce more backgrounds relating to the evaluation of sampling strategies.

1.4 Analysis of Sample Pattern

1.4.1 Spectral Analysis

To explain this frequency-domain behavior in detail, we have to understand how samples are viewed under the **power spectrum** perspective. It is a very important tool to study and analysis sample patterns from another different point of view. The method is first proposed by [Ulichney, 1988] to utilize the averaging periodograms methods from [Bartlett, 1978] to generate a meaningful representation in the frequency domain to represent the feature of a stochastic sample pattern generated with a specific strategy. Extend the formulation in [Lagae

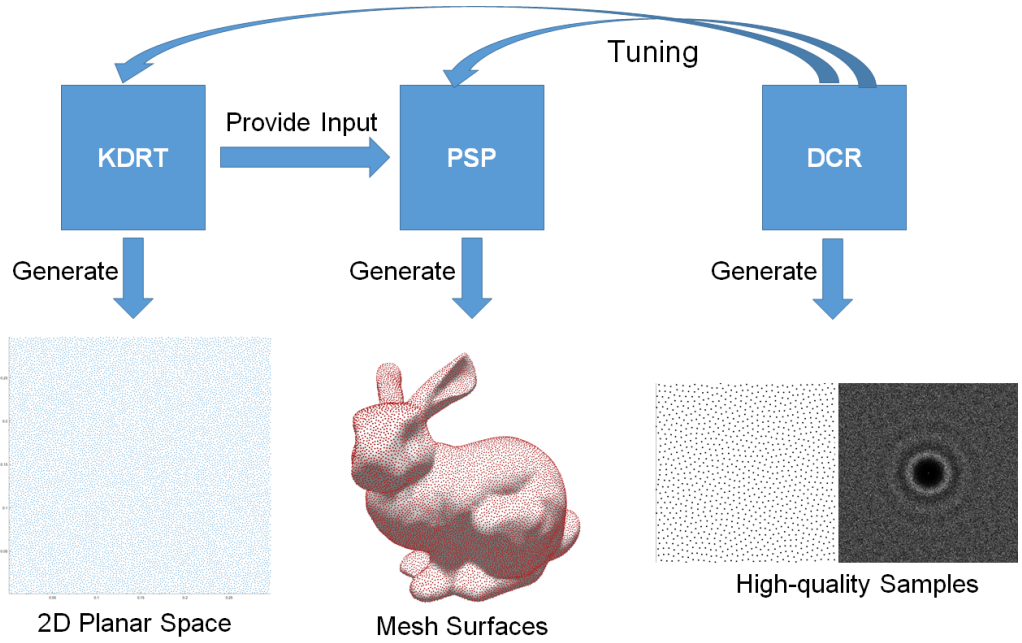


Figure 1.7: The relationship of the three techniques discussed in the thesis is illustrated in the figure. The KDRT can be used for extremely fast 2D planar space blue-noise sample generation with quality insurance of the maximal-coverage property. It will be in charge to provide the input that the PSP method needs to further reduce sampling time. The DCR can be used to tune the first two techniques, while it can also be used to generate high-quality samples from an initial state (as long as the initial states contain samples with differentiated sample positions and not in regular hexagonal lattice state).

and Dutré, 2008], and suppose that the discrete signal sampled in the sampling domain can be represented as the sum of Dirac delta functions:

$$S(\mathbf{x}) = \frac{1}{N}\sum_{k=1}^{N}\delta(\mathbf{x}-\mathbf{x_k}) \tag{1.1}$$

where each $\mathbf{x_k}$ represents a sample in the domain, that the Fourier transform of this function is:

$$f(\mathbf{w}) = \int_D f(\mathbf{x})e^{-2\pi i(\mathbf{w}\cdot\mathbf{x})}d\mathbf{x} \tag{1.2}$$

$$S(\mathbf{x}) = \int_D \frac{1}{N}\sum_{k=1}^{N}\delta(\mathbf{x}-\mathbf{x_k})e^{-2\pi i(\mathbf{w}\cdot\mathbf{x})}d\mathbf{x} = \frac{1}{N}\sum_{k=1}^{N}e^{-2\pi i(\mathbf{w}\cdot\mathbf{x})} \tag{1.3}$$

where f is the Fourier transform, S is the results of transforming the Dirac delta functions introduced above, and $\mathcal{P}$ will be the power spectrum that is commonly used as a measure of noise properties from a frequency domain perspective:

$$\mathcal{P} = \left|\sum_{k=1}^{N}e^{-2\pi i(\mathbf{w}\cdot\mathbf{x})}\right| \tag{1.4}$$

From the spatial domain as well as the power spectrum, we can see some common feature of a blue-noise sample pattern. They often appear to have low energy during low-frequency domain and raised rapidly at the distance of the conflict radius. Far away samples will then be distributed uniformly, which is

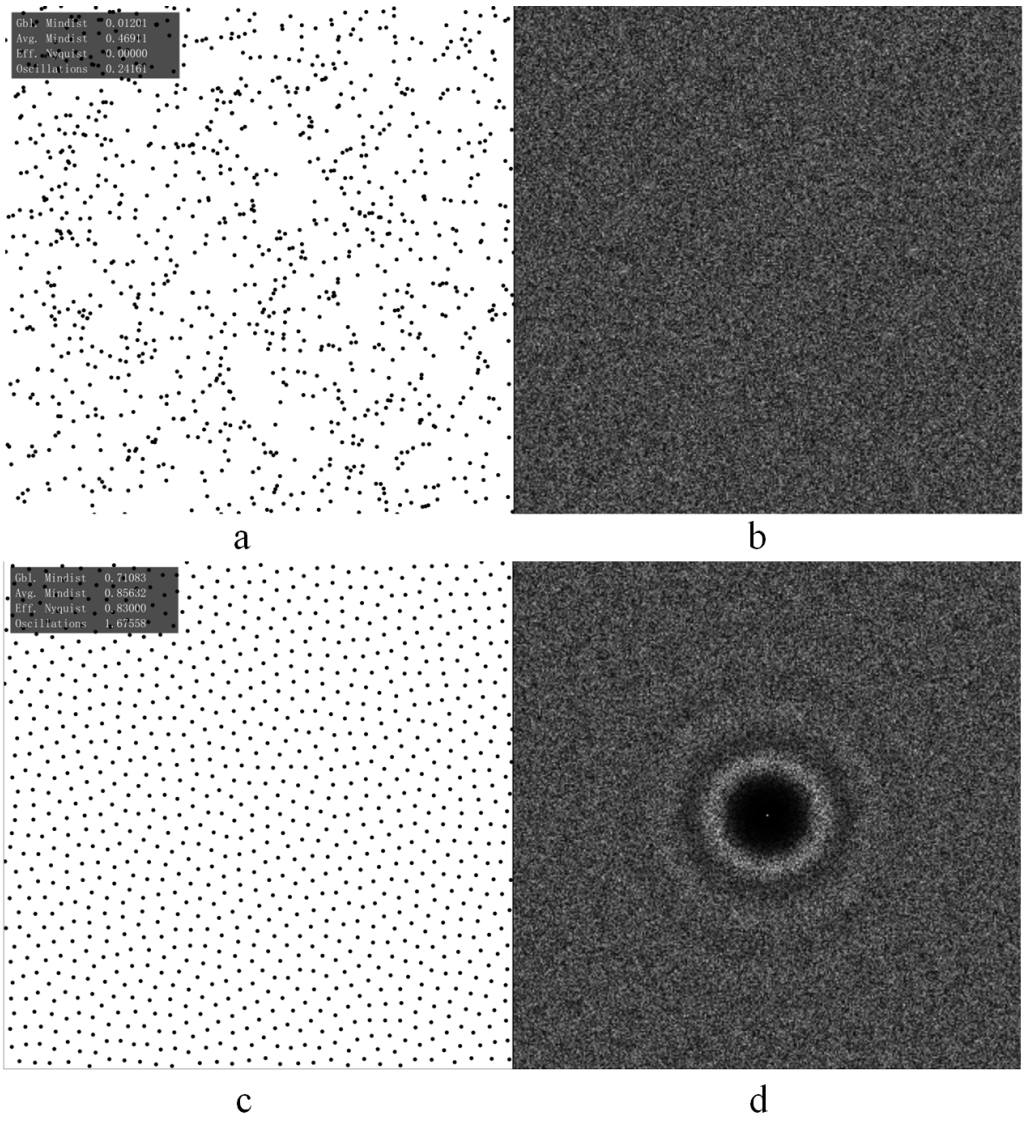


Figure 1.8: (a). 1024 Random samples. (b). Power spectrum of the Fourier transform of the random samples. (c). 1024 samples generated with a blue-noise sampling method CCVT (Capacity-constrained Voronoi Tessellation[Balzer et al., 2009]). (d). Power spectrum of the CCVT sample pattern.

showed by the white noise parts far away from the center. As a comparison, the power spectrum of a uniformly random sample pattern itself is white noise as a whole. Part of the power spectrum in the thesis is generated using the point set analysis tool introduced by [Schlömer and Deussen, 2011]. Figure 1.8 shows the comparison of random sample power spectrum and the blue-noise sample power spectrum.

From previous sections, we know that the Poisson-disk samples also satisfy the blue-noise property. Figure 1.9 shows the power spectrum of the Poisson-disk sampling pattern with 7730 samples.

From Figure 1.9 we also give some simple introduction of the 1D property: Radial means, generated from the 2D power spectrum. The spectral analysis is very important in sample analysis and synthesis. We will constantly be using these tools to reveal features of a sample set under the hood. Figure 1.10 introduced how we get some useful information of radial mean from the Fourier

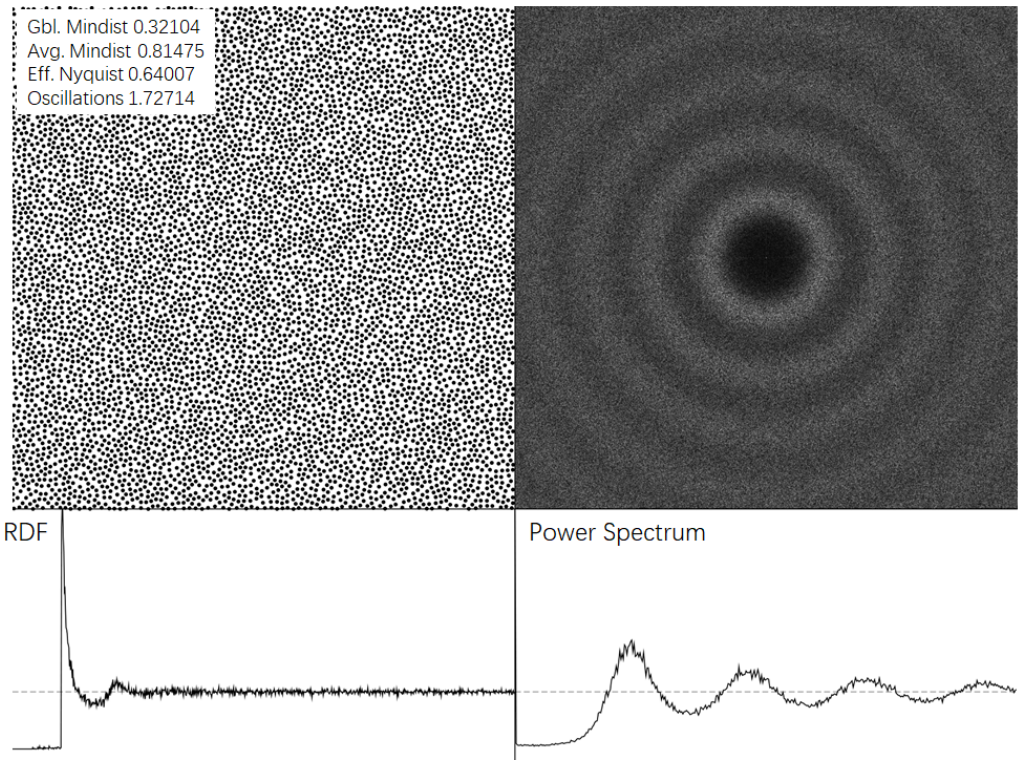


Figure 1.9: The power spectrum analysis of a Maximal Poisson-disk sample set. All samples generated by a Maximal Poisson-disk sample algorithm from [Ebeida et al., 2012]. Top left is the samples in the spatial domain, top right is the power spectrum 2D image. Bottom left plot is the radial distribution function, which will be introduced in 4.1. The bottom right plot is the radial average of the power spectrum. For isotropic sample set, it is convenient to transform the 2D power spectrum to a 1D radial average for quick observation and better understanding.

transform power spectrum.

1.4.2 Radius Statistics

The spectral analysis is more popular in applications such as rendering, where randomly generated samples are used to control an overall feature such as aliasing, which can be revealed through a frequency domain analysis. Whereas some other statistics of the synthesized sample set can find more usage in texture synthesis, object placement, remeshing, and stippling. In such applications, the relationship of each sample pair becomes more important than a global feature.

From [Steinhaus, 1999] we know that densest packing of disks in the plane is the hexagonal lattice. The packing density is defined as:

$$\phi = \frac{\pi}{2\sqrt{3}} \tag{1.5}$$

From which we know that if there are N samples arranged in a hexagonal way, the maximum disk area will indicate the maximum radius to keep each other from conflicting:

$$r_{max} = \sqrt{\frac{1}{2\sqrt{3}N}} \tag{1.6}$$

Bear this in mind that in chapter 4 the method of determining the r_{max} will be used to set the most important parameter in the relaxation-based method.

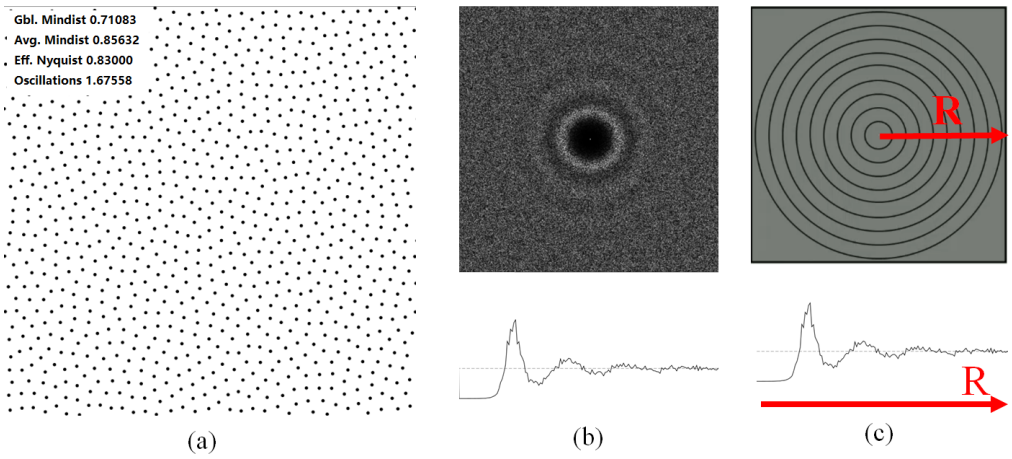


Figure 1.10: How radial means is calculated: here we have (a). the original point clouds (samples), (b). the power spectrum on the top, the radial mean at the bottom and (c). The radial mean is calculated by taking the average of the concentric circles from the middle-low frequency to the edge high-frequency domain. The bottom radial mean function is the calculated result.

1.5 Introduction to Related Applications

High-quality blue-noise samples can be used in various applications, especially in rendering, geometry processing, texture synthesizing, object placements, image stippling, etc. In this section, we will give some brief introductions to such applications.

1.5.1 Mesh Reconstruction

Our methods can be used in remeshing for mesh densification or simplification, as well as other purposes such as constraining triangle properties on the geometry. An evenly distributed and random mesh surface is preferred in most graphics applications. See Figure 1.11 for a comparison of triangulation using blue-noise samples and uniform random samples.

Figure 1.12 is an example of using our methods (the progressive sample projection method) to generate Poisson-disk point clouds combining with the ball-pivoting algorithm ([Bernardini et al., 1999]) to do the mesh reconstruction. There are some quality measurements for a mesh reconstruction results from samples. For example, the percentage of obtuse angles and acute angles (less than 30 degrees), the variance of triangles' statistics (area, length of edges), and more. It shows that the remeshing through a blue-noise sampling strategy can provide much more high-quality triangulation comparing with using other sampling strategies (see supplement materials of [Ahmed et al., 2016] for more examples of such comparison).

1.5.2 Sampling in Rendering

Blue-noise samples can be used in rendering [Pharr et al., 2016]. High-quality off-line rendering needs high-performance sample generators for various Monte-Carlo sampling happened during the whole process. For example, depth-of-field, soft shadowing, motion blur, etc. [Cook, 1986] provided the first proposal of such usage of Poisson-disk samples for anti-aliasing. Figure 1.13 showed how the noise

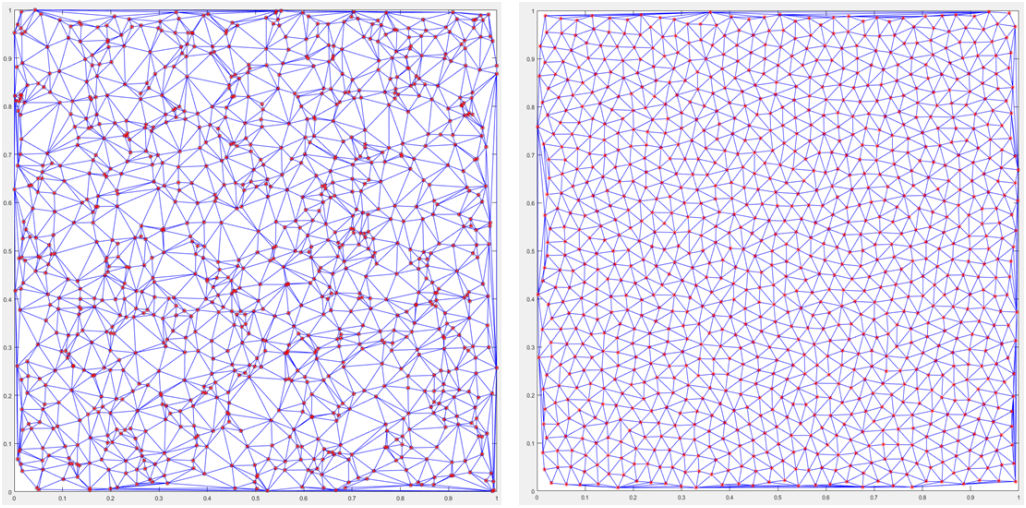


Figure 1.11: Delaunay Triangulations on the random point cloud (left) and blue-noise sample set(right). Most geometric applications that will operate on a triangle mesh favor a processing unit of the triangle to be with similar sizes and angles, while still appear to be random. The triangulation on the left side has triangles with varying sizes and angles, which may cause a workload balance problem on mass parallel hardware that often needs to process triangle meshes in parallel, such as GPUs. Blue-noise sampling provided a way to regulate triangles for such purposes.

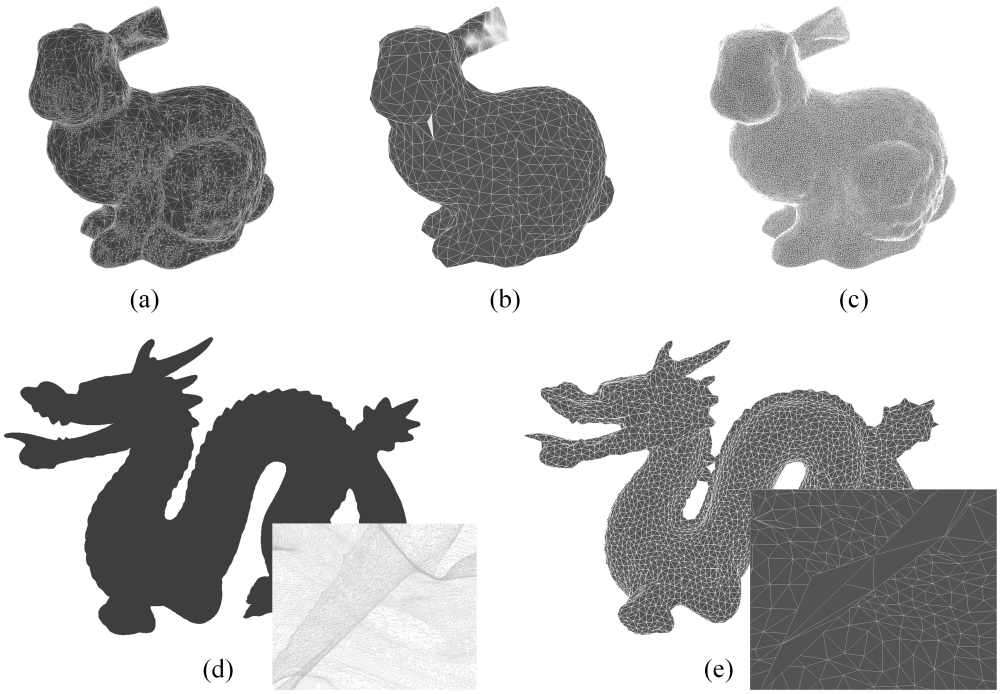


Figure 1.12: Application: remeshing. Here we show how remeshing working on the Bunny model in (a) to regulate the triangle properties and do refinements of triangles in (b) and (c). The 5K triangles are refined to 10 times more triangles (100K) in (c). And in (d) we can see how the 870K triangle dragon model is simplified to 5K with Poisson-disk sampling and reconstruction.

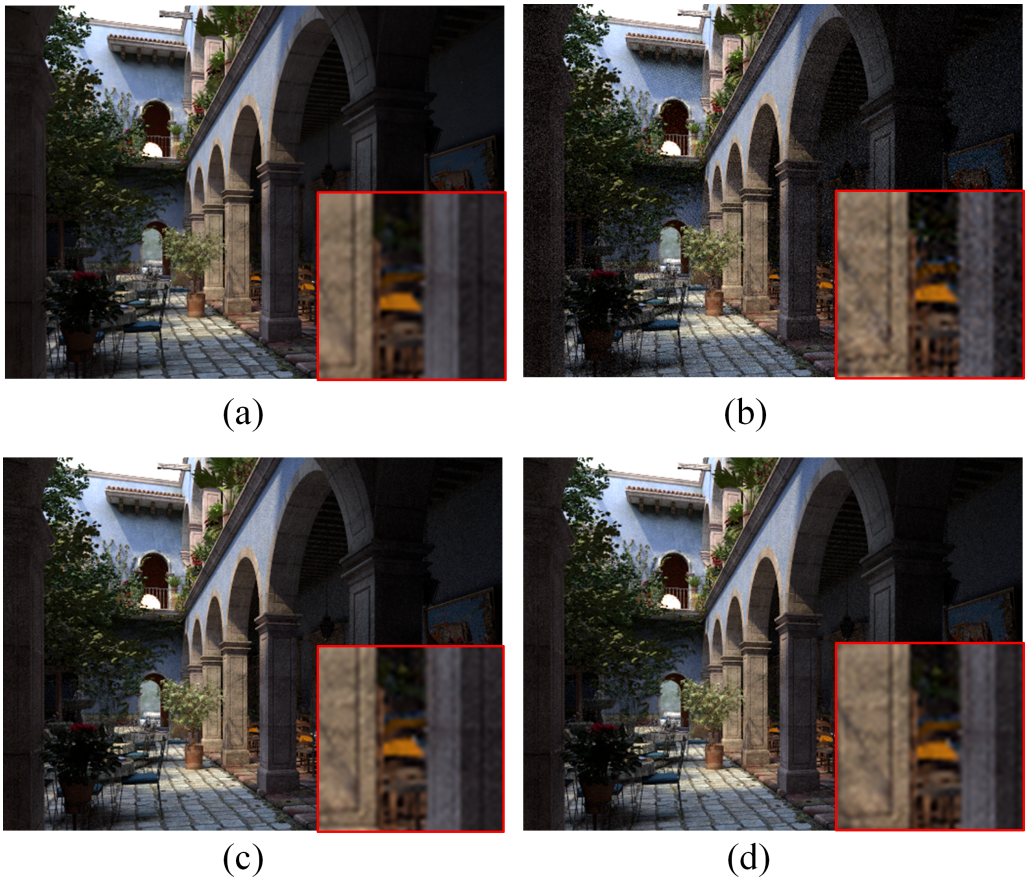


Figure 1.13: The San Miguel scene rendered with different samplers in charge of various sample generation in the process of Monte-Carlo based rendering. (a). Referenced image. (b). Uniform sampler. (c). Blue-noise sampler by BNOT ([De Goes et al., 2012]. and (d). Blue-noise sampler with our KDRT method generated 100x faster than BNOT. Image rendered with PBRT [Pharr et al., 2016]. Model courtesy of Guillermo M. Leal Llaguno of Evolucien Visual, downloaded from Morgan McGuire, Computer Graphics Archive, July 2017 (https://casual-effects.com/data)

property could be different for the scene rendered with physically based renderer based on Monte Carlo integration.

1.5.3 Image Stippling

Image stippling is another application where blue-noise samples are widely applied and generally preferred. Image stippling is a process to create an image pattern shading by small dots, with the density of dots represents the color of each part of the image. In our thesis, we can make image stippling extremely fast due to our efficient parallelism in progressive sample projection methods. To generate a stippling image can prove that the algorithm can generate samples with adaptive radius criterion. This method treats image stippling as a special 2D case from a 3D mesh sampling perspective and can generate high-quality stippling with an effective sampling rate of 200K samples per second. See Figure 1.14 for an example of stippling the famous Lena picture with a massive amount of samples.

Figure 1.14: High performance image stippling of the famous Lena image containing over 2 million samples within 10 seconds, which is by far one of the fastest high-quality image stippling generation method.

1.5.4 Point Cloud-Based Learning

Another important group of application is point cloud-based learning. Recently the effectiveness of point-based learning has been proved by some pioneering work such as [Qi et al., 2017a] and [Qi et al., 2017b]. The basic idea is that the traditional convolutional neural network can be modified so that it can directly consume point cloud data instead of using rasterized structural data, thus greatly simplifies many geometric based learning techniques and reduces the size of the network by orders of magnitudes. One important step in such an application is to make hierarchical Poisson-disk sampling on mesh surfaces, such as the hierarchical structure described in [Hermosilla et al., 2018], [Li et al., 2018a] and [Li et al., 2018b]. Meanwhile, they often require that the sampling rate should be very high in the learning process to find the proper sample for each hierarchical layer. Our progressive projection sampling method is best suited for such scenarios as it can provide enough sampling rate during the whole learning process. See Figure 1.15 for an illustration of the PointNet++ ([Qi et al., 2017b]).

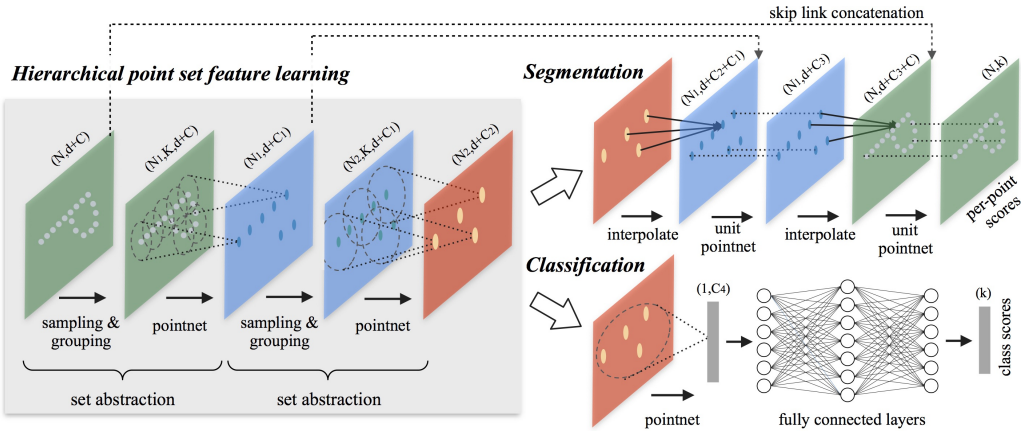


Figure 1.15: PointNet++ architecture from [Qi et al., 2017b]. This network will use the max-pooling based PointNet(stated in [Qi et al., 2017a]) as a building block to build a hierarchical learning network that directly consume point cloud data. The data will be learned hierarchically to form the global and local feature vectors, and connect them to segmentation or classification network with a U-net ([Ronneberger et al., 2015]) style skip link concatenation. This architecture is time and space efficient in capturing geometry features, and can achieve high accuracy in geometry-based learning applications such as geometry model's segmentation and classification.

1.6 Related Work

In this section, some generally related work will be listed, including how the blue-noise sampling can be formulated as an issue in computer graphics, some generally used generation methods for blue-noise samples with performance issue bear in mind, and how evaluation system of stochastic sampling method evolved through time. More specific works related to each of our proposed topics will be introduced later in the corresponding chapter.

Dart-throw and Variation

A great number of efforts have been placed on how to do stochastic sampling fast since the 80s of the last century. Some important early work including [Dippé and Wold, 1985] and [Cook, 1986] started the exploration of stochastic sampling patterns in anti-aliasing and various other applications in computer graphics. The method dart throwing is also proposed at the time: With a predefined disk radius r, point samples are continuously inserted into the sample domain. They put more efforts into the correctness of stochastic sampling patterns. The point will be accepted if not conflicting previously accepted points. One obvious disadvantage of this method is computation complexity. As the termination condition cannot be determined with minor efforts, the acceptance rate of samples will drop significantly as the sampling domain filled by more and more samples, which makes this algorithm extremely slow to converge, the convergence rate of dart-throw is often not acceptable. Generate point set with maximal coverage property is even nearly impossible using only dart throwing. A thorough survey of some earlier work in this field is presented by Lagae and Dutré [2008].

More recent works focusing on dart-throwing methods tend to use some specific techniques to reduce sampling area after each dart accepted, to increase the acceptance rate of the next dart. Gamito and Maddock [2009] try to keep all

disk-free area in a quad-tree data structure, throw darts against the tree, then update the data structure after each dart accepted. Ebeida et al. [2011] uses a similar grid data structure, and keeps tracking a convex polygonal approximating the void inside the grid. Ebeida et al. [2012] keeps a record of an implicit flat quad-tree of active grids, then continuously splits and tests sub-grid against each dart. The quadtree is kept as flat as possible, so memory consumption is significantly reduced. Yan and Wonka [2013] provides a deep study on concepts of general gap analysis in Poisson-disk sample sets with varying radii. This category of methods can be orthogonal to ours. Readers can refer to a survey like Yan et al. [2015] for more information. Reinert et al. [2016] proposed a method to retain blue-noise characteristics when projecting samples to lower-dimensional space, that can be considered a reversed problem of ours, which is to generate blue-noise feature from lower-dimensional space to manifolds in high dimensional space.

Relaxation-based Methods

Dart-throw generates the required samples from nowhere, while another group of algorithms tried to morph existing samples by re-positioning their locations to force them to meet Poisson-disk criteria. These techniques are relaxation-based methods. One significant work is Balzer et al. [2009] which solved the over-regularized sample pattern problem that often emerged in early works. A more detailed review of this family or methods will be given in chapter 4.

Tile-based methods

Another group of work sought to generate Poisson-disk samples by tiling existing Poisson-disk samples on the whole sampling domain. Important works including Cohen et al. [2003], Lagae and Dutré [2005], Kopf et al. [2006]. Tiles were designed and placed in a clever way such that samples in different tiles would not conflicting with each other. This method can generate lower-quality 2D Poisson-disk samples extremely fast. In our progressive projection method, a 2D sample pattern will be used for sample projection, and fortunately, the quality of the 2D pattern will not affect our results that much, so we choose to use a tiling based method drawing inspiration from Kalantari and Sen [2012] and Wang and Suda [2017] for simplicity and performance issues.

Others

The radial distribution function or a target power spectrum's differential can be used to guide synthesis of Blue-noise samples, such techniques are discussed in Zhou et al. [2012], Heck et al. [2013] and Öztireli and Gross [2012]. Meanwhile, there are also techniques that utilized high dimensional darts for blue noise sampling that is further discussed in Tzeng et al. [2012],Tzeng et al. [2012] and Sun et al. [2013]. There is also a family of methods that do an advancing front style dart-throw to generate blue-noise samples. The most recent is Mitchell et al. [2018]. A simple advancing front method is very popular due to its implementation is really easy, such as Bridson [2007], which used a point-annulus to limit the space of valid dart throws. Mitchell et al. [2018] extended this idea to a more general high dimensional spoke-darts method, that is capable to generate Poisson-disk samples in high dimension space.

Point-cloud based learning is an important issue in sampler development as well. Leimkühler et al. [2018] provide new thoughts on how to utilize point cloud-

based machine learning techniques to generate sampler filters automatically that will fit a general power spectrum.

Chapter 2

High-Performance Randomized Tiling Method

In this chapter, a high-performance tile-based Poisson-disk sampling method will be proposed. This chapter is organized as follows: First, we will give a brief introduction of the randomized tiling-based on a KD-tree based divide-and-conquer algorithm. The background including some previous high-performance sampling algorithm will be introduced. Then the algorithm will be explained in detail, and finally, we'll provide applications that utilized the algorithm and show results comparison and analysis.

Some contents of this chapter came from the author's already published paper in [Wang and Suda, 2017], DOI: 10.1145/3105762.3105778. Here is to claim that some reuse of such contents here in the author's thesis is allowed by the ACMs copyright license.

2.1 Introduction

We introduced that blue-noise sampling is of crucial importance in various application fields from previous chapters. It is generally accepted that Poisson-disk sampling provides really good properties and appear to have a blue-noise spectrum. One important aspect of a sampling algorithm is if it can provide enough sampling rate (the performance of generating samples) for interactive or even real-time applications. Most Poisson-disk sampling algorithms failed to provide enough sampling rate, which results in that even it can generate high-quality blue-noise samples, it can be impractical to use in some high demanding applications.

2.1.1 Poisson-disk Sampling with a Divide-and-conquer Method: Introduction

Blue-noise samples are proved to generate less aliasing in the synthesized image and prevent artifacts (discussed in [Cook, 1986], [Jiang et al., 2015], etc.), and Poisson-disk sampling is a good example of how simple regulations of samples can generate good blue-noise features.

A recap of Poisson-disk sampling will be: The Poisson-disk sampling pattern is a group of samples with no two of them within a specified radius. Furthermore, we would like to generate Poisson-disk sampling with ensured higher quality. The disk-free criterion can only guarantee no samples are near each other, but it's not able to resolve gaps among samples. while a Maximal Poisson-disk Sampling means that no more points can be added to the group of samples, which means that there are no *gaps* that are not covered by a sample disk. To formulate

it, we follow the definition in [Gamito and Maddock, 2009] and the extended version in [Ip et al., 2013], that if we denote sample set as X in domain D, with each sample as i, according to definition that maximal Poisson-disk samples have following properties:

$$\forall i \in X, \forall S \subseteq D : P(i \in S) = \int_S di \tag{2.1}$$

$$\forall i, j \in X, i \neq j : \|i - j\| \geq r \tag{2.2}$$

$$G = \{j \in D | \forall i \in X, \|i - j\| \geq r\} = \emptyset \tag{2.3}$$

Equation 3.1 means that the possibilities of sample placements on the sampling domain are uniform. This is also referred to as the *bias-free* property. Equation 3.2 is the conflict-free requirement, that no pairs of points can be closer than a minimum distance. Equation 2.3 is the maximal coverage requirement, that no other samples can be inserted in the sample set without breaking the minimum distant requirement. To meet the requirements of a wide range of applications, our goal is to generate a large group of good quality maximal Poisson-disk sample set in interactive speed.

To achieve this goal, we proposed a novel tiling based divide-and-conquer algorithm that can efficiently generate maximal Poisson-disk samples on a 2D planar domain with an extremely fast sampling rate. The outline of the algorithm is as follows: Consider N samples is required on a sampling domain, first prepare a 2D sample set sampling on a planar domain with n samples, $n << N$, which will be called the *pattern*. We then divide the target sample domain randomly with a KD-tree pattern, so that the target domain consists of many small leaf rectangle domain. Then each leaf rectangle will clip its samples from the pattern from a random starting position. And finally, all samples will be merged to eliminate any conflicts. A brief introduction will be presented for each step, then we'll go into details in the following sections. Figure 2.1 illustrated the whole process.

2.1.2 Divide: Randomized Tiling

The divided part is quite simple, it just divides the target sampling domain to randomized clips. One example of division would be doing it in regular to divide subspace to rectangles with the same dimensional parameters. This is being used by [Kalantari and Sen, 2012], but their methods fail to generate conflict-free sample patterns, while even being rotated, the regularly clipped underlying rectangle space appear to have regularity that is not acceptable for some high demanding applications, especially those demanding controlled aliasing.

Our idea will be using a randomized division, be specific, use a randomly generated KD-tree for the division. There will be a minimum dimension bounded so it will not generate a final clip with unacceptable size (too large or too small). It will also prove itself to be fast with the overhead being negligible. Figure 2.2 shows how the underlying 2D space is divided by a randomized KD-tree structure. In this way, we can have the benefit of high-performance sampling rate, while keeping the final results random. Our results showed that the final biases introduced is limited, and from the power spectrum perspective, randomized tiling also keeps the final sample pattern random and avoid a periodic appearance if using a regular grid such as stated in [Kalantari and Sen, 2012].

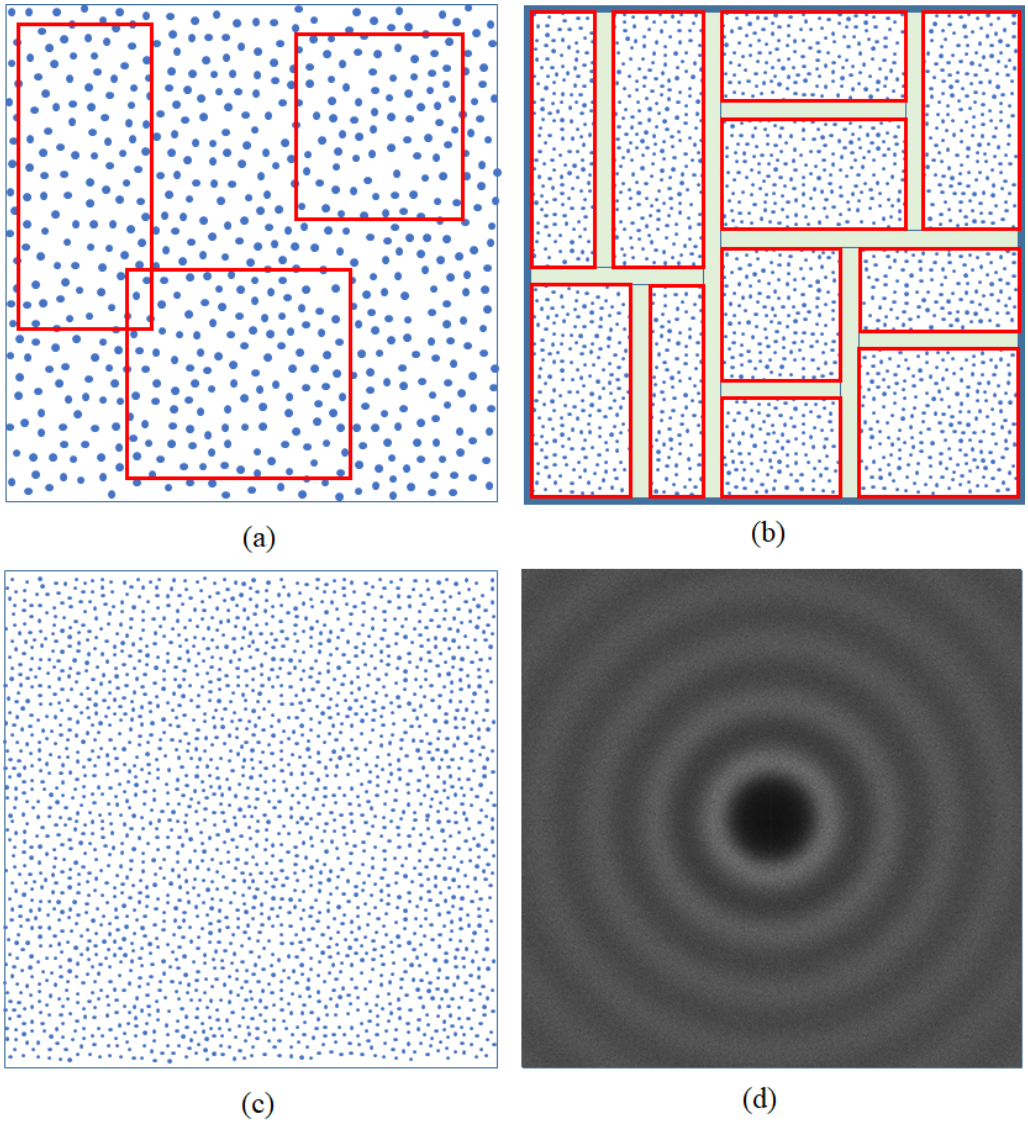


Figure 2.1: (a) The original maximal Poisson-disk sample pattern with n points generated by a Maximal Poisson-disk Sampling method. (b) Example of randomized tiling. Tile randomly clipped tiles from the previous pattern, to form the target sample set with $4n$ or more points. The size and position of each clipping are determined by a KD-tree. (c)After the merging, Eliminate conflict points in the margin area of each random tile and insert new points in the gap caused by the elimination to ensure maximal coverage property. (d) The power spectrum of Poisson disk distributions generated with our algorithm.

2.1.3 Conquer: Merging Tiles

To ensure a maximal coverage of samples, we have to tile redundant samples along the edges of each leaf node. But these samples along the edges will cause the violation of conflict-free criterion. Thus a compaction process that eliminates all conflicts among samples will be necessary. The details will be explained further in the next section. We use the **elimination and insertion** method to detect invalid samples, eliminate invalid samples, and if there is a gap left, fill the gap with a generated sample to ensure the maximal criterion. Figure 2.3 showed the samples generated only by randomized tiling, and also the conquered results

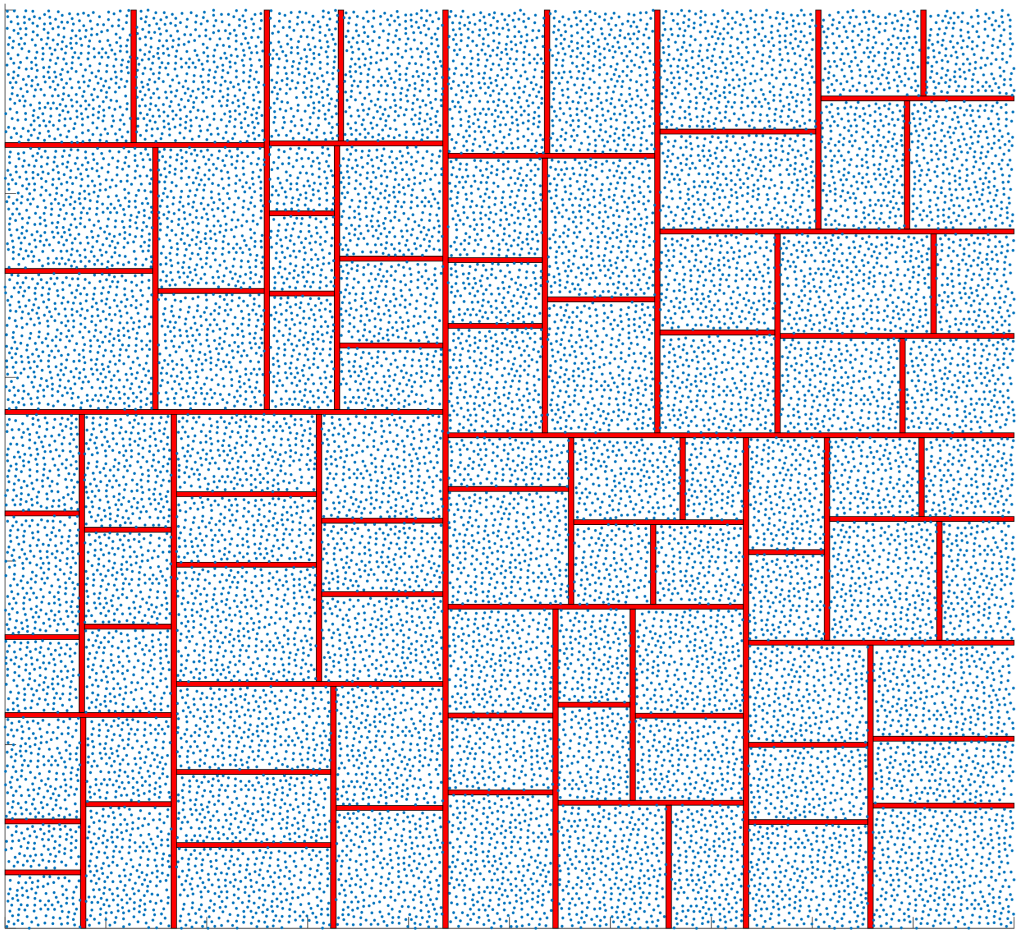


Figure 2.2: This shows how the underlying 2D space is divided by a randomized KD-tree. Each split is represented by the red bar. Each leaf node will then be tiled with a clipped pattern from the maximal Poisson-disk pattern.

using our elimination and insertion algorithm.

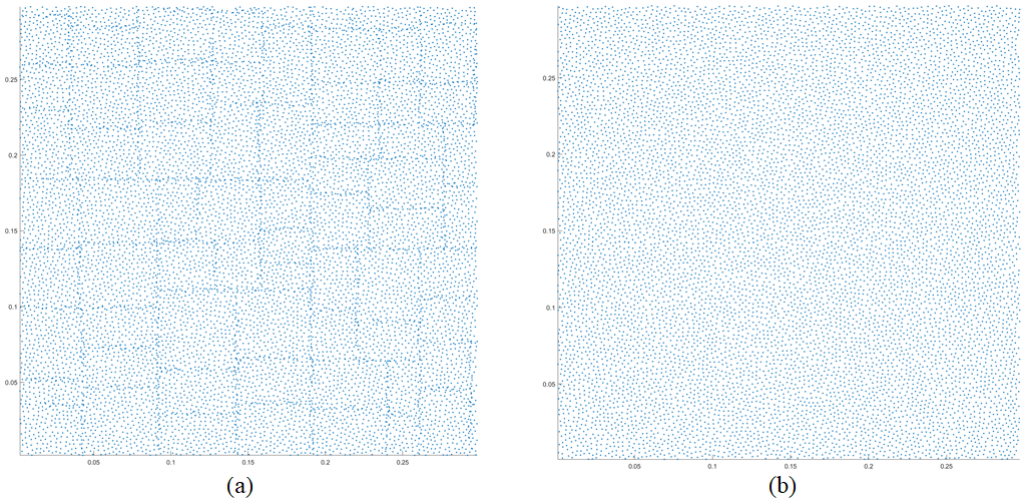


Figure 2.3: (a). To keep the final results maximal, it's necessary to do redundant sampling in each leaf node, thus redundant samples break the conflict-free law. (b). After the merging algorithm, all conflict samples are removed.

2.2 Related Work

We know that there are a lot of studies on how to achieve some or all of these maximal Poisson-disk sampling properties. Most of the previous methods focused on using dart-throwing based methods to insert as many points.

Tile-based methods can generate Poisson-disk samples much faster, but with serious sacrifice in sample quality or trade-off in bias property. The basic idea is to generate small sample sets and tile them in the sampling domain to form a large sample set. Some significant work including [Cohen et al., 2003], [Lagae and Dutré, 2005], [Kopf et al., 2006] and [Wachtel et al., 2014a]. But the quality loss of these tile-based methods are sometimes not acceptable, and few tile-based methods focused on the maximal coverage property of the sample domain. KDRT can also be classified as a tile-based method, but the error of KDRT is much less as more degrees of freedom in randomization are exploited in KDRT. Also few of tile-based method insists on a maximal coverage property. Comparing with [Kalantari and Sen, 2012] who also used a tile-based method fetching from a small pattern, our method is different in several ways. First, we randomized the tiling scheme on the sampling domain with a KD-tree based manner, which ensures a more random result. Second, our method met the conflict-free requirement that most applications will need, and third, KDRT guaranteed the final result sample set a maximal coverage property.

Poisson-disk sample sets can also be obtained by repositioning samples. Reposition base methods are generally referred to as relaxation-based methods. Samples generated by many previous Lloyd relaxation-based methods tend to be too regular for many applications until capacity-constrained relaxation methods are generally applied such as in [Balzer et al., 2009] and [Xu et al., 2011]. A more recent method using optimal transport is discussed in [De Goes et al., 2012], which will also be compared to our method in this chapter. This category of methods can also be orthogonal to this chapter. A more detailed review is in chapter 4.

Most recent solutions for Poisson-disk sampling would utilize a parallel computation pattern to boost its performance. [Wei, 2008] started the randomized phase group based parallelization pattern, which is enhanced to deal with 2D

manifold sampling in following works such as [Bowers et al., 2010] and [Ying et al., 2013]. Our method also used some of their methods for GPU-friendly execution. Other new and inspiration methods including [Ebeida et al., 2014] which is effective in high dimensional space, and [Yuksel, 2015a] with a simple and elegant elimination method to generate neat Poisson disk samples on 2D manifolds.

Almost all previous literature discuss the quality of Poisson-disk samples or other sampling patterns in a frequency domain analysis framework. See [Lagae and Dutré, 2008] for a more detailed discussion. More theoretical and practical details are provided by [Durand, 2011], [Subr and Kautz, 2013] and [Pilleboue et al., 2015] on analysis of sampling quality. Our method used a similar analysis tool as discussed in [Schlmer and Deussen, 2011]

Our method is also partly inspired by [Kalantari and Sen, 2012]. They try to tile the sampling domain with a rotated square pattern. Although they reached very fast sampling speed, the cost of low sampling quality and failed conflict-free and maximal coverage requirements sometimes are not unacceptable. Our method instead provides a more accurate solution that met the requirements of conflict-free and maximal coverage, with bounded error in biases.

To provide an overview and comparison, this chapter has the following contribution:

1. We present a fast and robust tile-based maximal Poisson-disk sampling method (KDRT) by randomly dividing sample domain into tiles, tiling each tile with a clipping from the pre-generated pattern, and gluing all the tiles together to generate a large sample set.

2. We show that this method can be a proved framework to do divide-and-conquer parallelization of almost any other Poisson-disk sampling methods.

2.3 Algorithm Details

In this section, we will be going through details of the KDRT algorithm. It is a divide-and-conquer tiling based blue-noise sampling algorithm. The randomized tiling algorithm is a two-step process:

1. Clipping tiles from a small pre-generated Maximal Poisson-disk(MPS) pattern, and tile them on the sampling domain to form the large sample set.

2. Eliminate conflict points in the margin area of each clipping and properly insert points to cover gaps caused by elimination.

There are some notes and abbreviations that will be constantly referred, including the following:

- *MPS*: Maximal Poisson-disk Sampling.
- n: Size of the output MPS set.
- *Pattern*: A pre-generated MPS set with size k, $k << n$.
- *Clipping*: The operation of fetching a subset of samples within a specified rectangle (2-D) of the pattern.
- $\mathcal{S}$: The set of Poisson-disk samples.

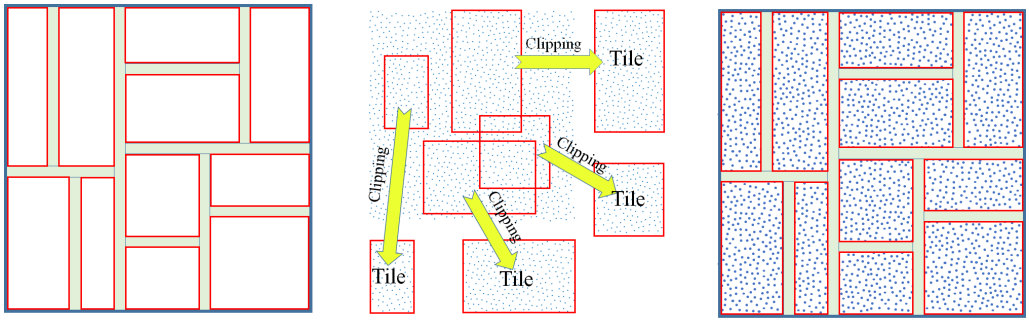


Figure 2.4: This is the outline of the process *clipping and tiling*. (a). Divide the target domain to randomized rectangles by KD-tree spliting. (b). Fetch all clips of each rectangle from the MPS pattern. (c). Tile the samples to the rectangles.

- $\mathcal{C}$: The set of clipping and tiling result (containing conflict points).
- $\mathcal{D}$: The whole sampling domain.
- $\mathcal{P}$: The whole pattern domain.
- r: Poisson-disk radius (minimum distance for samples).
- *disk*: The circle with a sample as the center, and r as the radius.
- *threshold*: The minimum edge length of the leaf node, as a pre-defined parameter.
- *ratio*: The number of samples of the result MPS sample set divided by the number of samples in the pattern set, roughly means how many times more samples a user want to produce from the pattern set.

The clipping and tiling process will divide the target sampling domain to random rectangles by KD-tree division, then clip each rectangle sampling domain from the original MPS pattern, and finally tile the clip to the corresponding rectangle in the target domain.

2.3.1 Divide: Clipping

Clipping means fetching a subset of samples within a specified bounding rectangle. The first step of this algorithm would be dividing $\mathcal{D}$ into randomized rectangles which will be later used as a clipping unit, as illustrated by Figure 2.4. We call these rectangles building blocks. One convenient way to do this is just to fetch each leaf node of the KD-tree that subdivide domain $\mathcal{D}$.

KD-tree parameters

Notices that out KD-tree is generated from top to bottom, as shown in Algorithm 1. The function GetBuildingBlocks() will be in charge of generating an array of bounding boxes as building blocks: These are actually the randomized discretized target domain. The domain is being divided into such rectangles to be ready for tiling. Note that each of these building blocks contains the maximal and minimum position for each dimension, in 2-D they are $xmin, xmax, ymin, ymax$. As in the randomized tiling step, we will be choosing the dividing plane for each subdividing dimension randomly, and also within a range that can satisfy a minimum

Algorithm 1: Generate leaf rectangles as building blocks

Input : The bounding box of $\mathcal{D}$: b, current splitting axis: $axis$, threshold: Len

Output: Array of bounding box: BBoxes

```
1  Function GetBuildingBlocks (b, axis);
2  if DimensionCheck(b) == ALL or DimensionCheck(b) == axis then
3  |  BoundingBox leftBox ← RandomSplitLeft(b, axis);
4  |  BoundingBox rightBox ← RandomSplitRight(b, axis);
5  |  GetBuildingBlocks(leftBox, nextAxis);
6  |  GetBuildingBlocks(rightBox, nextAxis);
7  else if DimensionCheck(b) == nextAxis then
8  |  GetBuildingBlocks(b, nextAxis)
9  else
10 |  BBoxes.add(b);
11 end
12 Function DimensionCheck ( b, axis, Len);
13 if b.xmax − b.xmin > Len and b.ymax − b.ymin > Len then
14 |  return ALL;
15 else if b.xmax − b.xmin > Len then
16 |  return xaxis;
17 else if b.ymax − b.ymin > Len then
18 |  return yaxis;
19 else
20 |  return leaf;
21 end
```

dimension length parameter, so that each dividing in this procedure will ensure all building blocks' edge length be larger than a threshold. To put it another way, the position for each building blocks is better to be randomized, while the shape of the rectangle is better to have bound on each side, as a particularly long rectangle will suffer from the workload problem during the future conquering step. The function DimensionCheck() will be in charge of this functionality. It is going to check if the current bounding box is ready for subdividing on the specific axis. If a specific axis is not suitable for further subdivide, the subdivision should only happen on other axes instead. For the choice of parameters, see the discussion section of the algorithm. Figure 2.5 demonstrated how the R_{leaf} affect the leaf node dimension size.

When all building blocks are generated, the MPS pattern can be generated. We can generate an MPS pattern slightly larger than the largest leaf node in the KD-tree, or to avoid periodicity and increase randomness. This pattern can be generated by any other methods that met the bias-free, conflict-free and maximal coverage requirement. Here we use algorithms in [Ebeida et al., 2012] to generate the pattern.

Maximal Coverage Sub-domain

Another important issue about clipping is that we always have to clip a redundant edge other than the bounding box itself. The reason is that when samples are clipped from the MPS pattern, actually samples inside the clipping bounding box will not satisfy maximal coverage requirements anymore. It is because that some edge samples that will previously cover a gap in the bounding box will

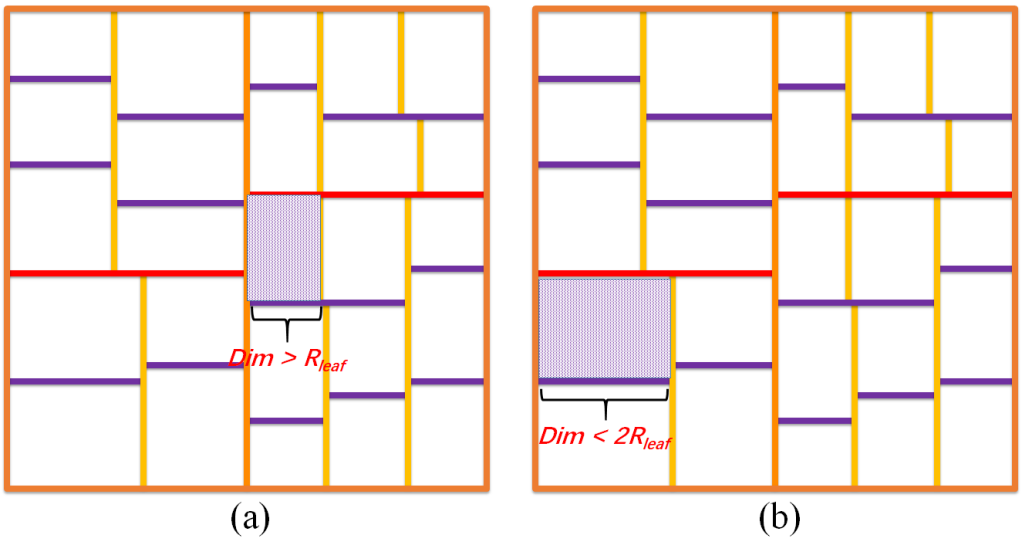


Figure 2.5: The KD-tree based subdivision will have a parameter of the largest dimension to stop further subdivision, which will be a key parameter. For example if we set the minimum subdivision radius to 10r, then with the painted square in (a), that the dimension will be larger than the R_{leaf} parameter, which is controlled by the random choice of subdivision position: it will avoid subdivision near the edge of the node. And in (b) the dimension of the rectangle is smaller than $2R_{leaf}$, thus will not be further subdivide

not be clipped at all, leave a gap uncovered inside the box. Thus we have to include all samples that participate in covering the bounding box, to be clipped in the clipping. In this way, the final target domain can be guaranteed to be with maximal coverage feature.

We define a *shrink domain* a domain that's been reduced by $2r$ in each dimension, and an *enlarged domain* a domain that's been extended by $2r$ in each dimension to make a more detailed clarification of how the enlarged domain being valid to help the final maximal criterion.

Refer to Figure 2.6 for a simple visualization. We know from this that for all the building blocks acquired from Algorithm 1, we have to clip an enlarged block in the pattern to ensure maximal coverage of the sub-domain.

Theorem 1 *(maximal coverage sub-domain): Let $\hat{\mathcal{D}}$ be the shrinked domain of $\mathcal{P}$. $\forall \mathcal{S} \subset \hat{\mathcal{D}}$, Let $\mathcal{E}$ be the enlarged domain of $\mathcal{S}$, the disks clipped by $\mathcal{E}$ covers $\mathcal{S}$.*

To prove it by contradiction, assume that a point $i \in \mathcal{S}$ not covered by disks clipped by $\mathcal{E}$. By definition of enlarged domain, we know that $\forall p \in \mathcal{E}^{\mathcal{C}}$, the distance between i and p is larger than r, thus no disks in $\mathcal{E}^{\mathcal{C}}$ can cover i. And with the assumption we know that no disks in $\mathcal{E}$ can cover i, which means that i is in a gap. But as $i \in \mathcal{S} \subset \mathcal{P}$, and $\mathcal{P}$ is a domain with maximal coverage property, contradiction exist.

Samples will be generated by the clipping algorithm. Now we need to know how the tiling algorithm actually works.

2.3.2 Divide: Tiling

With the discussions above, the pseudocode to generate tiles will be straightforward. Note that as $k << n$, so when clips are clipped from the pattern, they

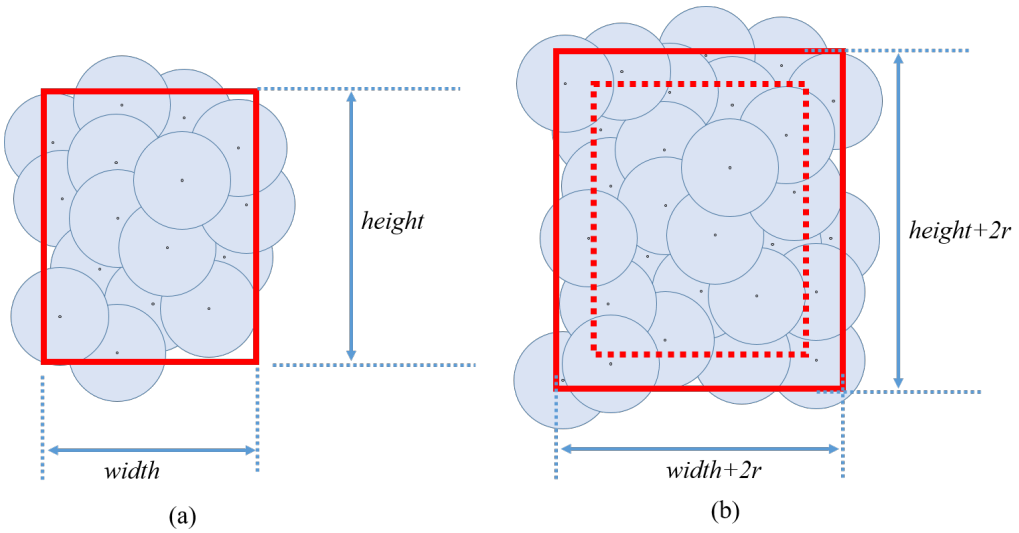


Figure 2.6: (a) Notice that clipping tiles itself may not satisfy maximal coverage requirements. As samples that previously covered the gaps are not clipped in this region. (b) If we enlarge the clipping domain by $2r$ in each dimension, then include all samples in outer margin domain, it is guaranteed that the original clipping is ensured to be maximal coverage from the simple theorem 1.

have to be scaled by a pre-defined parameter *ratio* to fit in the building block. We also have to translate each clipping to the correct corresponding position of each building blocks in the final sampling domain. This procedure is described in Algorithm 2.

Algorithm 2: Psudocode of Generating Tiles

Input : BBoxes $BoxArray$, Pattern $\mathcal{P}$, Ratio $ratio$
Output: Sample set $\mathcal{C}$ that contains conflict samples

```
Function GenerateTiles(BoxArray, P, ratio) ;
forall b of BoxArray do
    e ← Enlarge b by 2r;
    e ← Scale e by ratio;
    Point op ← Original position of e in KD-tree;
    Point rp ← Random position in pattern;
    e ← Translate e to rp;
    ClippedSamples ← Clipping pattern using e;
    TransSamples ← Translate ClippedSamples to op ;
    TransSamples ← Scale TransSamples by 1/ratio;
    C.Add(TranslatedSamples);
end
```

After each tile being tiled on the building blocks of $\mathcal{D}$, now we get a point set $\mathcal{C}$ with maximal coverage, as is shown in Figure 2.6(a). But the enlarged boundary does not satisfy the requirements of minimum distance, as each tile may have several conflicts in margin area with its neighbors. Next, we need to do elimination and insertion to resolve this problem.

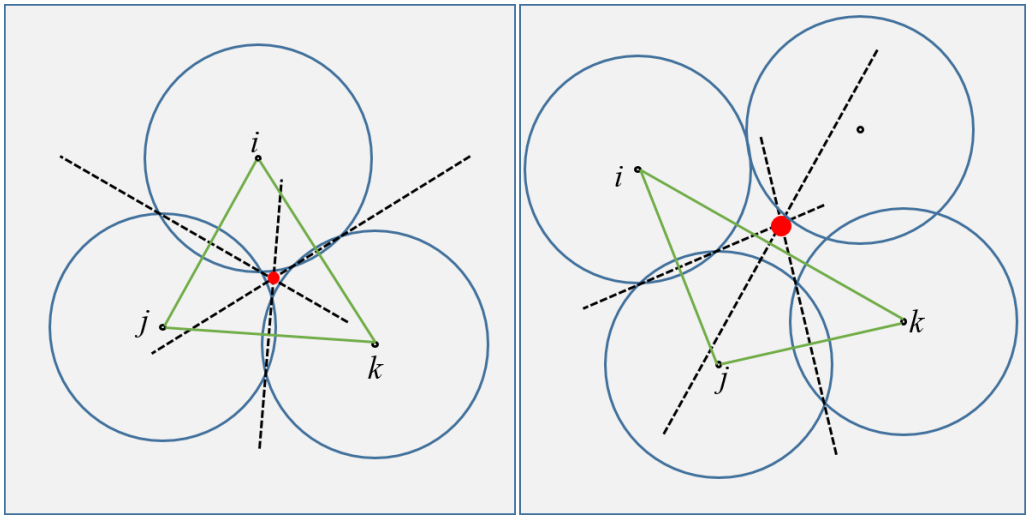


Figure 2.7: If the circumcenter of the triangle i, j, k is not in a gap, then there are no gaps exist between these three disks.

2.3.3 Conquering: Elimination and Insertion

Currently, as we have clipping and tilling result with maximal coverage, the most important task now is to resolve conflicts inside this sample set. Elimination procedure is simple and intuitive, for each sample point being processed, just mark the conflict points as INVALID. After each elimination, a gap may or may not appear due to the elimination. We call the gaps left by each elimination the *Generated Gap*. We call the neighbors within a certain radius of the eliminated points the *Text Buffer*, as these samples in the buffer are candidate samples for gap detection. The main goal of elimination and insertion step is to eliminate all conflict sample points and insert new sample points to cover all generated gaps caused by sample elimination.

Generated Gap Analysis

Drawing inspiration from [Yan and Wonka, 2013], we know that:

Theorem 2 *(Gap-detection): A gap exist among three disks centered at i, j, k ($i, j, k \in \mathcal{C}$), iff the circumcenter c of the triangle i, j, k is not covered by any disks.*

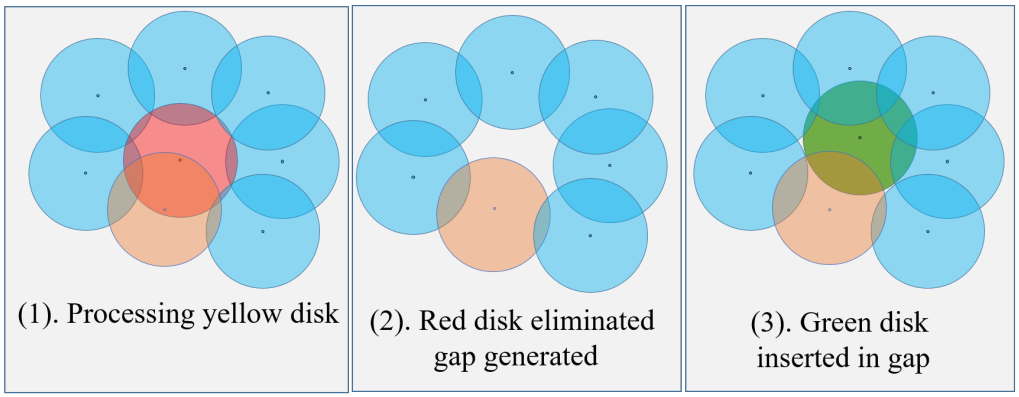
This is also illustrated in Figure 2.7. Using the above theorem, we can do the actual gap detection given a sample buffer called TestBuffer. We define a gap detection and insertion procedure in Algorithm 3.

Note that if GapDetectionAndInsertion being applied on all sample points $p \in \mathcal{C}$, it is not guaranteed that all gaps could be covered. In Algorithm 3, only one point is inserted in each gap detected, which is not enough for some special cases, as illustrated in Figure 2.8. So we introduce a concept *GapDetector* to help locate all gaps in multiple iterations after the first elimination loop.

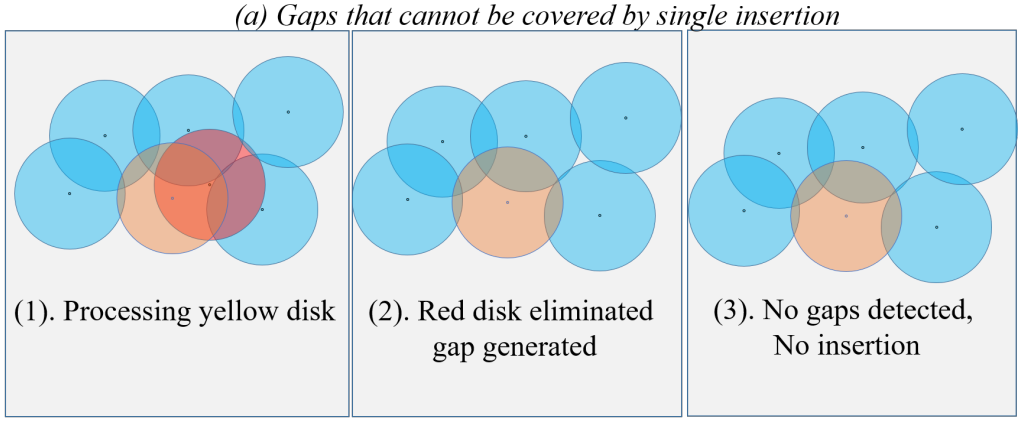
We define the point currently being processed as *pivotpoint*. when a sample point is eliminated from the original set due to conflicts with a pivot point, we mark all disks intersecting the eliminated sample point as *GapDetectors*. It is obvious that all gaps consist of arcs from gap detectors.

Algorithm 3: Psudocode of Gap Detection and Insertion

```
1 Function GapDetectionAndInsertion(TestBuffer,i,C);
  Input  : Sample Point i, TestBuffer, C
  Output: C with conflicts removed and new samples inserted
2 forall every two samples m, n in TestBuffer do
3 |   center ← Circumcenter(i, m, n);
4 |   if center in a gap then
5 |   |   Insert(center, C);
6 |   else
7 |   |   continue;
8 |   end
9 end
```



(a) Gaps that cannot be covered by single insertion



(b) Gaps that cannot be detected by current processing sample

Figure 2.8: (a) Not all gaps can be covered by a single insertion. (b) Not all gaps can be simply detected by currently processing sample point.

Eliminate and Insert Algorithm

There are two main loops in the function: the first loop eliminates each conflict points, and while elimination, inserting only once inside any gap that can be detected directly by pivot point. We know from Figure 2.8 that the gap might not be covered completely with a single insertion, and potential gaps might still exist after each insertion.

To resolve this problem, the second loop is introduced. For each gap detector, do GapDetectionAndInsertion() for a TestBuffer with increased search range for from $2r$ to $4r$ to include all sample disks that consist the gap in case that gap

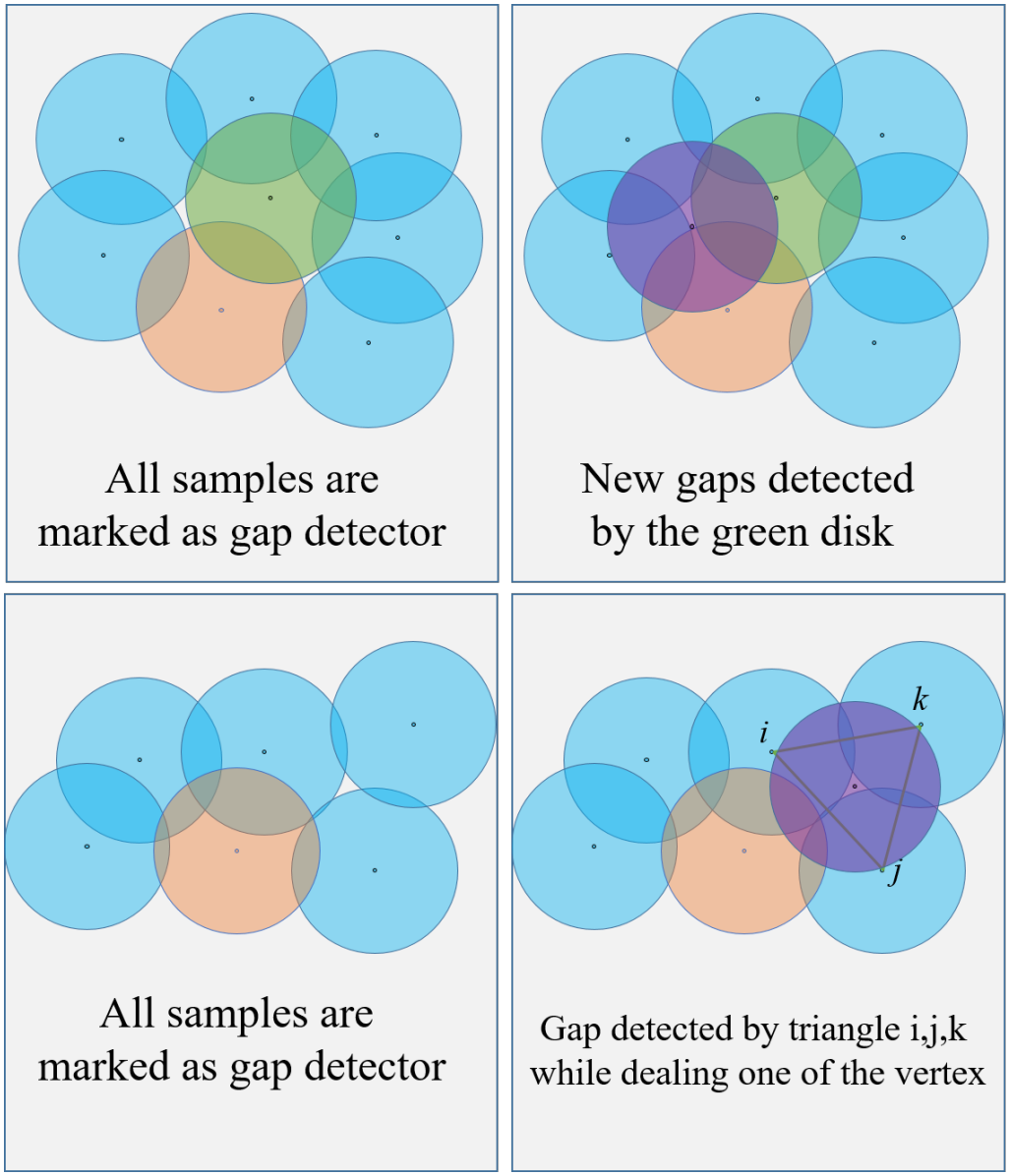


Figure 2.9: Solving problem stated in Figure 2.8 by adding another loop on gap detectors.

detectors might fail to detect neighboring gaps (In practice, we find that only search gap detectors in range $2r$ is enough for gap detection. But here we cannot provide a strict proof to claim so.) From theorem 2 we know that at least one gap detector can successfully detect the gap, and insert a point inside the gap. Mark the inserted point as gap detector again (as the inserted point will be the most possible candidate to help in detecting the clipped gaps caused by the insertion). After the whole loop finished, stream compaction will gather all gap detectors, and start the next iteration. See Algorithm 4.

Termination

The first for loop of Algorithm 4 will terminate as there is a finite number of samples in $\mathcal{C}$. The following while loop will terminate, as for each insertion in the gap, the newly inserted sample disk will cover the finite area, until there are no gaps detected, and no samples inserted. The algorithm will converge after

Algorithm 4: Psudocode of Gap Detection and Insertion

```
Function EliminateAndInsert C;
  Input  : Clipping and Tiling result C
  Output: Maximal Poisson Disk Sample Set S
forall Sample p in conflict area of C do
    Mark p as GapDetector;
    ConflictBuffer ← RadiusSearch(p, r);
    forall conflictPoint in ConflictBuffer do
        Eliminate conflictPoint from C;
        TestBuffer ← RadiusSearch(conflictPoint,2r);
        Mark all samples in TestBuffer as GapDetector;
        GapDetectionAndInsertion(TestBuffer, p, C);
    end
end
Mark all inserted samples as GapDetector;
while True do
    GapDetectorArray← StreamCompaction(C, isGapDetector());
    forall Sample p in GapDetectorArray do
        TestBuffer ← RadiusSearch(conflictPoint, 4r);
        GapDetectionAndInsertion(TestBuffer, p, C);
        Mark all inserted samples as GapDetector
    end
    if No new sample point inserted then
        break;
    else
        continue;
    end
end
```

enough iterations. We can also prove the termination in this way:

It can be concluded that for each iteration of processing gap detectors, big gaps partly covered by inserted samples turn into smaller gaps, small gaps completely covered by inserted points disappear. If we define a gap that needs at most N inserted points to cover as *type N gap*, then each iteration solves the *type 1 gap*, and reduce the type of other gaps by 1. As each gap must have a finite number of type, in this case, the algorithm is able to meet a termination condition, which is all gap type decreased to 1, and then solved by a single insertion loop.

Figure 2.10 illustrated this proof.

We tested iterations needed for convergence by generating 10 million MPS results for 10 times, and actually, all the generation needed only 2 iterations to finish the gap-type decreasing, special cases that need more insertion rarely showed out.

Also note that we don't have to run this algorithm on all $p \in \mathcal{C}$, only operate points in conflict area is enough, as all other points are conflict-free with maximal coverage property (see Figure 2.6), thus no points will be eliminated, and no generated gaps can be detected in those areas.

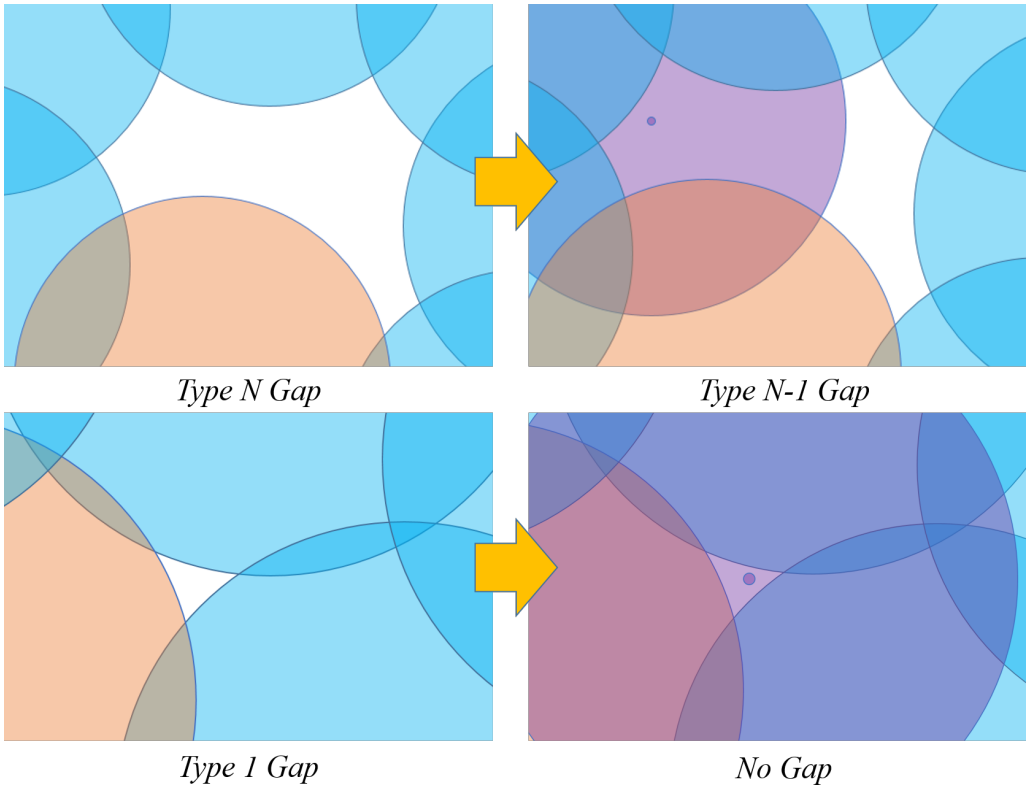


Figure 2.10: Gap type means the largest number of samples we could insert into the gap without breaking conflict-free requirement. Each iteration processing gap detector reduces the gap type for each gap by 1.

2.4 Implementation, Application and Discussion

2.4.1 Implementation Settings

Our testing machine is a PC with a moderate spec: Intel Core i7-4930K Quad-core CPU with 32GB of RAM. GPU test is on a single NVIDIA GTX 1080. There are no intrinsic parallelization patterns of KDRT, we divide the sampling domain into grids, and apply a similar phase group based parallelization pattern as stated in Wei [2008]. That is, dealing with samples separated by a long distance as independent samples, and process them in parallel. Refer Wei [2008] for more details. We used a uniform grid as a radius search data structure on CPU and GPU. There is a clear upper-bound of the number of samples stored for each uniform grid, so the size of each uniform grid can be fixed, and future data will overwrite invalid data in the grid. Note that no atomic operation needed, as the operation on existing samples is marking. Stream compaction after each iteration will manage all the elimination.

2.4.2 Sampling Correctness

The randomized tiling is a tile-based method with the tiling process being completely randomized, thus the results have very similar spectral quality compared to a reference. Refer to Figure 2.12 for more details. KDRT is proved to satisfy the minimum distance (conflict-free) requirement and maximal coverage requirement. Notice that this method is biased, but the biases are bounded, See 2.5.1 for more details.

Avg. Levels	CPU(Avg. Time/ms)	GPU(Avg. Time/ms)
10	16	95
12	27	99
14	59	116
16	172	130

Table 2.1: The time cost of randomized subdivision.

2.4.3 Quality and Performance Analysis

It's obvious that the subdividing depth as a parameter for the randomized KD-tree structure plays an important role in the final quality of this algorithm. If we do the subdivision of target domain really deep, then that means that each tile will be small, which add some more randomness than using a comparable flat KD-tree to clip a big portion of the pattern as the building tile. But it may not worth the overhead as a whole when dealing with too fine-grained subdivision. Also, we know that as a limitation of this algorithm, the minimum length of each dimension for a leaf rectangle should be $6r$, and we find that using $20r$ to $40r$ as the leaf node threshold a good trade-off between speed and quality. When subdividing procedure detects that the dimension will be smaller than the threshold, the recursive subdividing on the particular axis will stop. The whole process will stop until all edge length of every leaf node cannot be further subdivided.

It turns out that the most important parameter in KDRT is the minimum dimension length of the KD-tree used in clipping and tiling: R_{leaf}. Increasing R_{leaf} can reduce the area of conflicting, reduce the number of necessary operations of elimination and inserting. But larger tiles tend to lead the tiling result more regular, as more similar portion can be clipped from the pattern. Fortunately, KDRT has another degree of freedom in randomness: the position of each building block is random, and the position of clipping on the pattern is also random. We did not observe serious quality drop by even increasing the minimum dimension length of KD-tree leaf node to a size closer to the size of the MPS pattern itself.

Discussion on GPU execution and Parallelism

Parallelism is crucial for GPU friendly execution. We divide the algorithm into three-part and analyze how GPU is processing threads for each part and why our algorithm increased through to extend parallelism and GPU friendly execution during each process.

For the KD-tree subdivision part, we use a single thread on CPU to do the generation. We also did an alternative generation to use GPU to effectively generate the randomized KD-tree (developed from Li et al. [2014] which also inspired from Karras [2012]). But we observed (see table 2.4.3 that as our KD-tree is relatively shallow (need less than 16 necessary levels), thus the whole KD-tree construction time consumption is on par with the CPU construction (considering overheads for GPU kernel launch and data transfer). Notice we don't have any underlying geometry to search or divide and only building block's bounding boxes to be recorded in an array, which further reduced the advantages from GPU execution.

For the tiling part, the full parallelism (considering the tiles or samples to be tiled) can be reached easily. There are two patterns to proceed the tiling parallelism on GPU: Per-sample or Per-tile. Each will have a different impact on

R_{leaf}	6r	20r	40r	80r
Percentage	63%	21%	9%	5%

Table 2.2: The percentage of samples being processed.

the final results based on different parameter chosen. The per-sample can reach high parallelism, but each sample will be processed through each bounding box to tile on a correct location. While the per-tile pattern's parallelism is low (the same as the number of leaf-node), but the execution is regular for GPU friendly execution. We use an empirical number that with less than 14 levels, the per-sample parallelism will be used. While with more than 14 levels, the per-tile pattern will be used, maximize the parallelism as well as total performance.

For the elimination insertion part, we used the phase-group based parallelism described in Wei [2010] to make the process parallel. This process is not an intrinsic parallelization problem, as the insertion will be ensured to happen after the elimination. One effective part of our elimination pattern is that the elimination and insertion will only be processed on samples in conflict area, thus greatly reduced the number of samples to be processed. The percentage will be showed in table 2.4.3 (from an averaged randomized tiling). We can see that the effective parallelism will be accessed by threads for valid samples being processed.

Using consistent thread: To effectively maintain parallelism for GPU execution, we also used a technique called consistent thread (Aila and Laine [2009] for the tiling process, that the warp of threads would automatically fetch new work to do, and observed a 3 5 percent performance acceleration. Notice that this tuning is heavily architecture-dependent.

Single Core CPU Performance

Besides the standard GPU implementation, the algorithm can also be fast in CPU performance. Though we didn't use the single-core CPU algorithm for computing Poisson-disk pattern, it is still useful to reveal how the algorithm performs linearly accordingly to the number of samples. See Figure 2.11 for more information.

Comparison

We compared our result with BNOT(De Goes et al. [2012]) and MPS(Ebeida et al. [2012]). Ebeida et al. [2012] is an unbiased Poisson-disk sampling method with maximal coverage feature, and also being one of the fastest high-performance MPS generating methods. De Goes et al. [2012] is also famous for its good performance in applications require blue noise samples. We know from various theoretical analysis that suppressed low-frequency domain power is critical in applications such as Monte-Carlo integration, which explains the advantages that BNOT(De Goes et al. [2012]), MPS(Ebeida et al. [2012]) and our methods over uniform random sampling in Figure 2.12. The first row in Figure 2.12 is 2D point samples, the second row is the power spectrum of the frequency domain, the third row shows how a zone plate $z = sin(x^2 + y^2)$ is rendered by 1 million of each sample, and the last row shows the San Miguel scene (modeled by Guillermo M. Leal Llaguno) rendered with a bi-directional path tracer with 200 million camera ray sampled by each algorithm. We could observe similar anti-aliasing and denoising quality of De Goes et al. [2012], Ebeida et al. [2012] and our algorithm, with our algorithm having a clear advantage in sample generation speed. Another important aspect of our algorithm will be the percentage for each part. Figure

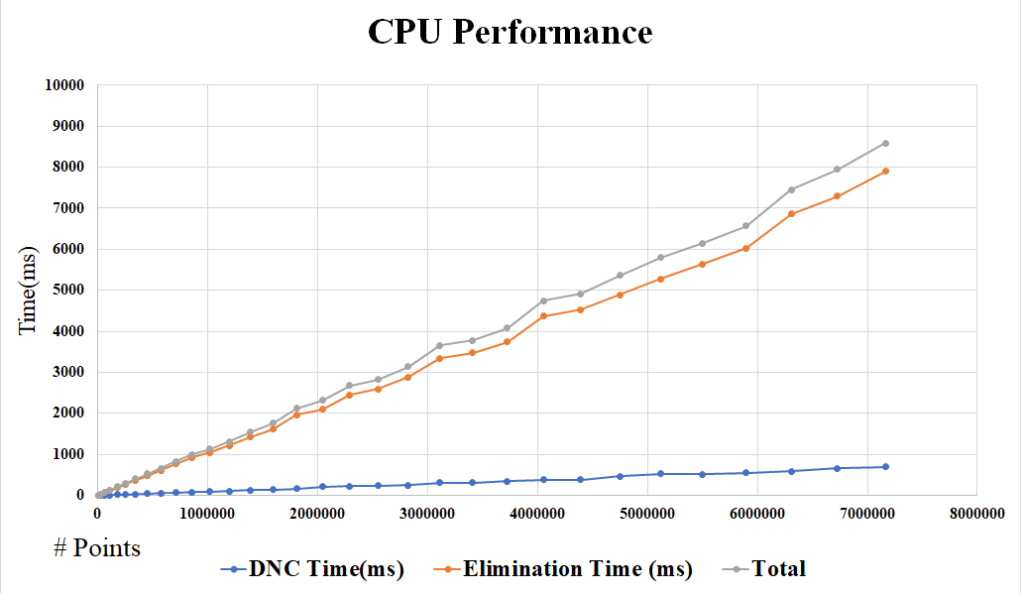


Figure 2.11: Single core CPU performance. The time is of two part: divide and conquering. From the CPU execution time we can observed that the elimination will take most of the generation time.

2.13 will shows the percentage for clipping and tiling, elimination and insertion, gap detection iterations for different R_{leaf}.

We compared our implementation to GPU implementation of Ebeida et al. [2012] for performance. As their result is an implementation of unbiased Poisson-disk samples with maximal coverage feature, and also being one of the fastest high-performance MPS generating methods. In the grouped Figure 2.12, we calculated 10 average of various radius requirements and different parameter of R_{leaf} (the threshold of tile size) for each unit to compare. The frequency power spectrum of samples with 0.01 and 0.005 radii are scaled to the same level as radius 0.0025 for better visualization, with radial mean of the power spectrum calculated on the resolution according to the scaling. From the results shown in the figures (Figure 2.15 shows the reference results generated by [Ebeida et al., 2012], Figure 2.16 shows $R_{leaf} = 6r$, Figure 2.17 is the results with $R_{leaf} = 20r$, Figure 2.18 shows $R_{leaf} = 40r$ and 2.19 with $R_{leaf} = 80r$. It can be noticed that even combining big tiles at a level of $80r$, we can barely see any regularity patterns that often appear in other tiling based method. Also noted that KDRT provides 2.5x to 4x sampling rate for high-performance applications.

2.4.4 Combining Large MPS Patterns

Besides generating MPS using KDRT, the elimination and insertion framework can also be applied in combining independently generated MPS patterns together to form a complete sample set. Sometimes these MPS patterns are generated by different cores or different nodes in the network all in parallel. Using the elimination and insertion framework can effectively turn every serial algorithm into a parallel version, with only very small overhead. See Figure 2.14 for a simple example of this application. Figure 2.14 shows how this procedure works.

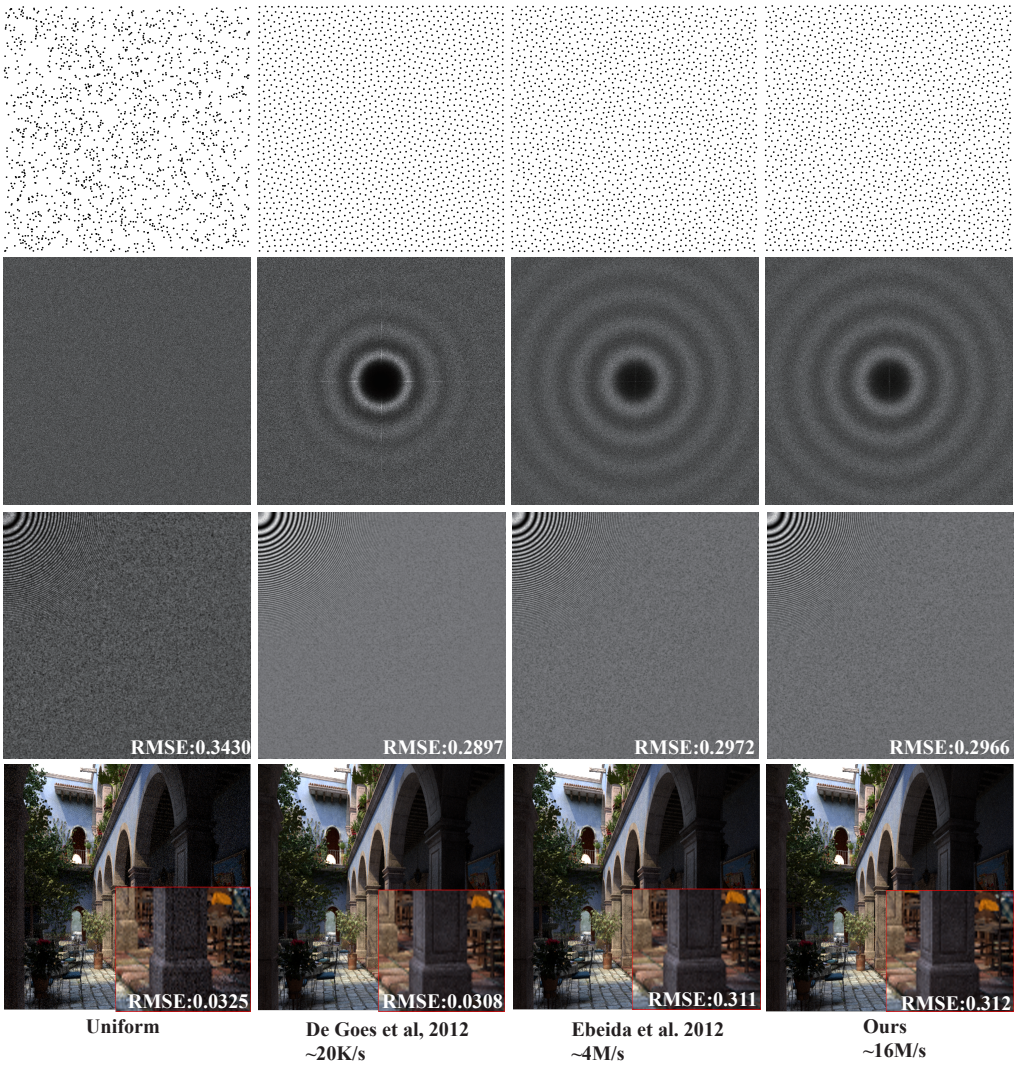


Figure 2.12: Comparison of sample quality. Row 1: 2D samples. Row 2: Power spectrum. Row 3: Zone plate function $z = sin(x^2 + y^2)$ sampled by 1 million samples each with Gaussian filter. Row 4: San Miguel scene rendered with 200 million camera rays (generated by each method).

2.5 Conclusion

In this chapter, we present KDRT, an efficient and robust method to generate maximal Poisson-disk samples. This method replicates tiles clipped from a very small pattern on a randomly subdivided KD-tree based sampling domain, then eliminate all conflicts in margin area, and insert new samples to ensure maximal coverage. This method is intuitive and easy to implement. The elimination and insertion framework can also be used orthogonally to parallelize almost any other MPS generation methods. The rough sketch in Figure 2.14 illustrated a process of combining 9 really large sample sets with our method.

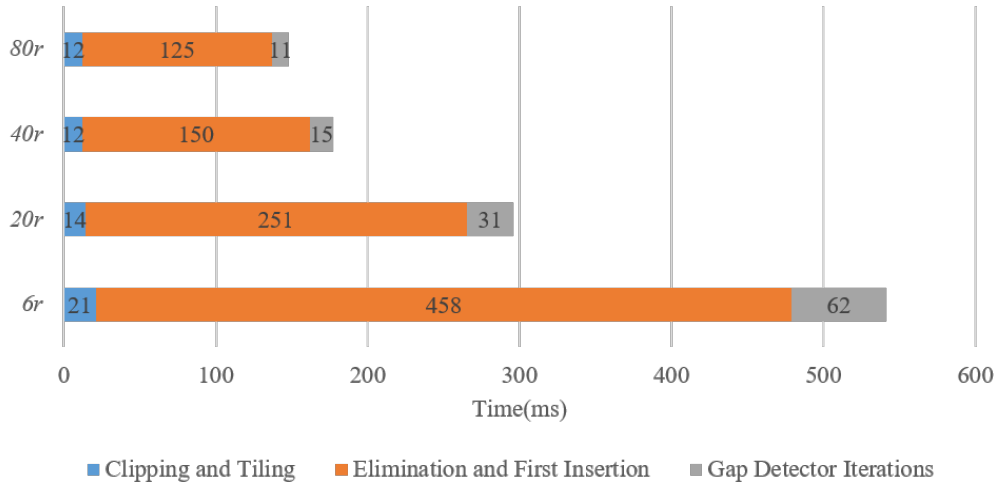
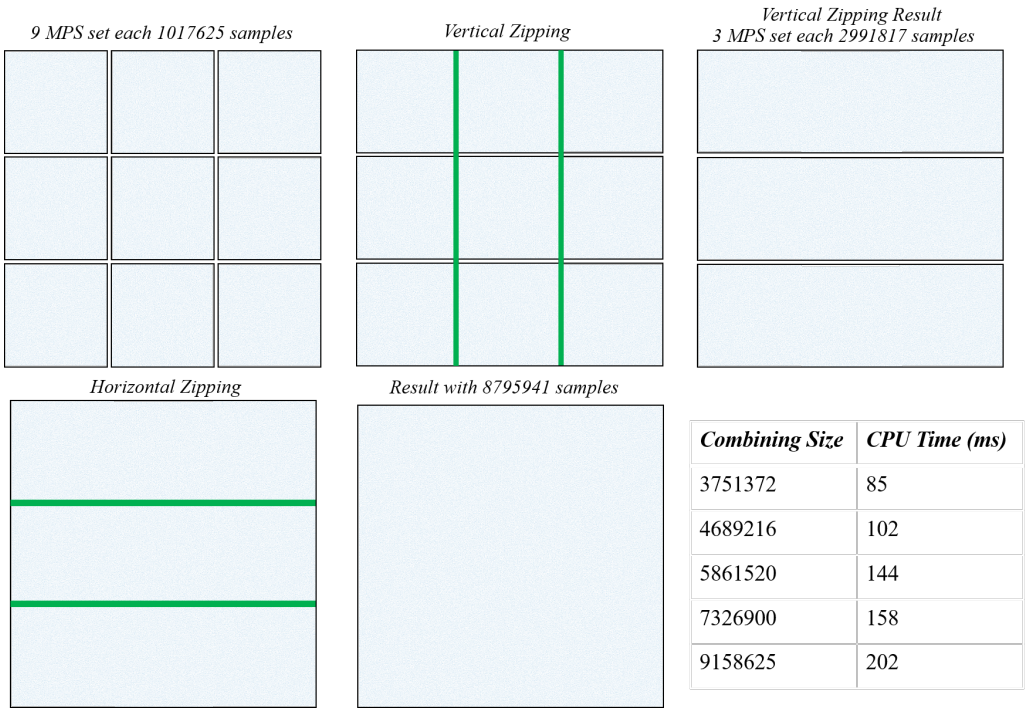


Figure 2.13: Time taken by each procedure to generate 1 million maximal Poisson-disk samples. The elimination and first insertion part takes the longest, as it is operated against the most amount of samples.



Combining Size	*CPU Time (ms)*
3751372	85
4689216	102
5861520	144
7326900	158
9158625	202

Figure 2.14: Example of combining independently generated MPS samples by applying our elimination and insertion framework. Here is an example of combining 9 MPS sets into one. With a vertical and horizontal *zipping*, all points are glued together by elimination and insertion, with only minor loss of samples in the margin area.

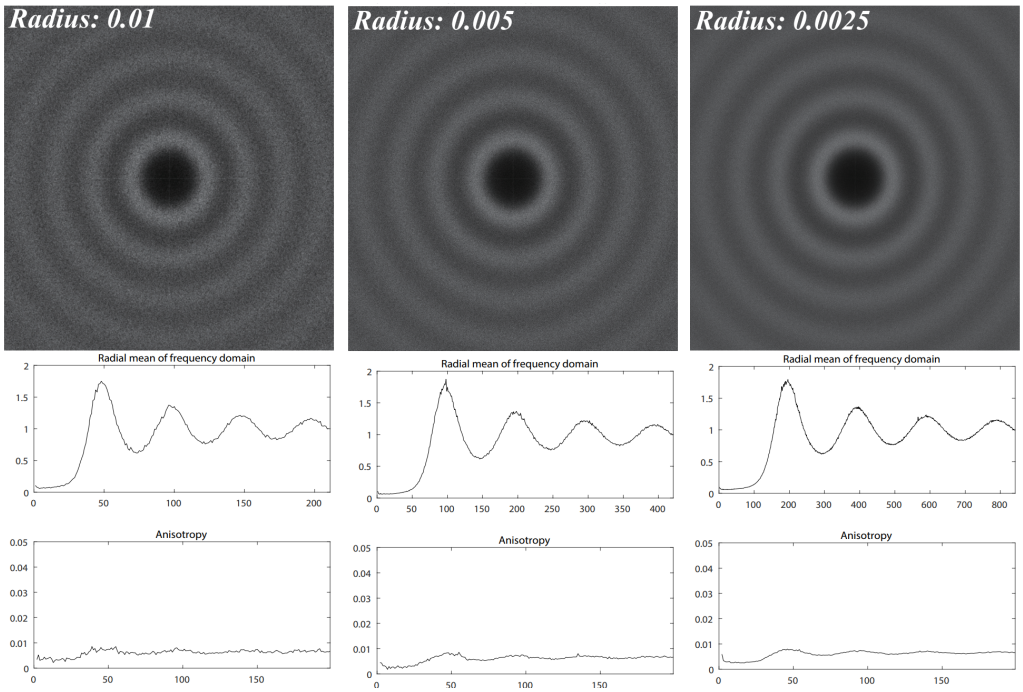


Figure 2.15: The power spectrum, radial mean and anisotropy of the reference samples, generated using the Maximal Poisson-disk sampling methods from Ebeida et al. [2012], with a sampling rate of **4.3M/s** on GPU.

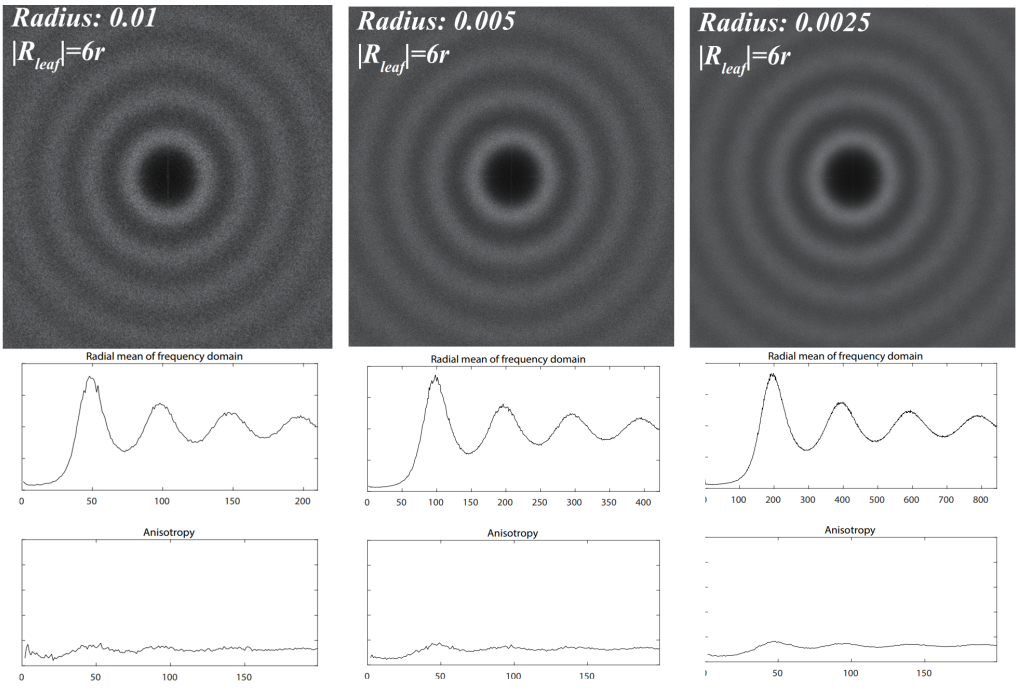


Figure 2.16: KDRT sampling result with the leaf node dimension constrained to 6r. This is a quite fine-grained division of sampling domain, thus with the lowest sampling rate. Even so, the results are comparable to reference, with a sampling rate of 1.84M/s on CPU and **10.56M/s** on GPU, which is 2.4 times faster than the referenced method Ebeida et al. [2012].

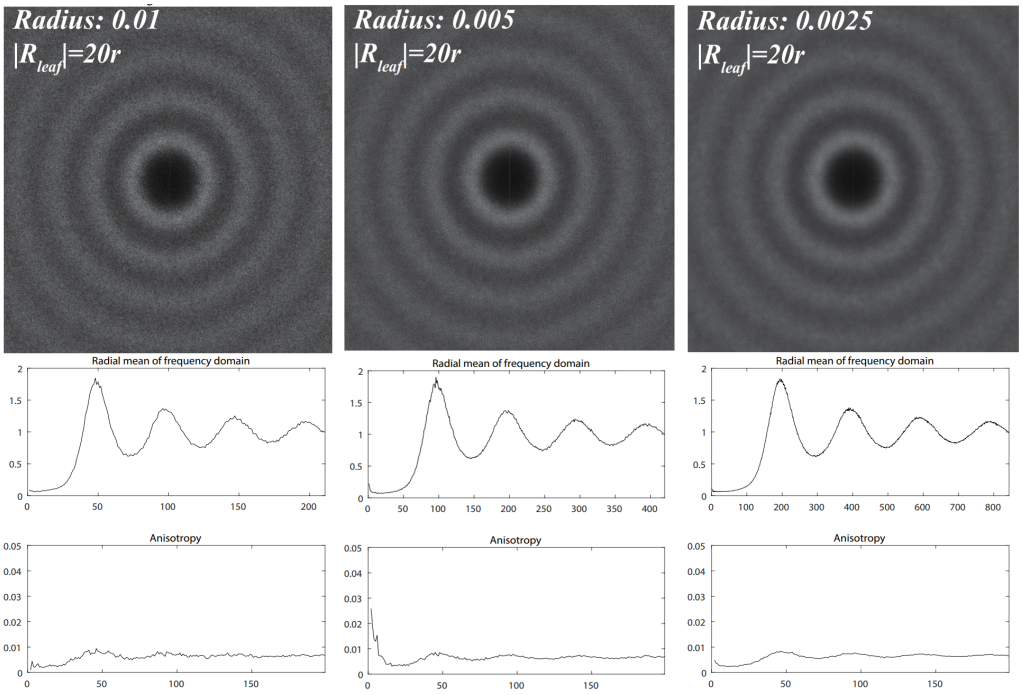


Figure 2.17: KDRT sampling result with the leaf node dimension constrained to 20r. The results is comparable to reference, with a sampling rate of 3.37M/s on CPU and **13.15M/s** on GPU, which is 3 times faster than the referenced method Ebeida et al. [2012].

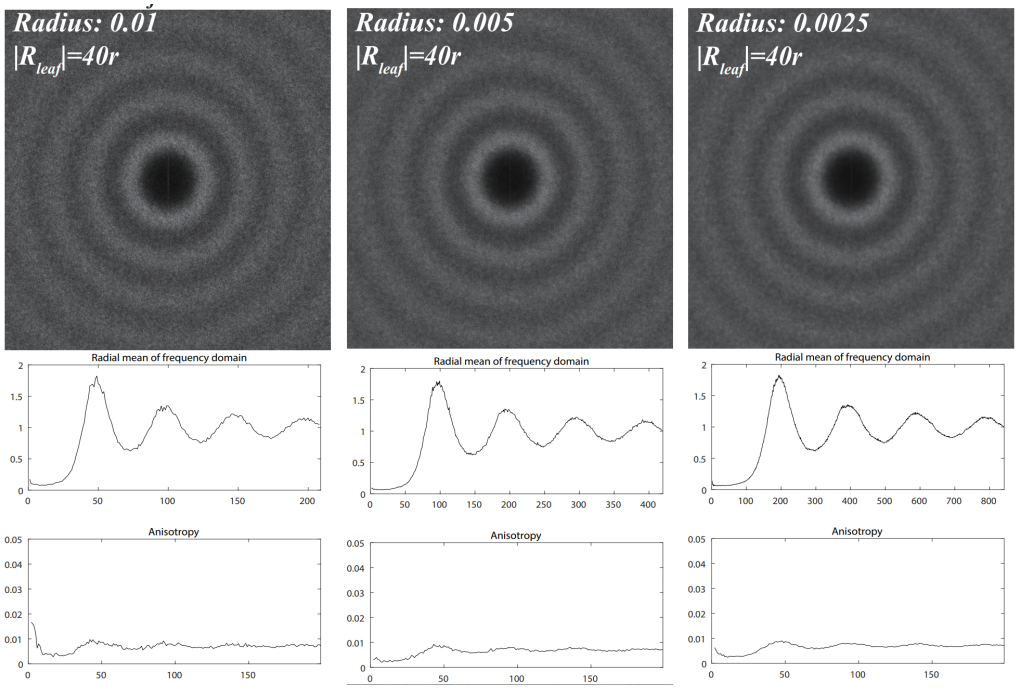


Figure 2.18: KDRT sampling result with the leaf node dimension constrained to 40r. The results is comparable to reference, with a slight visible bias artifacts, with a sampling rate of 5.66M/s on CPU and **16.75M/s** on GPU, which is 3.9 times faster than the referenced method Ebeida et al. [2012].

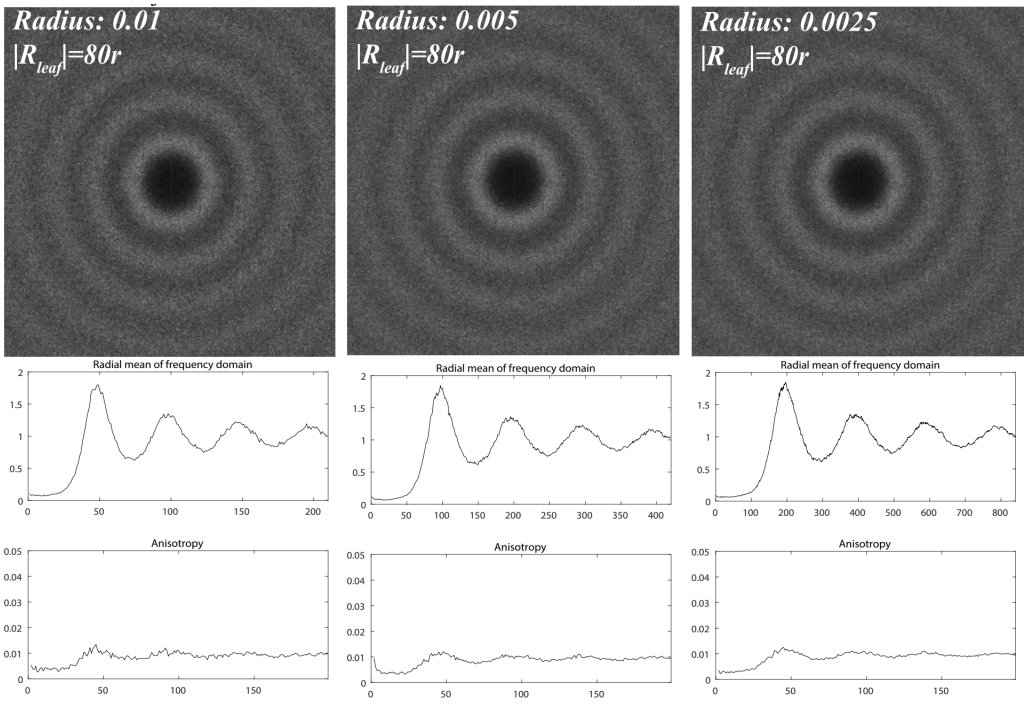


Figure 2.19: KDRT sampling result with the leaf node dimension constrained to 80r, which indicates a large tile. The results is comparable to reference, with a slight visible bias artifacts, with a sampling rate of 6.76M/s on CPU and **18.14M/s** on GPU, which is 3.9 times faster than the referenced method Ebeida et al. [2012].

2.5.1 Limitation

Pre-generation of MPS Pattern

A clear limitation is that this method cannot generate MPS from scratch, a pre-generated pattern with relatively high quality is needed.

As is discussed in previous chapters, that the size of the MPS pattern is quite flexible, that with two randomnesses, a fairly small MPS pattern (less than $\frac{1}{256}$ of the final target) can already generate high quality maximal Poisson-disk samples. In a practical implementation of this algorithm, a pre-generated Poisson-disk sampling pattern of size 8192 can be hardcoded into the program. But it is still one limitation of the algorithm, that if samples needed to be generated from scratch, a fair amount of time will be needed for the generation. We use the Ebeida et al. [2012] as a standard generation method of MPS pattern used in the KDRT.

Parallelization of Insertion and Elimination

Although we managed to extend parallelism to the most, parallelization of insertion and elimination process is still an imitation. There are no intrinsic parallelization patterns of this method, as sample markings are directly related to insertion and elimination during the process. Hence parallelize KDRT take efforts in dealing with scattered memory access and irregular operations on a data structure.

One important issue of our method that has been mentioned many times is that this method has a bias in the conflict area, which is introduced by the

deterministic sample insertion step. But it can be observed that the number of samples introducing bias is bounded, thus the total error of the tiling result is bounded. Consider the minimum tile edge length being Nr, r being the radius of Poisson-disk in the final sample set. With tiling redundancy introduced by our method, the area that each tile covers in the sample domain is $(Nr + r)^2$. Thus the portion of the area that the elimination and insertion happening will take:

$$\frac{(Nr + r)^2 - (Nr)^2}{(Nr + r)^2} = \frac{2N + 1}{(N + 1)^2} \tag{2.4}$$

which is inversely proportional to $N(N > 0)$, and will be less than 10 percent with $N > 19$. So as long as the tiling unit is not too small, the bias error in the conflicting area can be controlled easily.

2.5.2 Discussion: Some further discussion on R_{leaf} and is two randomness actually needed

As we discussed in the previous section that R_{leaf} can be an important parameter that will be related to the correlation corresponding to the radial mean of the power spectrum. We compared different R_{leaf} to see that it will be obvious that the larger the R_{leaf}, the more regularity artifacts revealed in the radial mean. Also as we proceed experimenting the pattern size and its relationship with R_{leaf}, it raises a question that whether the two randomnesses needed.

To explain this problem: we have introduced two randomnesses in the KDRT, the randomness in subdividing the domain, and the randomness in randomly chosen position in the pattern to clip the building block. It seems that we can omit one randomness to accelerate the whole technique. But still, we empirically proved that both the randomness are needed.

As is explained in 2.20, we need two randomnesses to fully ensure the best quality. Also, only less than 5 percent of time taken in the subdivision of the square, so to obtain the best quality from the tiling, it will be recommended to always include two randomnesses in the subdivision as well as in the clipping procedure.

Still, this could be a more complicated problem with two parameters change. We tested limited combinations of pattern size and R_{leaf} to have the above conclusions. To explore a full relationship of the parameter of pattern size and R_{leaf} will be left as related future work.

2.5.3 Discussion: How to sample a specific number of samples

Some applications would require to sample a specific number of samples on the sampling domain. In KDRT, the Poisson-disk radius is easy to control, and the final results show that the sampling quality is of maximal property. But the problem is that it will be hard to control a specific number in the final sampling domain. To achieve the goal, we first generate a rough number of samples according to an average statistics of sampling density calculated from the maximal Poisson-disk sampling pattern relative to the maximal layout of hexagonal regular pattern. With this rough number, we then apply two different techniques working separately to decrease or increase samples to the user-specified number. Suppose the user-specified number is M and what we get from the rough radius calculation is N.

If $N < M$, and we need to increase samples to the final specified number: We can use the method described by Kanamori et al. [2011]. They managed

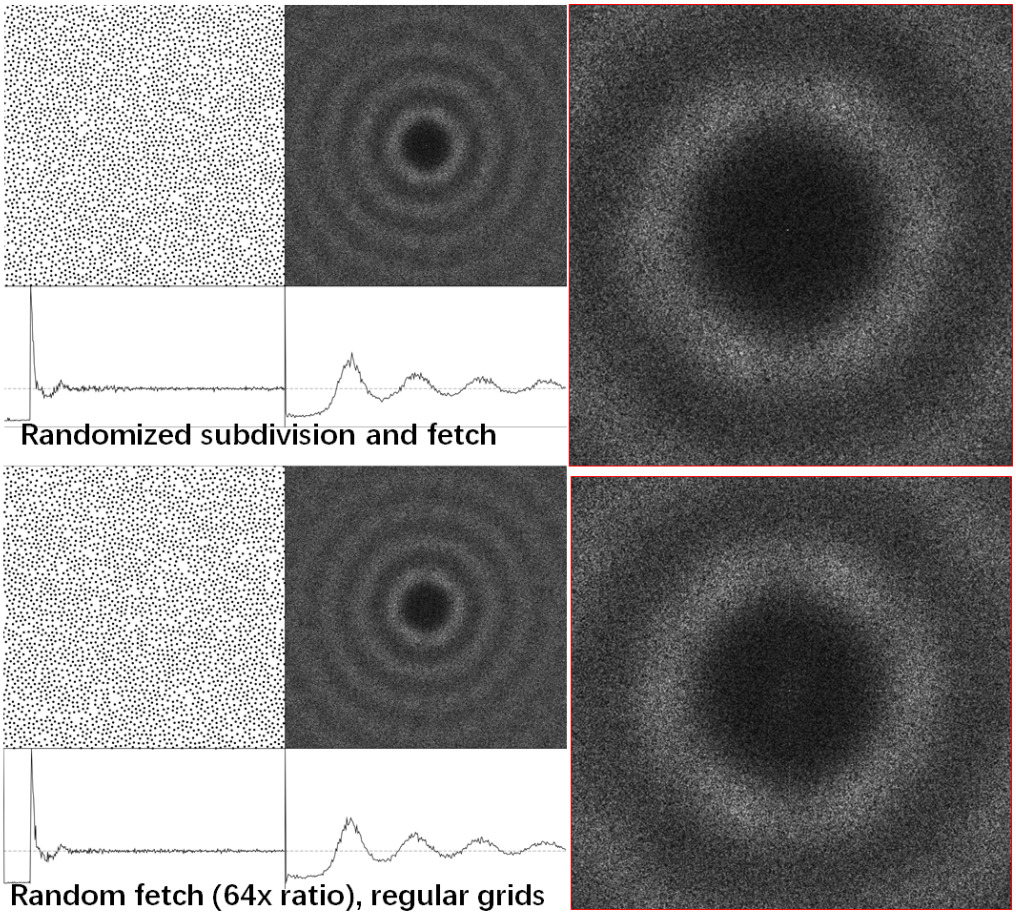


Figure 2.20: It is obvious that only the randomness in the subdivision without randomness in random clipping can cause serious bias problem. And also this bias can be observed if there is only random clipping without randomized subdivision from the above example. The top spectrum is obtained with 16x ratio upscale of the pattern, and the bottom is enlarging the pattern 64x to get the results. We observed the artifacts in the lower part. Which means if we need more effective tiling with more ratio of up-scaling pattern, the randomness in the subdivision will be necessary.

to generate blue-noise sampling by inserting $M - N$ new samples in the largest triangle in the Delaunay Triangulation of the sample pattern. In this way, new samples can be added with the best possible blue-noise property. Although this process will not maintain a maximal Poisson-disk sampling pattern. Conflict will be sure to happen based on the metric of maximal coverage we discussed before.

On the other hand, if $N > M$, it is needed to decrease samples from the overly sampled pattern. We use methods described in [Yuksel, 2015b] to first build a weighted heap for all samples, then eliminate $N - M$ samples from the sample pool. This will also harm the maximal coverage property described in this chapter. To tackle this will be left as future work. We consider using the disk tuning technique described by [Ebeida et al., 2016] can manage to keep the maximal coverage property.

Limitations Comparing with Previous Work

We have compared the KDRT's advantages with maximal Poisson-disk sampling methods and discussed their advantages. Here we'll recap the previous work discussed in the background section, and focus on discussing the limitations of our algorithm compared with them.

Comparing with other Tile-based methods: Comparing with Cohen et al. [2003], Ahmed et al. [2015] and other tile-based algorithms that need to design tile pattern and tile with the pattern size, our method have a drawback in tiling post-process: that after tiling, the post-process of elimination and insertion will be needed and will need a good amount of time to execute. Also, we consider our algorithm better be compared with other maximal Poisson-disk sampling methods instead of other tile-based methods.

For Approximate tile-based methods like Kalantari and Sen [2012], their method can achieve faster tiling speed.

Comparing with maximal Poisson-disk sampling: Comparing with Ebeida et al. [2012], our algorithm is harder to implement, harder to extend to adaptive sampling, and also being more complex thus can be less robust. The Ebeida et al. [2012] also have an ensured quality by its unbiasedness and maximal sampling quality so that the sampling quality will be an upper bound of KDRT's quality, with some sacrifice in the sampling speed of course. Methods like Ip et al. [2013] can also achieve fast sampling speed, and they can achieve better sampling quality, though they require sampling with the rasterization hardware on GPU with limitations in rasterization resolution.

Comparing with relaxation-based methods: Comparing with some typical relaxation-based methods, the KDRT will generate a maximal Poisson-disk sampling pattern instead of a more high-quality blue noise sampling pattern as most relaxation-based methods generated. So KDRT will not perform as well in applications requiring more blue-noise properties. Although relaxation-based methods (BNOT De Goes et al. [2012] mentioned in the comparison part of the previous section) have a low generation rate, their advantage in remeshing is still irreplaceable by a maximal Poisson-disk style sampling strategy like KDRT.

Chapter 3

Progressive Projection Method to Generate Poisson-disk Sample on Mesh Surfaces

In this chapter, we are going to introduce a novel method called Progressive Sample Projection that can generate massive Poisson-disk samples on mesh surfaces in a very short time by projecting blue noise sample patterns from 2D planar space onto meshes. This parallel scheme can exploit full parallelism of GPU without deep recursion or atomic operations, which are often required by other methods. Compared with state-of-the-art methods, the effective generation rate of our method can be 2x to orders of magnitude faster, while still preserving good sample quality. An overview of the algorithm and some application results will be shown in Figure 3.1. This method is also progressive with memory usage bounded, thus being flexible for both performance and quality demanding work. The implementation is straightforward and easy to understand. It can be easily applied to adaptive sampling as well.

This chapter will contain four part. The first part will give a brief introduction to the algorithm. The second part is the introduction of related literature. The third section will explain the algorithm in detail, and the last section is about the implementation, application, and discussion. Some contents of this chapter came from the author's already published paper in [Wang and Suda, 2018], DOI: 10.1145/3233310. The reuse of such contents here is allowed by the ACMs copyright license.

3.1 Introduction

Poisson-disk samples are samples that are uniformly distributed while being no closer to each other than a minimum distance. This pattern has a good blue noise power spectrum and is believed to be beneficial for many computer graphics applications. In this chapter, we are going to propose a comprehensive and efficient method that can exploit full parallelism of your hardware to generate Poisson-disk samples on mesh surfaces in real-time.

Among stochastic sampling patterns, Poisson-disk distribution is a famous example of how simple regulations can produce good features. Remember about the characteristics of Poisson-disk samples we talked about before, that if we denote the sample set as X in the domain D, with a specific sample as x_i, Poisson-disk samples have following properties:

$$\forall x_i \in X, \forall S \subseteq D : P(x_i \in S) = \int_S dx \tag{3.1}$$

$$\forall x_i, x_j \in X, x_i \neq x_j : \|x_i - x_j\| \geq d \tag{3.2}$$

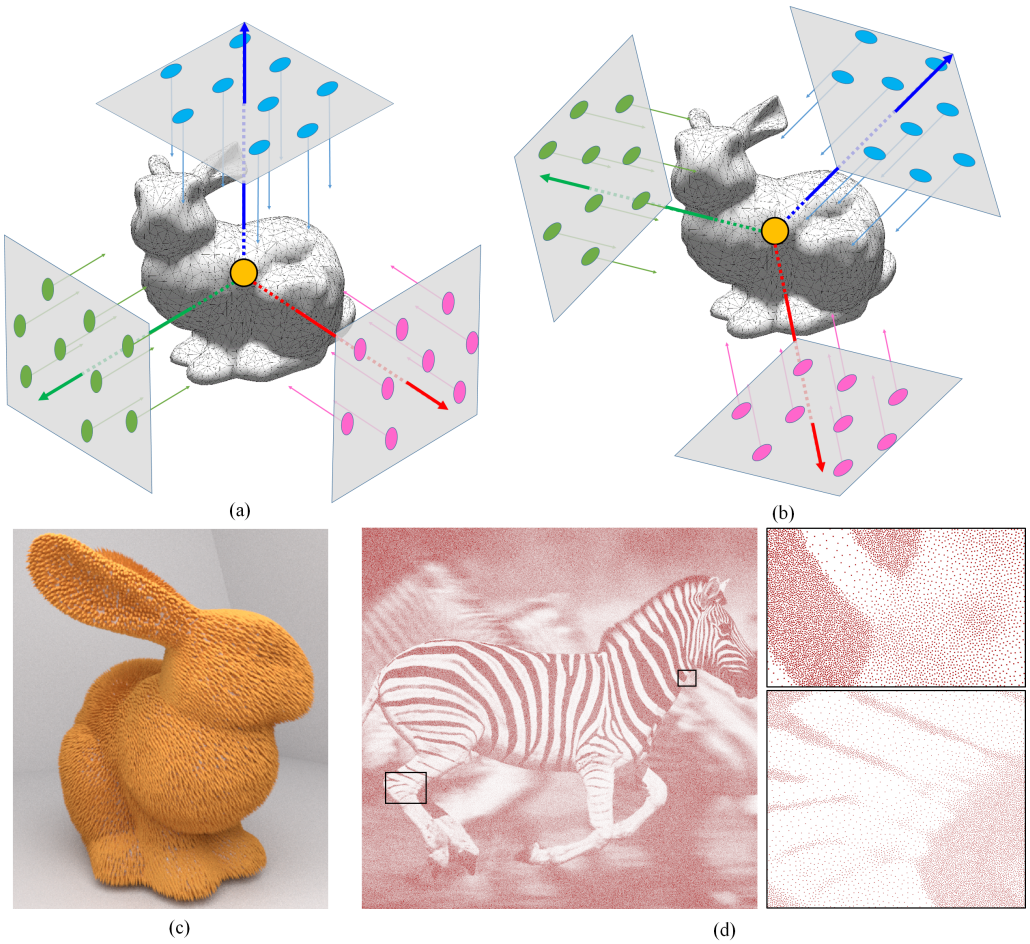


Figure 3.1: (a) Orthogonal ray batch group, 2D Poisson-disk patterns are placed on three planes that are perpendicular. Trace rays starting from the pattern to the mesh, then collect valid intersections. (b) Rotate ray batch group around the center of mesh to start the next iteration. Merge new and previous intersections. (c) 3 iterations can generate applicable results for many applications such as fur geometry synthesis, at a sampling rate of 4.3 M/s on a single GPU. (d) A degenerated application: image stippling of a zebra taken by Sadie Hart. 1.5 million samples of high-quality stippling can be produced with 100 iterations at an effective sampling rate of 200K/s.

Equation 3.1 means that the sample placements should be uniform and unbiased. Equation 3.2 (conflict-free criterion) is a criterion enforcing that samples are further away from one another than d. There are some other requirements, including maximal requirement: no samples can be inserted in the sample set without breaking the minimum distance requirement, and adaptive density requirement: each sample can have its own distance criteria.

To meet the sampling rate requirements of many emerging applications in fields such as global illumination, geometry-based planning and remeshing, etc., we proposed a novel method named *Progressive Sample Projection* (or sometimes *PSP* in this article) that can generate a massive amount of high-quality Poisson-disk samples on 3D models (mesh surfaces) in real-time. This method is also flexible for applications that require very high-quality samples, as the sample set can be progressively improved over time, with bounded memory footprint.

The main idea of PSP is: generate a 2D Poisson-disk sample set, then project

the pattern onto mesh surfaces with ray tracing. The progressive process is inspired by rotating food in the oven such that the fixed heating element in the oven can radiate heat evenly on the surface. In practice it is more calculation-intensive to rotate the object, so we are using similar techniques to rotate the 2D pattern in order to *project* the 2D Poisson-disk pattern evenly onto the 3D model.

Some main advantages of PSP are:

- **High Performance.** One can reach a higher generation rate than 10 million samples per second to generate a point set which meets the demands of many interactive to real-time applications, thanks to the GPU's capability of dealing massive primary ray-objects intersections in a very short time.
- **Flexible.** It can also generate high-quality, near-maximal samples in an efficient progressive manner with high parallelism.
- **Memory footprint bounded.** Unlike some previous works that have to generate the point set from selecting or eliminating a large initial random point set that occupied 10x or more memory than the actual outputs, the memory usage of our method is bounded.
- **Easy implementation.** Understanding and implementing the PSP is straightforward. With the progressive nature of the algorithm, it's also easy to use user-specified density function for adaptive sampling.

3.2 Related Work

Some recent works focused on a sampling domain of mesh surfaces rather than planar spaces. Some focused on ensuring high quality or even maximal property of sample set, such as [Yan and Wonka, 2013], [Guo et al., 2015], [Jiang et al., 2015] and [Zhang et al., 2016]. These works managed to increase the quality of samples to a high level for demanding applications. [Guo et al., 2015] extended a void-management method in planar space to mesh surfaces. [Jiang et al., 2015] analogized SPH method used in fluid simulation on blue noise sampling. [Zhang et al., 2016] managed to generalize capacity-constrained method from 2D to mesh surfaces. These works can generate good quality samples with a low generation rate of thousands per second, which could be more suitable for non-interactive applications.

On the other hand, some put efforts on sampling rate in particular. Most of them used a parallel computational pattern for a boost. The [Bowers et al., 2010] is a a natural extension of [Wei, 2008] in mesh surface domain. They used similar concept from [Wei, 2008] called *phase group* to parallelize dart throw process on mesh surfaces. They also introduced a technique which will be referred in the following sections as *mass elimination method*, similar to [Corsini et al., 2012], to further optimize this process. [Ying et al., 2013] managed to assign each sample a pre-defined priority value to avoid using phase group, while still achieving good parallelism. Methods used in planar space can also provide valuable insights on mesh surface cases, such as [Ip et al., 2013], which is a direct inspiration for our method.

To do an easy evaluation of the quality of a Poisson-disk sample set, people tend to transform some features of the sample set to the frequency domain for direct visualization. [Lagae and Dutré, 2008], [Durand, 2011], [Subr and Kautz,

2013] and [Pilleboue et al., 2015] provided more details. For mesh surface quality evaluation, [Wei and Wang, 2011] provided a comprehensive framework that elegantly transformed this analysis to the differential domain, which can reflect metrics of point set quality directly. We applied their methods in our analysis, too.

3.3 Algorithm Overview

To summarize the Progressive Sample Projection (PSP) method: Generate several 2D Poisson-disk sample planes facing and around the center of the model, trace rays starting from the samples on the plane towards the model, then collect and merge intersections as the result. We call the planes in the 3D space that contains the 2D Poisson-disk pattern as *projectors*. The projectors can be carefully designed so that the model will be better covered by rays, while the stretching effects caused by projection can be controlled. Our design of projectors would be using three planes facing the model and perpendicular to each other. These three planes with samples are called top view, front view and left view projectors. Rays that will be generated from the projectors will be called *orthogonal ray batch group* (or *ORBG*). To make the mesh evenly covered by samples from projectors, the model will be rotated (or equivalently rotate the projectors/rays) to approach better coverage of samples.

This section will briefly go through the entire process. An analysis of the results will be presented in section 3.4. The design of projectors will be discussed in 3.4.3. Some applications using this method will be discussed in 3.4.4.

Figure 3.2 is an overview of this algorithm. The following psudo-code implies some function names which will be explained in detail later:

1. **BuildORBG**;
2. while(true), do:
   (a) **TraceORBG**;
   (b) **MergeIntersections**;
   (c) if meets demands, then break;
   (d) else, **RotateORBG**;

3.3.1 Generation of pattern and ORBG

Generate 2D Poisson-disk Sample Pattern

2D Poisson-disk samples are easier to generate as they did not suffer much from the curse of dimension. To generate a small number of samples, we will be using the method of [Ebeida et al., 2012] for simplicity and robustness. For a mass generation of samples at millions per second rate, a tile-based method focusing on performance will be more suggested. This 2D sample set will be called the *pattern* in the following article.

The length of the 2D sampling domain will be normalized to the diagonal length of the mesh's bounding box. Unless the 3D mesh is extremely prominent in one dimension, the redundancy will not be a serious problem, as shutting down rays with no intersections is extremely fast. The pattern only needs to be generated once for a specified radius.

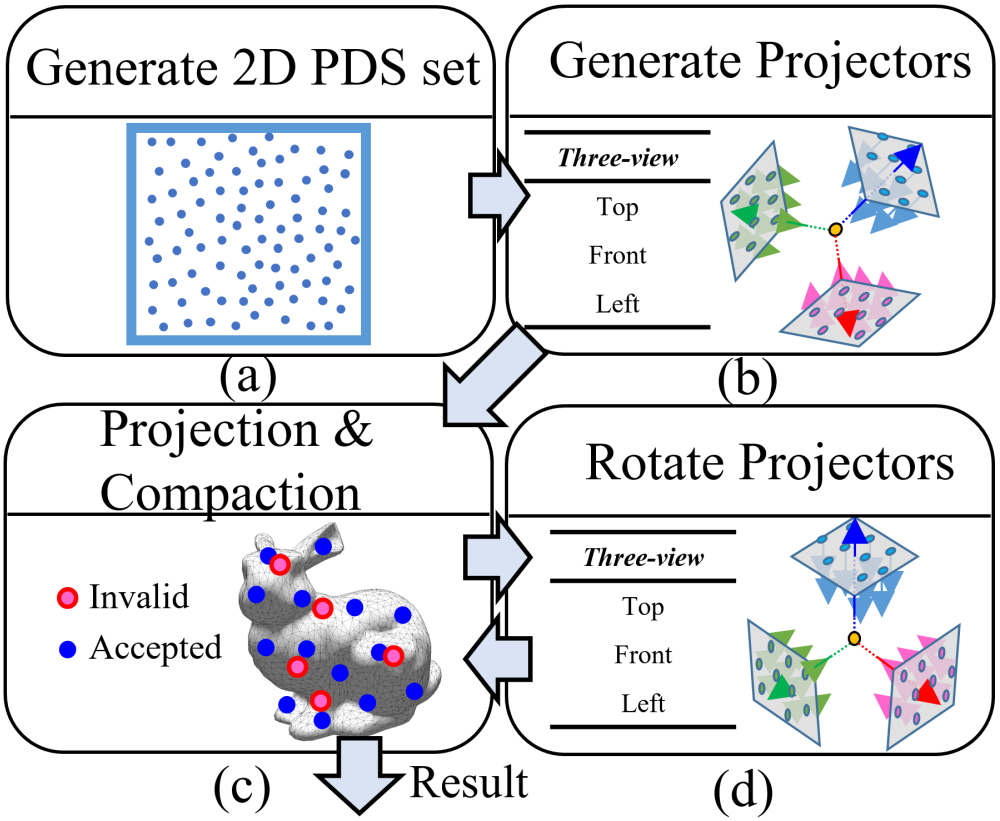


Figure 3.2: (a). Generate 2D pattern. (b). Generate projectors and rays. (c). Trace rays towards model, and merge intersections. (d). Rotate the projector set, and do the ray tracing process again in progressive manner.

Build ORBG

First, specify a random point on the bounding sphere of the model as the top projector's center. Then build the orthogonal basis with the other two vectors pointing to the left and front plane center. Fill all three planes with pre-generated 2D samples using the local coordinate vector u, v to locate the correct positions. Finally, generate rays using these samples as starting points and using normals of these planes as directions. Figure 3.3 shows how rays will be generated from the projectors.

3.3.2 Projection and Compaction

Projection is the process of projecting samples from projectors to the model. Compaction is the process of checking the validity of each sample to decide whether to accept or reject, and merge valid samples with existing samples.

Projection

Given a batch of rays and a triangle mesh, the next goal is to collect the intersection point cloud as the projection results. This can be done by tracing the ray batch with a ray tracing engine. Rays within a ray batch can be relatively friendly for GPU execution due to its regular directions, which is essential for the efficiency of our algorithm. We used the NVIDIA OptiX Prime framework ([Parker et al., 2010]) to trace ray batches in parallel.

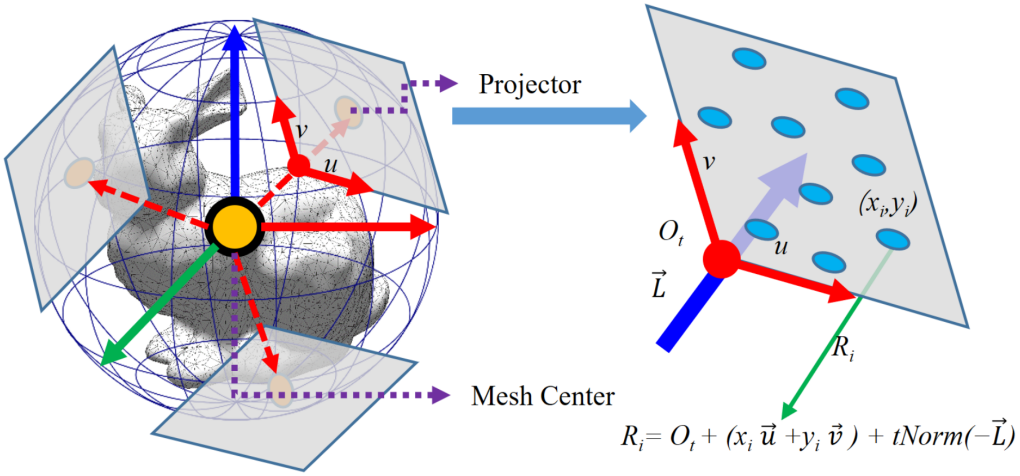


Figure 3.3: Rays generated from projectors.

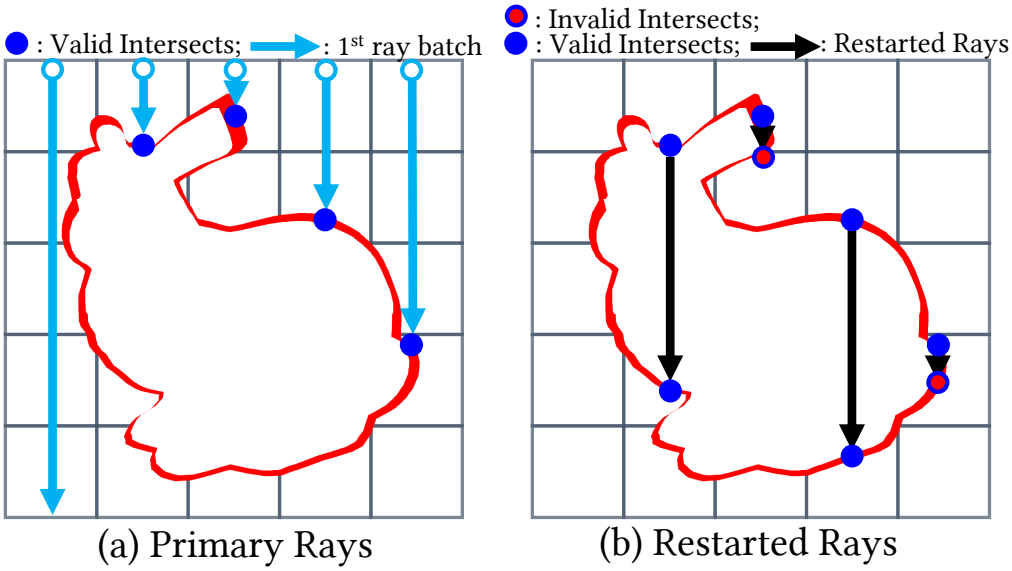


Figure 3.4: A 2D illustration of restarting rays. Consider the contour as the analogy of a 3D model in space. (a) shows the process of tracing primary rays and recording valid intersections in a buffer. (b). Restart a new ray batch from the valid intersections to get more samples.

To sample occluded part of the models efficiently (e.g. models with genus), it is necessary to collect other samples along the ray beside one primary intersection. A new batch of rays will be generated with identical directions but started from the first intersection points instead of the original projectors. Figure 3.4 depicted this process. Note that intersections of restarted rays might *conflicting* (within Poisson-disk radius with) previous samples generated by the same ray, we call these conflicts the *restart conflicts*. Figure 3.4 explained this kind of conflicts. Restart conflicts will be checked and marked on-the-fly in this ray tracing step. The only possible conflicts within the same batch (generated from the same projector) is restart conflicts. While this being taken care of, there would be no intra-batch conflicts anymore. All we have to deal with next is inter-batch conflicts: that samples may have conflicts with other samples generated from a different projector.

All samples are organized in a simple hash grid data structure that will be reconstructed after each iteration, which is similar to [Green, 2010]. We choose the length of a unit grid to be just longer than the specified Poisson-disk sample radius, which can empirically be fast. A `RangeSearch(`s`,`r`)` method is implemented on this data structure that can fetch all samples within a certain radius r from a certain sample s, by searching all samples residing in neighbor grids along three dimensions.

Compaction

From literature such as [Bowers et al., 2010] and [Corsini et al., 2012] we know that eliminating conflicts from a uniformly generated sample pool in parallel is not trivial. In fact, it is not fully parallelized until [Ying et al., 2013] proposed a priority-based technique to solve it, at a cost of introducing arbitrarily long recursive irregularity execution on GPU. Fortunately, our current sample set has no *intra-batch conflicts*, in that case, samples are automatically divided into 3 phase groups with full parallelism, which greatly simplified the complexity of this process.

We test samples for inter-batch-conflicts as follows: A unique number called *priority number* from the set {1,2,3} will be assigned to each of the three ray batches to ensure all rays from the same projector have the same priority. Rays will then carry their priority numbers to each generated samples. Samples conflicting (within the radius of) a valid sample of higher priority will be marked as invalid.

In this case, samples with the priority of three will be marked valid immediately. Priority-2 samples will search for priority-3 samples within the radius to decide validity for itself. While priority-1 samples will be more complex: it will be marked invalid when priority-3 samples detected within the radius. But for any priority-2 samples within the radius, it will have to first check the priority-2 samples' validity recursively before making decisions. In practice, this shallow recursive pattern (at most one recursive call) can be easily unrolled to a for-loop.

Finally, the valid samples will be tested against samples already existed on the mesh surfaces. We find it faster to first resolve inter-batch conflicts and then merge results with existing samples, as there will be more parallelism and less kernel launch. Besides, the reconstruction of the underlying hash grid can be done once instead of three times in the iteration. Stream compaction will be performed to wipe out all invalid samples, and add the legal samples to the existing sample array, marking the end of one iteration. Figure 3.5 illustrated the merge intersections process of different ray set with a 2D contour analogy. Algorithm 5 gives a description of the compaction process. More discussion concerning the design schemes of projectors is in 3.4.3.

3.3.3 Rotate ORBG

Finally, rotating the ORBG and progressively re-do the iteration can provide better sample coverage. The rotation process is simple, just apply the same random rotation around the center to all samples on all projectors. To set a destination of rotation, a new random point on the bounding sphere will be chosen, and a rotation matrix will be calculated to transform the old projector center to the new center. All samples will be rotated under the same rotation matrix calculated.

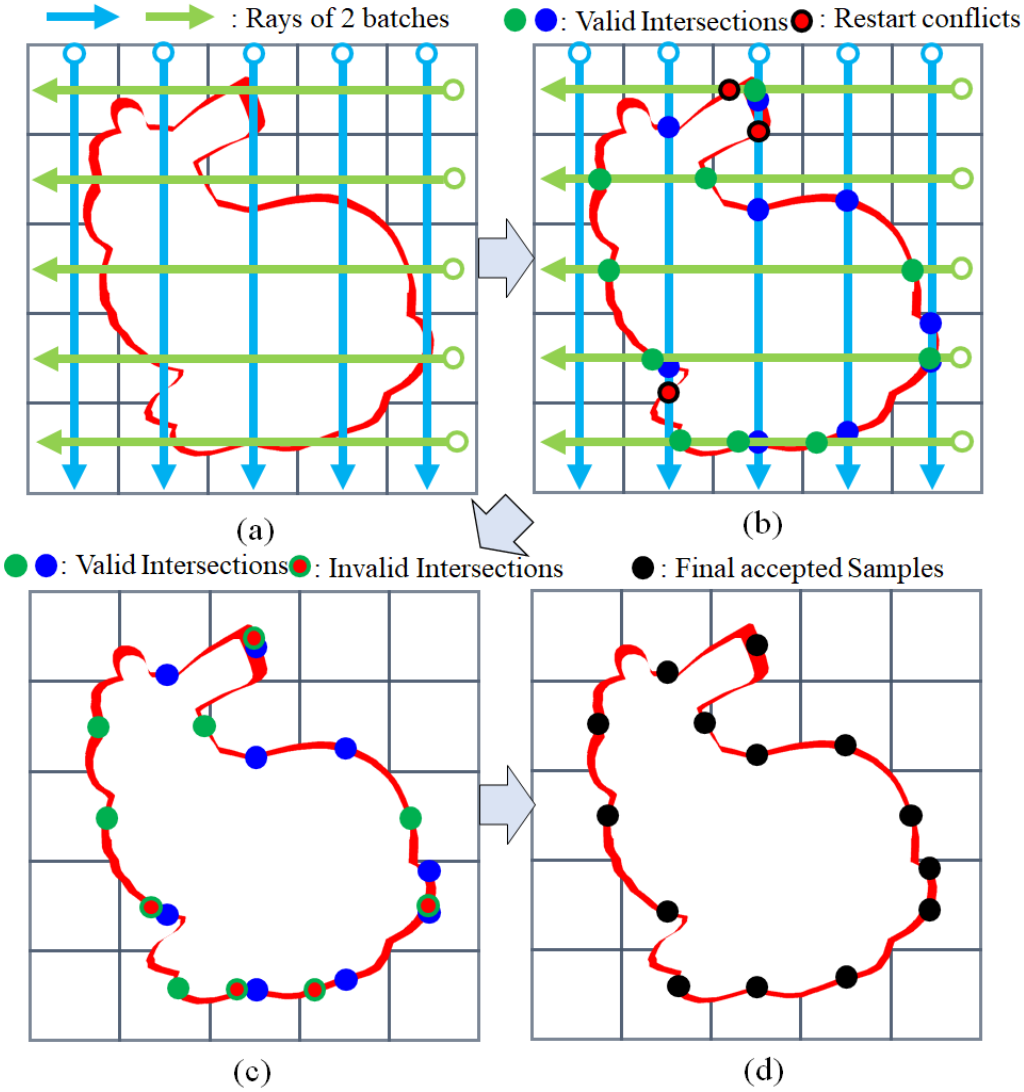


Figure 3.5: A 2D illustration of merging results from different ray batches. Consider the contour as the analogy of a 3D model in space. (a). Two batches (in 3D, 3 batches) of rays generated from projectors. (b). Trace rays to get intersections, eliminate restart conflicts. (c). Eliminate conflicts between batches (inter-batch conflicts). Without loss of generality, the top projector will have a high priority, thus samples projected from the other projectors conflicting with top projector samples will be marked invalid. (d). The final results. Samples will then be merged with existing samples.

One issue is that we are using a rejection sampling technique to reject rotations with too small changes from the previous one to ensure a distance between iterations. Sampling from a projector that is extremely near the previous one often fails to provide many samples that can be accepted, so this rejection sampling scheme is necessary in choosing the next destination for the rotation.

Besides the previous strategy, we could use [Kanamori et al., 2011]'s strategy to generate new position to do the sampling. Their method of generating samples with progressively denser density but still kept the sampling with blue noise will also be suitable for the choise of next sampling position.

Algorithm 5: Tracing ORBG and merge projected samples with existing samples.

Input : Ray batches: $\mathcal{R}_t$, $\mathcal{R}_l$, $\mathcal{R}_f$, Existed Samples: $\mathcal{S}_{exist}$, model: $\mathcal{M}$
Output: Updated $\mathcal{S}_{exist}$ with newly generated samples

Function *TraceORBG($\mathcal{R}_t$, $\mathcal{R}_l$, $\mathcal{R}_f$, $\mathcal{M}$)*
  forall $r_i \in \mathcal{R}_t, \mathcal{R}_l, \mathcal{R}_f$ **do**
    $r_i.priority \leftarrow$ 1 or 2 or 3 accordingly.
  end
  $\mathcal{R} \leftarrow \mathcal{R}_t \cap \mathcal{R}_l \cap \mathcal{R}_f$
  SampleSet $\mathcal{S}_{current} \leftarrow$ TraceAndRestart($\mathcal{R}$, $\mathcal{M}$);
  MergeIntersections($\mathcal{S}_{current}$, $\mathcal{S}_{exist}$);
end
Function *TraceAndRestart($\mathcal{R}$, $\mathcal{M}$)*
  SampleSet $\mathcal{S}_{current}$;
  Rays $\mathcal{R}_{restart}$;
  IntersectionPoints $\mathcal{P}_x \leftarrow$ Trace($\mathcal{R}$, $\mathcal{M}$);
  while *CountValid($\mathcal{P}_x > 0$)* **do**
    forall $p_i \in \mathcal{P}_x$ ***in Parallel*** **do**
      Ray $R_i \leftarrow$ Restarted ray from p_i;
      $\mathcal{R}_{restart}$.Add(R_i);
      if p_i *conflicts previous intersections* **then**
        mark p_i as invalid;
      end
    end
    $\mathcal{S}_{current}$.Add(Valid($\mathcal{P}_x$));
    Clear($\mathcal{P}_x$);
    $\mathcal{P}_x \leftarrow$ Trace($\mathcal{R}_{restart}$, $\mathcal{M}$);
  end
  Return $\mathcal{S}_{current}$;
end
Function *Trace($\mathcal{R}$, $\mathcal{M}$)*
  IntersectionPoints $\mathcal{P} \leftarrow$ RayTracingAPI($\mathcal{R}, \mathcal{M}$);
  Record corresponding priorities to intersection points;
  return $\mathcal{P}$
end

3.4 Implementation, Application and Discussion

We tested our results on a moderate consumer spec PC with an Intel Core i7-7700K CPU processor and a graphics card of NVIDIA GTX 1080. We tested different radius specification of 2D Poisson-disk samples on a total of 6 different meshes. Figure 3.8 shows the models and some effective sampling rate with different numbers of ORBG rotation iterations specified. Figure 3.7 are sampling results for the Bunny model with different radius specifications, each with five iterations of ORBG rotations. Our method is not sensitive to geometry complexity, so sampling on a more complex model such as model can still keep a very high sampling rate. Figure 3.9 is a topologically complex scene that is not friendly for ray tracing with restart, but our method can still manage to generate a massive number of samples at a good quality and generation rate.

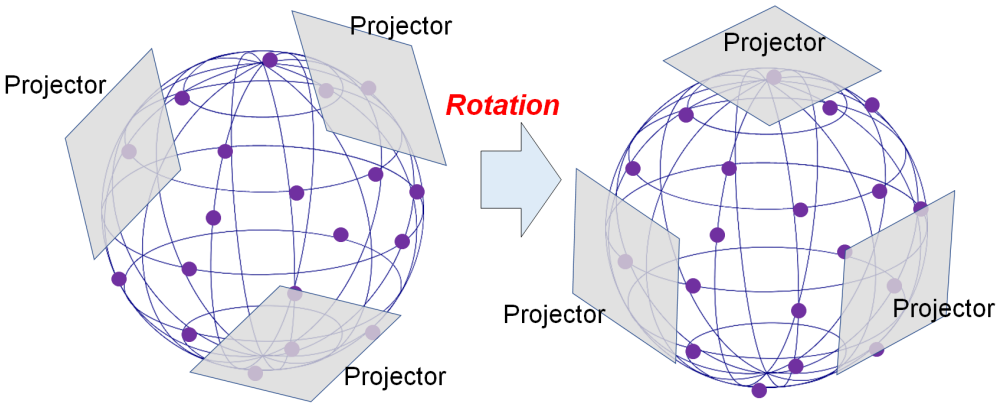


Figure 3.6: The next position of the ORBG can be decided deterministic beforehand with blue noise sampling on the sphere, or it can be determined by rejection sampling: that if the current sampling is near the previous one, reject the sampling and start over.

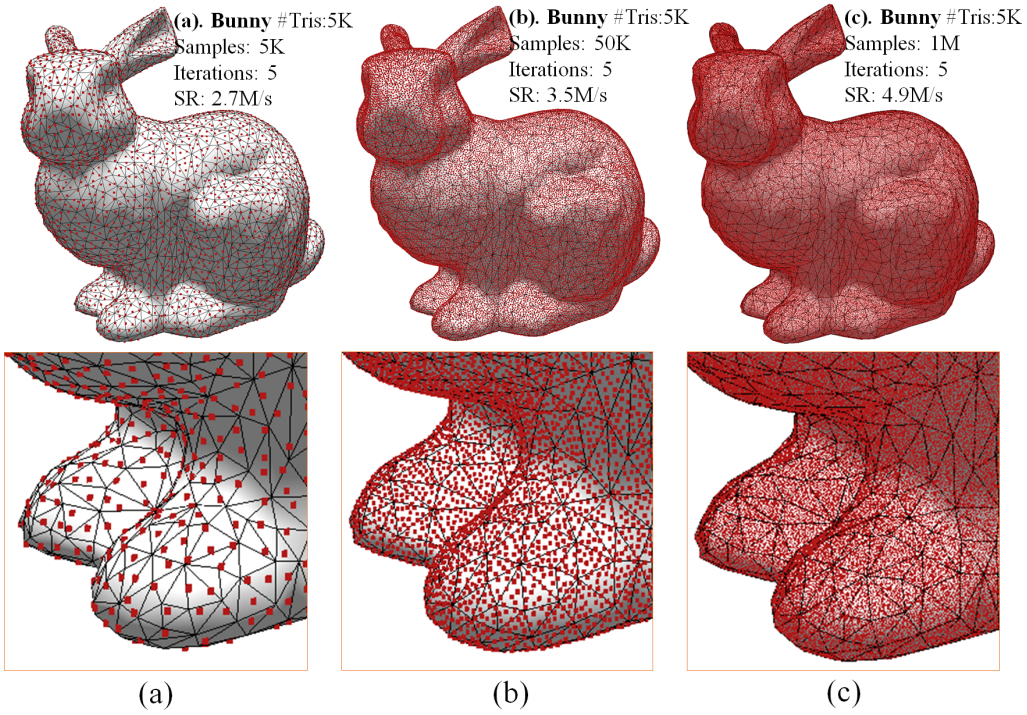


Figure 3.7: Bunny model (courtesy of Stanford 3D Scanning Repository) Poisson-disk sampling results. The sampler can reach a performance up to 11.1M samples/s.

Algorithm 6: Implementation of MergeIntersection

Input : Existed Samples: $\mathcal{S}_{exist}$, Current sample set $\mathcal{S}_{current}$
Output: Updated result sample set $\mathcal{S}_{exist}$

```
1  Function MergeIntersections(S_current, S_exist)
2      ValidityCheck(S_current);
3      Merge(S_exist, Valid(S_current));
4  end
5  Function ValidityCheck(S_current)
6      forall s_i ∈ S_current in Parallel do
7          if IsValid(s_i) then
8              mark s_i as valid, return thread;
9          end
10         mark s_i as invalid, return thread;
11     end
12 end
13 Function IsValid(s_i)
14     if s_i.priority == 3 then
15         mark s_i as valid, return true;
16     end
17     if ∃s ∈RangeSearch(s_i,r), s.priority¿2 then
18         mark s_i as invalid, return false;
19     end
20     if s_i.priority == 1 then
21         forall s_n ∈RangeSearch(s_i,r), s.priority==2 do
22             if IsValid(s_n) then
23                 mark s_i as invalid, return false;
24             end
25         end
26     end
27     mark s_i as valid, return true;
28 end
29 Function Merge(S_exist, S_new)
30     forall s_i ∈ S_new in Parallel do
31         if RangeSearch(s_i,r) in S_exist== ∅ then
32             S_exist.Add(s_i);
33         end
34     end
35 end
```

3.4.1 Quality Evaluations

Spectral Analysis

[Bowers et al., 2010] proposed the first evaluation tool on mesh surfaces that is similar to frequency spectrum analysis on the planer sampling domain, and [Wei and Wang, 2011] generalized the evaluation by gathering statistics of the differential domain of samples. Figure 3.10 compares our results with 50 iterations and the reference by the differential analysis. It's clear that our results preserve the blue noise feature, and have a spectrum similar to the reference.

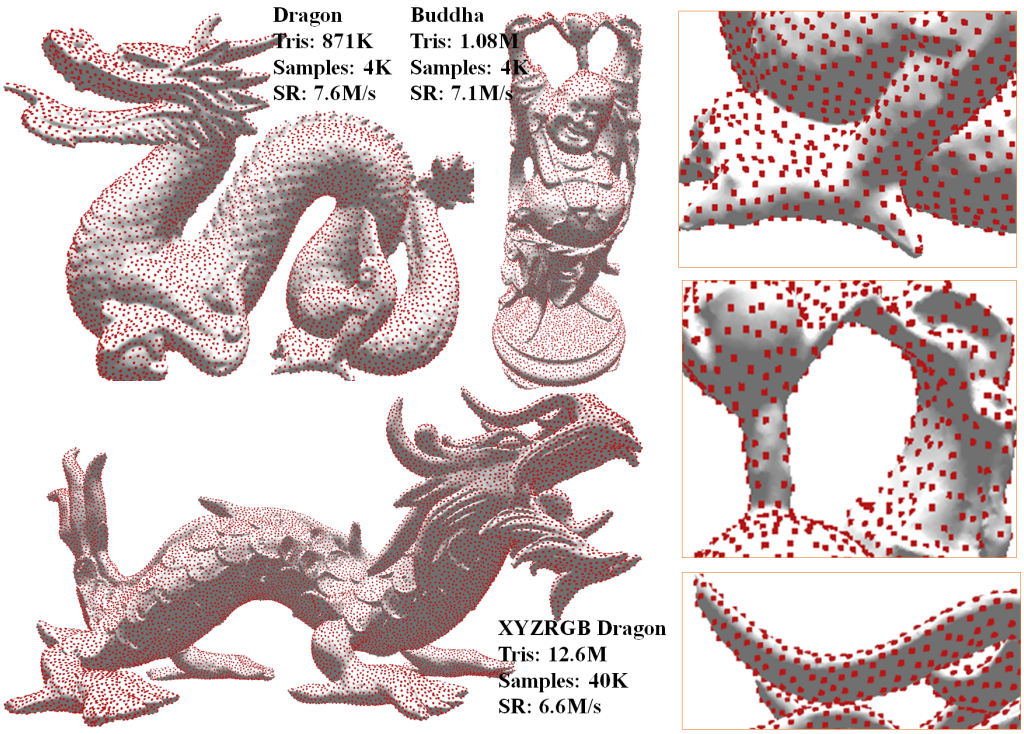


Figure 3.8: Dragon model, Happy Buddha model and Asian Dragon model (courtesy of Stanford 3D Scanning Repository) Poisson-disk sampling results. Notice how the sampling method can deal with extremely complex models with more than 12.6 million triangles all at once, and reach an impressive 6.6 million samples per second sampling rate.

Radius Statistics

We also applied a radius statistic to evaluate our results. The radius statistics are often used to measure how densely a sample set is packed. We used a similar technique as [Lagae and Dutré, 2008] described: A relative radius parameter ρ is a fraction of the maximal packing. The densest packing radius is defined as $r_{max} = \sqrt{\frac{A}{2\sqrt{3}N}}$, where A is the area of sampling domain, and N is the size of sample set. A Poisson-disk sample distribution will have a smaller radius $r = \rho r_{max}$. We will consider a relative ρ parameter, which means that we only consider a fraction of the target Poisson-disk distribution and the parameter ρ of a maximal reference Poisson-disk sample distribution: ρ_{max}. Figure 3.13 shows the relationship between our method's performance and radius statistics. In practice, a sample set with more than 90 percent relative ρ can be qualified to apply in most applications, while a 98 percent sample set shares many similar features with a maximal Poisson-disk sample set.

Figure 3.12 shows how to sample quality will be changed if we apply more rotation iterations of ORBG on the same mesh. For only one iteration we can get enormously high sampling rate on 3D models, with acceptable sampling quality. The quality of the one iteration will be related to how large is the "boundary area" on the mesh surface, which means that on these boundaries, both sides of the ORBG projector can hardly capture their shapes, thus leave additional gaps on those boundaries. Figure 3.11 gives a simple illustration of the places where one-iteration sampling fails to capture effectively. If the boundary space is too large, then a second iteration is needed to cover the boundary area.

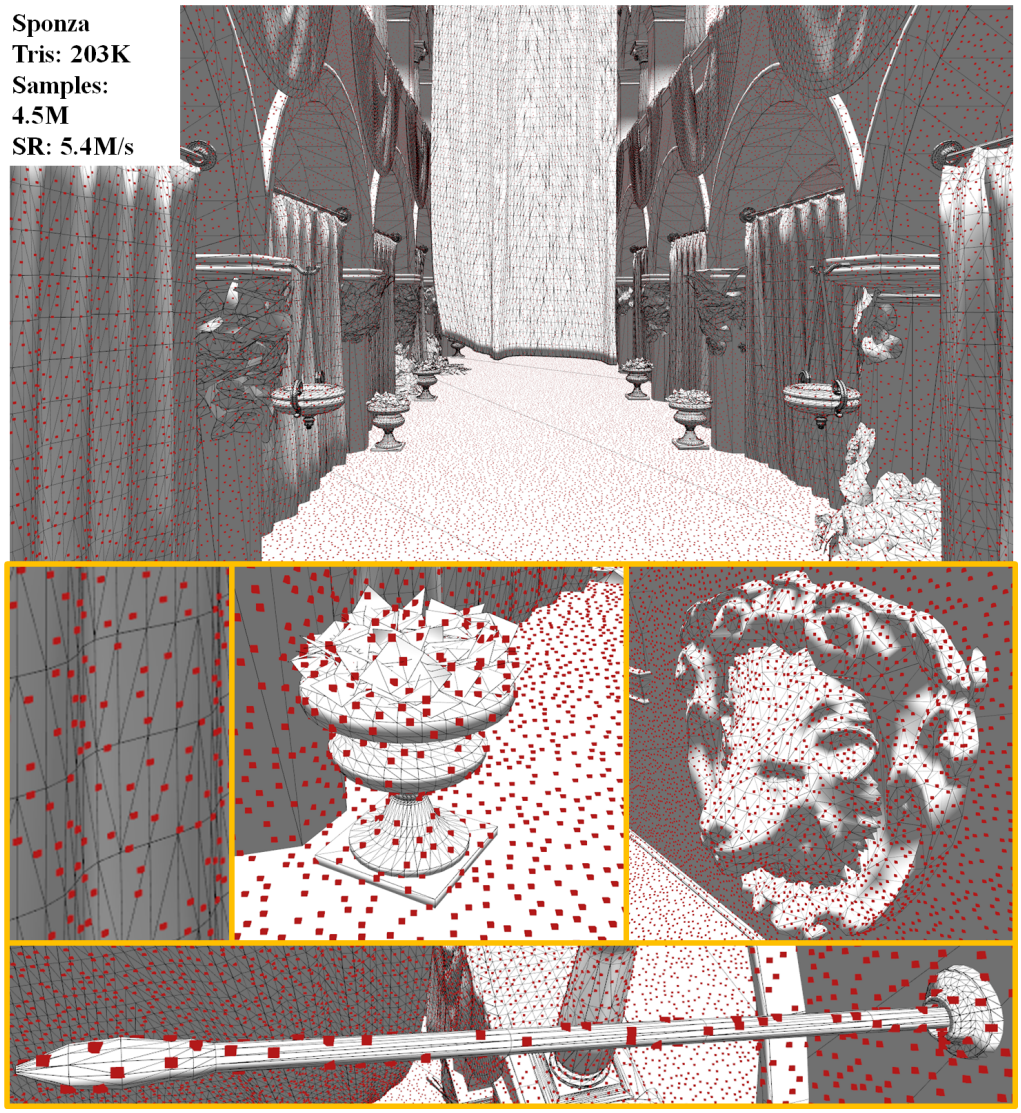


Figure 3.9: The enclosed Sponza scene (courtesy of Frank Meinl and Marko Dabrovic) with our progressive sample projection generation. Note that this method is capable to deal with various complex geometries in the scene, with an effective sampling rate of 5.4M/s on GPU.

Figure 3.13 shows how quality increases with the number of iterations. This means that PSP can generate tens of millions of samples per second for applications that ask for pure speed, while the quality can be continuously increased using extra iterations.

3.4.2 Performance

Our method can produce up to 11.5 million samples per second on the Bunny mesh surfaces on GPU for one iteration with a .0001 Poisson-disk radius. The evaluation of time includes the tile-based 2D pattern construction, ORBG building, ray tracing, intersection merging, and ORBG rotating.

As the kernel initialization in the framework cause processing overhead frequently, the performance will be greatly affected by the batch size. With a larger batch size, the overhead can be alleviated. Figure 3.14 shows how our method is more effective when dealing with a mass amount of samples with small Poisson-

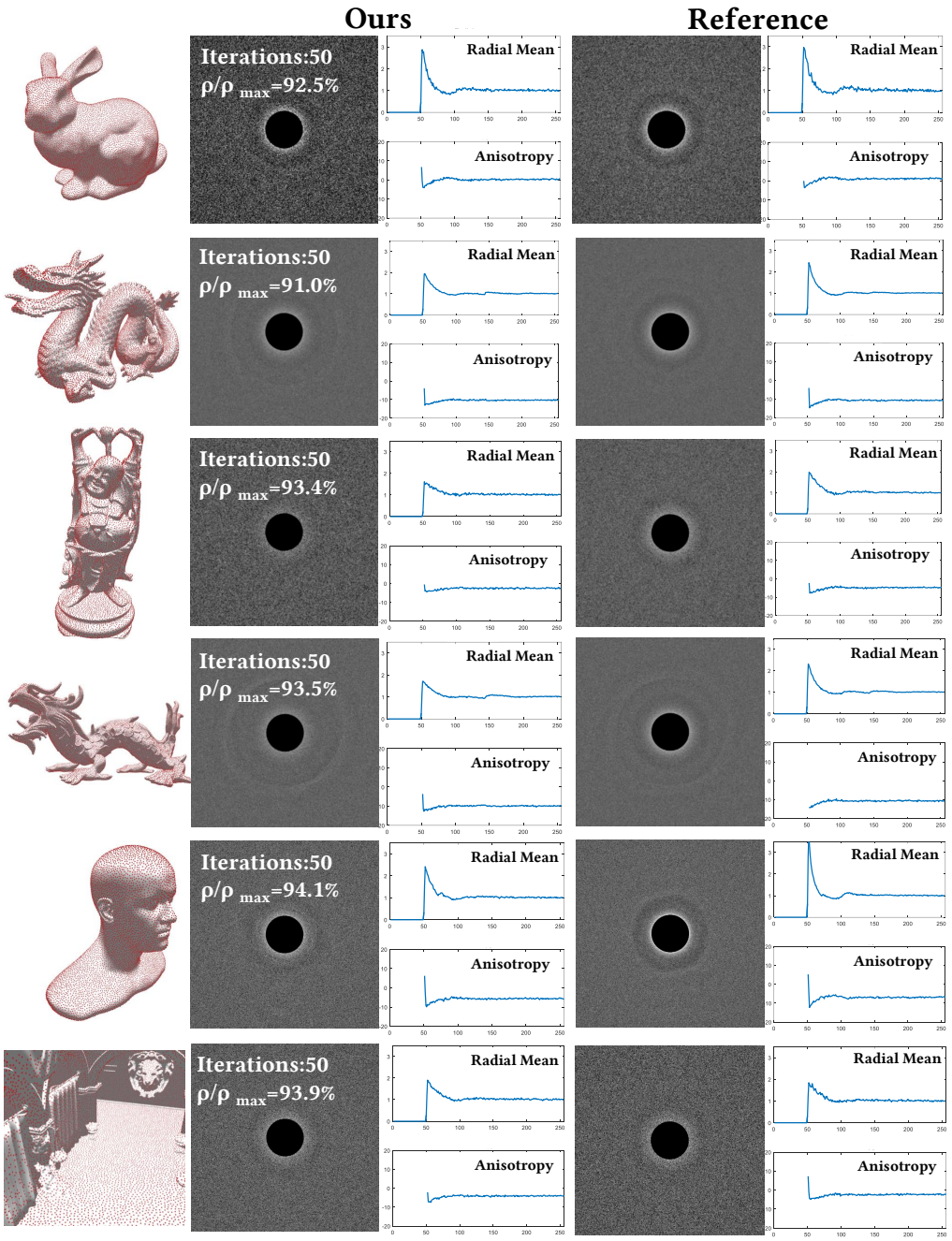


Figure 3.10: Differential domain analysis.

disk sample radius.

Figure 3.15 makes a comparison of time with other methods by tuning each methods' parameter to generate a result of the same quality under the same radius metric. We generate sample radius of 0.00025 on the Bunny model, and our method can be 1.5x to 2.5x faster in most cases concerning fixed radius statistics as a quality metric comparing with other parallelized methods running on GPU, and can easily outperform serial algorithms on CPU by orders of magnitude.

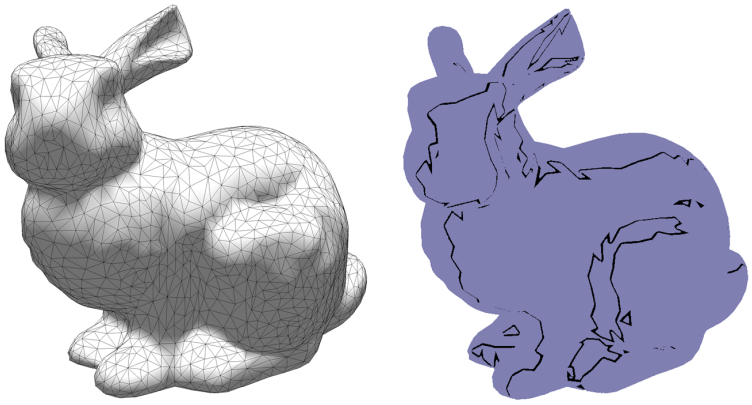


Figure 3.11: The boundary area of a 1-iteration projection. ORBGs are orthogonal to each other, and the ORBG itself is already a mechanic of avoiding gaps and stretches. But they sometimes fail to capture the entire surface when there is some area that is with a near similar angle degree to all surfaces. To eliminate boundary effects, add more iterations by rotating ORBGs randomly. The current boundary will become the other iteration's central area where the projection can act effectively, which will finally cover the entire sample domain.

Intra-Conflict Free

The essence of the proposed method is an effective way of dart-throw parallelization. To throw a batch of darts on mesh surfaces in parallel, the disk of each darts should not intersects each other. In other words, for the dart batch $\mathcal{B}$ that is thrown in parallel, we have the distance of each two points $dist(s_m, s_n) > r, \forall s_m, s_n \in \mathcal{B}$. We call this Intra-Conflict Free (ICF) feature of a dart batch. The generation of ICF batch without conflicts can be a challenge. Our method nailed the generation of ICF batch with the approach of projecting 2D patterns onto 3D meshes, which is why the parallelism of our method is naturally high.

Sample-pool free

Another important note is that our method does not require an initial random point sample set to generate the result samples from. This initial random point set is also sometimes referred as *sample pool* in [Corsini et al., 2012], or *presampling* in [Ying et al., 2013]. We call these methods as *mass-elimination* based methods, as they all need a huge pre-generated set of uniformly distributed samples on the mesh, and do eliminations to get desired sample set.

One serious problem of the mass-elimination based method is that they cannot approach arbitrarily good sample quality with bounded memory. Most of these methods ask for 10x or more pre-generated samples than the result sample set size to get acceptable results. All pre-generated samples have to participate in the spatial data structure for range search, sorting and compaction, thus suffering more from the curse of dimensionality, while restraining the horsepower of parallelization. Our method's memory usage, on the other hand, is bounded by the result sample size (when the requirements of sample storage are significantly larger than the mesh itself of course, which is the most frequent case. Or the memory will be determined by the geometry itself).

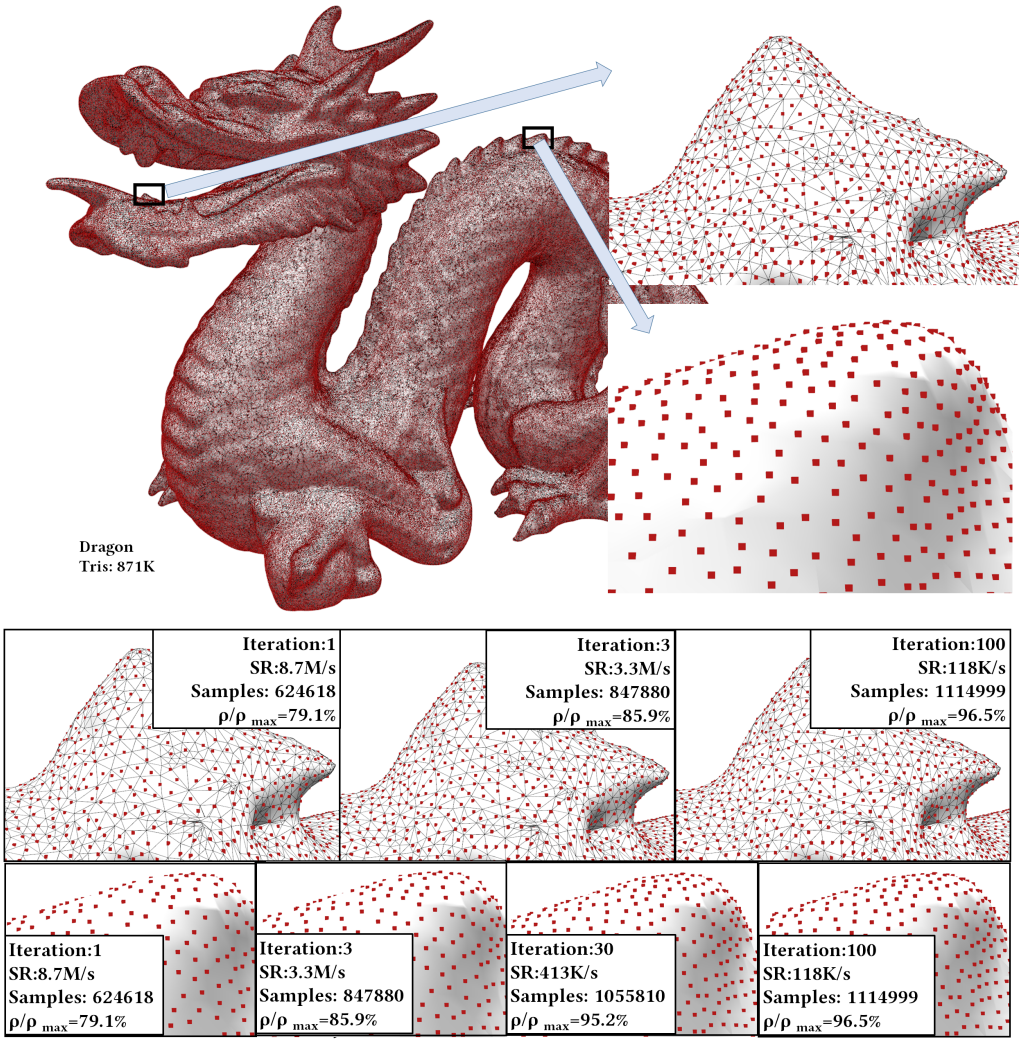


Figure 3.12: Dragon model sampled with a different number of iterations. Few iterations can generate usable samples extremely fast while more iterations can push the quality of sampling towards the limit. Notices that when one iteration is used, that means there are only three projectors projecting samples from the ORBG, thus will leave edges alongside the boundary positions. With the increasing of iterations, gaps will be filled by further parallel samples, and finally, the relative ρ will be approximately the same to a maximal Poisson-disk sample pattern.

3.4.3 Practical Design of Projector Set

Our design of the projector set is a three-view orthographic plane set. One may argue that this design seems arbitrary and empirical. Here we will discuss other possible designs of a projector set, and their trade-offs comparing to our choice. Three parameters of a projector set will be discussed in this section:

Position.

The positions of projectors should be low-discrepancy to ensure randomness and coverage of samples, so practically we will ensure a certain distance of projector centers between iterations by rejecting sampling during projector rotation.

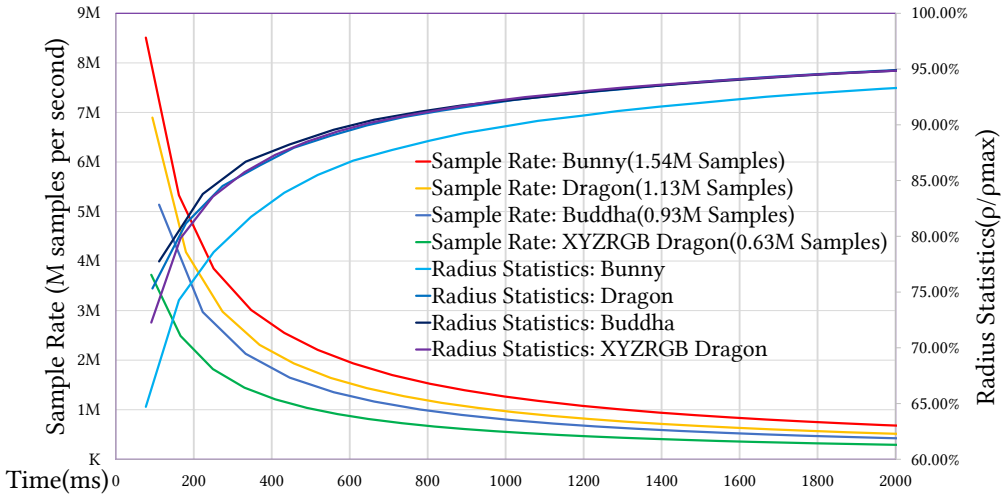


Figure 3.13: Time consumption and quality improvement. Spending more time on different models can get a progressively better quality measured by radius statistics of samples.

Shape.

The major concern of projector design is to ensure high-quality results using fewer iterations while keeping high parallelism. The foundation of our method is to ensure that all samples generated from the same projector have no intra-batch conflict, thus a projector set should consist of 2D planes, each containing conflict-free samples. Samples projected from a non-Euclidean surface cannot ensure this criterion, which will degenerate the merging step to a mass elimination problem that is much more complex to solve. In practice, we'll keep the shape choice of 2D planes for its simplicity.

Size.

The choice of size can be interesting concerning different trade-offs. An arbitrary number of N random positions of the bounding sphere can be chosen to generate a projector. The obvious trade-off in size would be: more projectors can increase parallelism, but each iteration will be more time-consuming.

Things can get tougher when projector set size increased. According to 3.3.2, we assigned a priority value for each batch of samples. As the whole projector batch set will be processed in a fully parallel pattern, more projectors processed in parallel means more priority numbers assigned, which indicates more potential recursive execution for validity check during the compaction step.

Priorities.

When samples have more than three different priorities, recursion will be necessary for validity check during sample elimination. Figure 3.16 shows an example of this. During the compaction step, samples will mark themselves as invalid when conflicting with higher-priority samples.

In 3.16 (a), four samples are generated from four different projectors with different priority numbers. And the process of nested recursion will happen in such an induction order:

- Thread T1 will perform a range search of sample P2 for validity check.

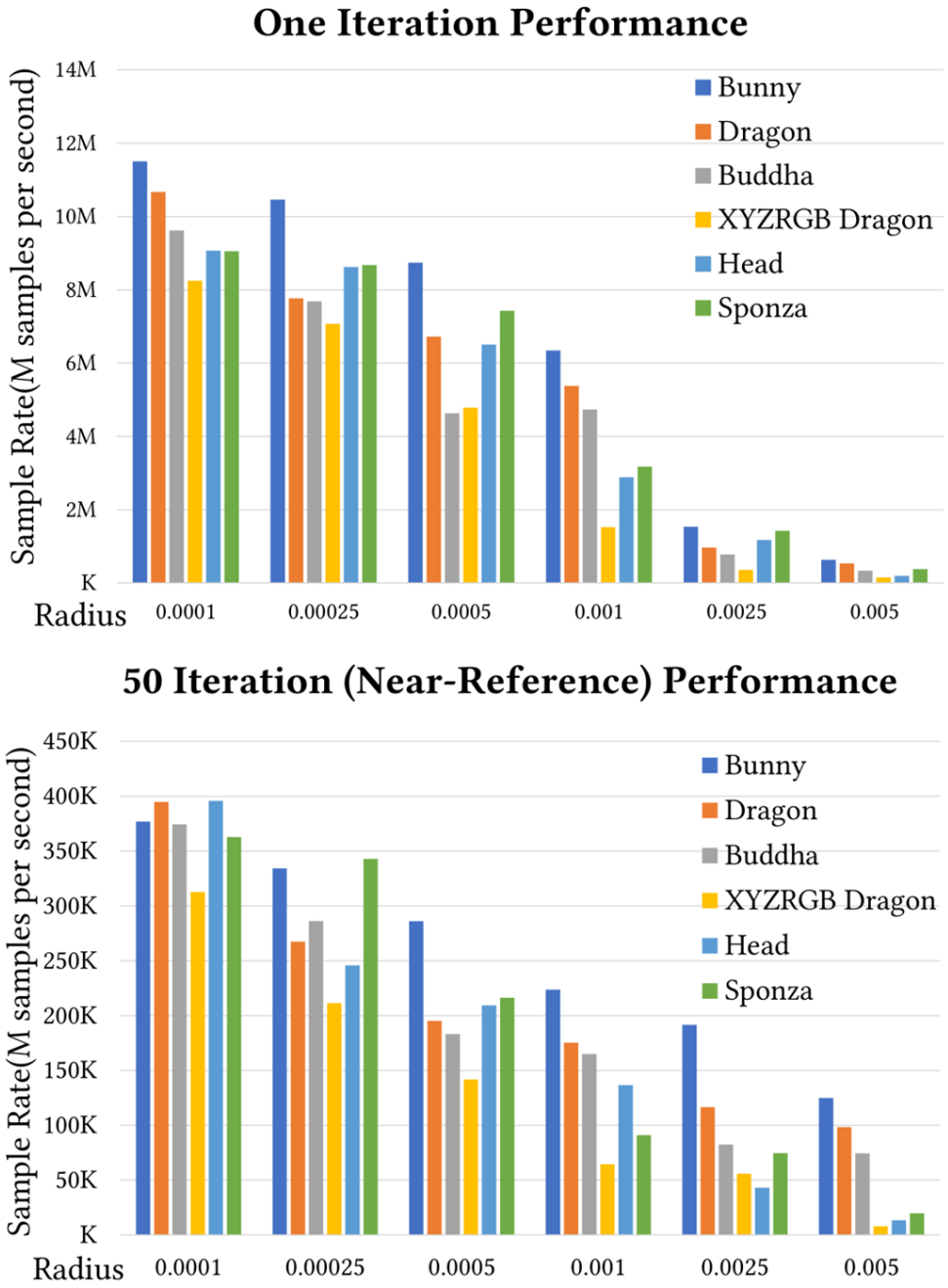


Figure 3.14: Poisson-disk sampling radius and sampling rate performance for samples generated with one iteration and the near-reference samples generated by 50 iterations. Smaller Poisson-disk sample radius means more samples on the mesh, thus applications can evacuate more performance out of our algorithm by generating massive dense samples instead of sparse samples.

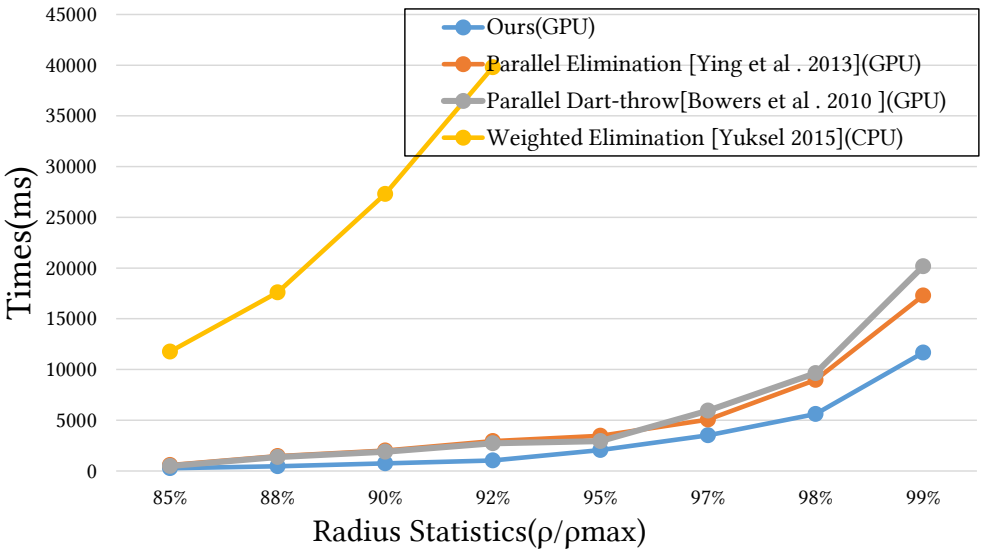


Figure 3.15: Comparison of generation time.

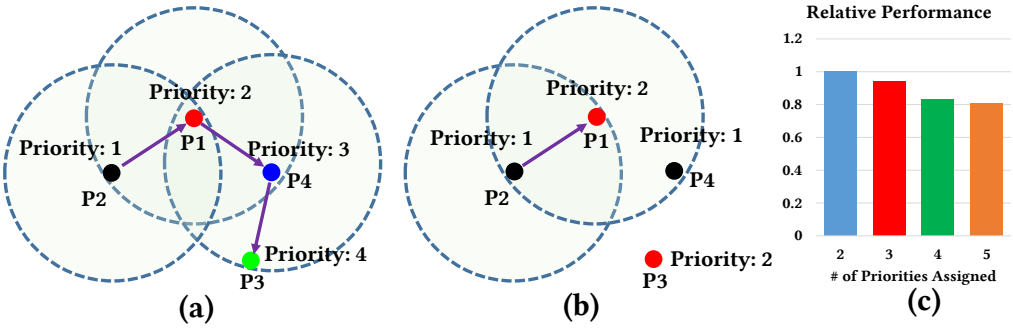


Figure 3.16: Consequences when the size of projector set processed in parallel increased. Take 4 projectors for example. Details explained in 3.4.3

- P1 is of higher priority within the radius of P2, the validity of P1 is unknown to T1.
- Recursively checks P1, eventually goes through P4 and ends at P3.
- Mark P4 as invalid.
- Mark P1 as valid.
- Mark P2 as invalid.

Note that there might be multiple samples involved in each validation step. Figure 3.16 (b) is a similar layout of samples generated by two projectors thus with two unique priorities assigned. The validation of P2 can be done with one range of search. Figure 3.16 (c) shows the relative performance for the same amount of samples generated with different numbers of priorities tested on the Dragon model with 50K samples generated, averaging ten trials for each. This "reaction chain" of conflicts can cause a deep recursion, which will harm the regularization, increase divergence and introduce unbalanced workload for different threads.

Our choice of three projectors perpendicular to each other introduced 3 different priority numbers, which is a decent trade-off selection between parallelism

and regularity in practice. Also, the orthogonal view of planes can provide better coverage of the model, so it is possible to generate usable samples for only one or a few iterations.

An interesting choice would be setting the projector set size to 1, which means that each time we only trace samples from one projector, then merge the results with existing ones. This is sometimes more efficient due to its regularity in some configurations and hardware, but our tests showed that the multi-kernel execution paradigms have more overhead in kernel switching and the reconstruction of the underlying data structures.

Discussion on GPU Execution and Parallelism

The progressive parallel sample projection can reach high parallelism without using priority-based or phase-group based methods. As we discussed during previous chapters that phase-group based methods will reduce parallelism, which is not suitable for massively parallel hardware like GPU to execute, while although priority-based methods increased parallelism to intrinsic parallization, the long recursive chain remains a problem. Besides, the parallelism is on the total samples in the sample pool, which is often 20x or more samples to be processed at once thus being ineffective.

With our progressive sample projection method use parallel ray tracing ([Wald et al., 2014]) to process ray tracing against geometry. The NVIDIA OptiX Prime ([Parker et al., 2010]) parallel ray tracer can reach significantly high performance while tracing rays. As discussed before, we used the restart style ray tracing technique. For CPU ray tracing test, we used Embree parallel ray tracer ([Wald et al., 2014] for its effective usage of Intel CPU's AVX lanes. In 3.4.3 we can observe the performance that the parallel ray tracing can approach on our projection.

Scene	**OptiX Prime ([Parker et al., 2010])**	**Embree ([Wald et al., 2014])**
Bunny(5K)	470MRays/s	45MRays/s
Dragon(871K)	290MRays/s	27MRays/s
Buddha(1.08M)	276MRays/s	28MRays/s
Sponza(203K)	379MRays/s	31MRays/s

Table 3.1: Tested ray tracing performance on a single NVIDIA GTX1080 and Intel i7-4930K on our implementation with OptiX Prime and Embree library.

We will do the so-called **bounding box filtering** to further increase effective parallelism in the algorithm. There will be numerous invalid rays for each batch of rays. Thus for the first generated rays (or original rays) from the ORBGs, the bounding box of the geometry will be projected onto the sample plane to do the first screening of possible sample candidates (before generation of rays), that can reduce the number of possible missing rays. Figure 3.17 shows how the first screening helped to increase the effective parallelism.

The longest elimination priority chain in our algorithm is 3: with the already placed samples as priority 0, and three priority number assigned on each sample. Comparing with [Ying et al., 2013], our parallelism is not intrinsic, but the parallelism is more effective, as the theoretical recursive chain of [Ying et al., 2013] equals to the total number of unique priority number assigned, and in practice, the longest recursion needed to resolve during our algorithm will be above 20. While [Ying et al., 2013] will need more than 20x memory usage for the storage of sample pool, thus the parallelism is not as effective as ours, which also revealed

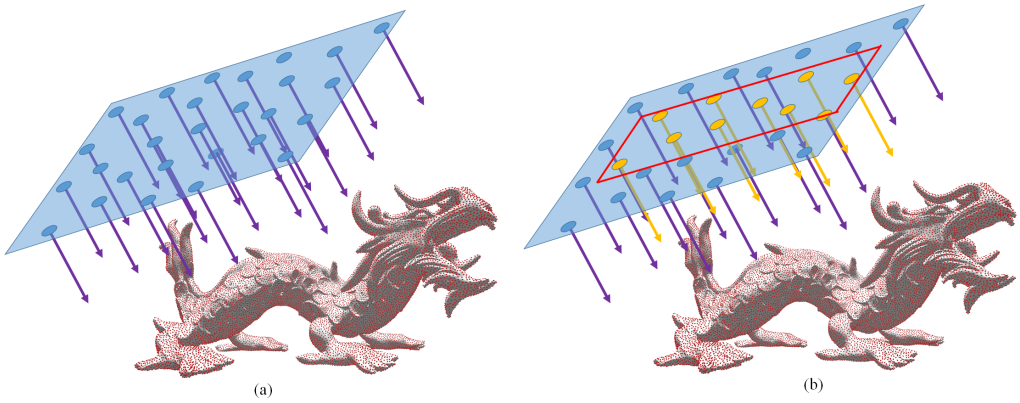


Figure 3.17: Sample plane with out the bounding box filtering in (a). There will be a great percentage of rays generated from the samples miss the object. Though the rays will be terminated early during the processing, it's better to project bounding box of the geometry back with orthogonal view as is showed in (b).

from the performance in 3.15.

Thus our highest parallelism can be reached with our algorithm will be proportional to the valid rays generated from the samples within projected bounding box, with the longest recursive depth of four. We exploiting the effective parallelism from the massively parallel nature of our algorithm, and managed to reach a good performance shown in 3.15.

Rasterization vs. Ray Projection

We will also briefly introduce an equivalent implementation of this algorithm in this section. We used ray tracing as the way to project samples from the projector onto 3D models. As only primary rays and restarted primary rays are used, this process is actually equivalent to rasterizing the models onto the projector, and fetch samples at 2D pattern's position.

It will be easy to implement the rasterization equivalent, but as occluded parts of the model are also needed to be rasterized to get samples on them without being z-buffered out, the process turns out to be irregular, atomic counters will be needed in shaders to keep track of samples. Our rasterization based implementation have similar performance on moderately complex models compared with the ray tracing based projection method, see Figure 3.18. We still choose ray tracing projection as the default method as it has better computational complexity concerning the scale of geometry.

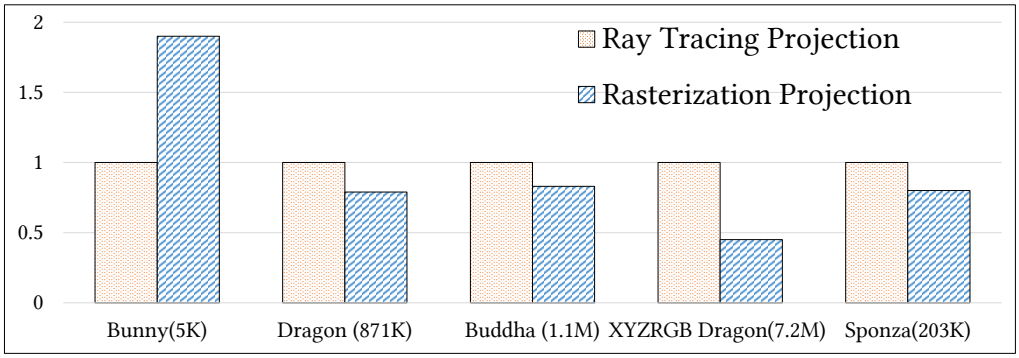


Figure 3.18: Relative performance (sampling rate) on different scenes with a rasterization equivalent implementation of our method.

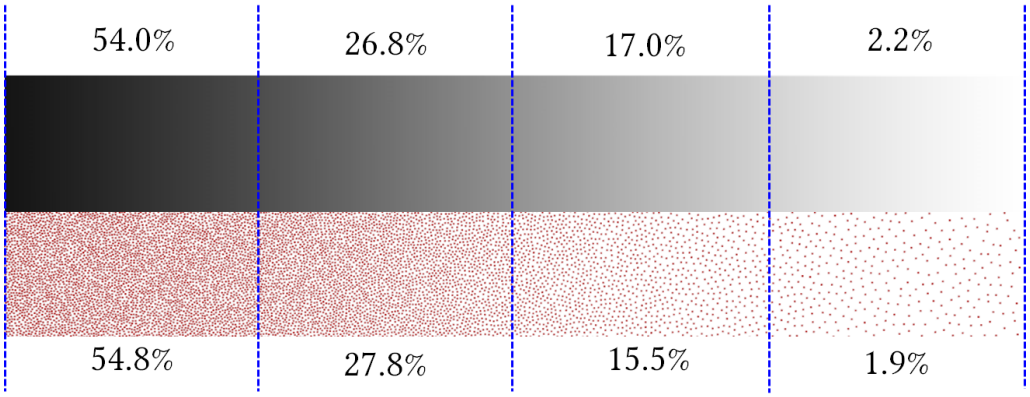


Figure 3.19: Image stippling testing on a density texture. The percentage of samples vs. density maps are near each other, which means the stippling can produce reliable image on morphing densities with various of descending and ascending gradients.

3.4.4 Examples of other applications

The progressive sample projection method is flexible and capable of dealing with various demands from many applications. Besides sampling on 3D models in general, we tested some possibilities on adaptive sampling, fur geometry placement, and degenerated 2D application such as image stippling.

Adaptive sampling

It is quite natural to extend the method to adaptively sampling the underlying model by specifying a radius for each vertex. Figure 3.20 gives a result by applying a simple fading effect on different kinds of 3D models.

Image stippling

Our method can degenerate to 2D in order to be generalized to some 2D applications, which will be useful in some scenarios where the stippling is required to be done inside a 3D uniform framework. Our method can deal with 2D image stippling with good quality and performance. The stippling is done by creating a plane in 3D and specifying the radius of each sample by interpolating a density texture on the plane. Figure 3.21 shows examples. An important benchmark will be stippling the density texture, which is shown in Figure 3.19.

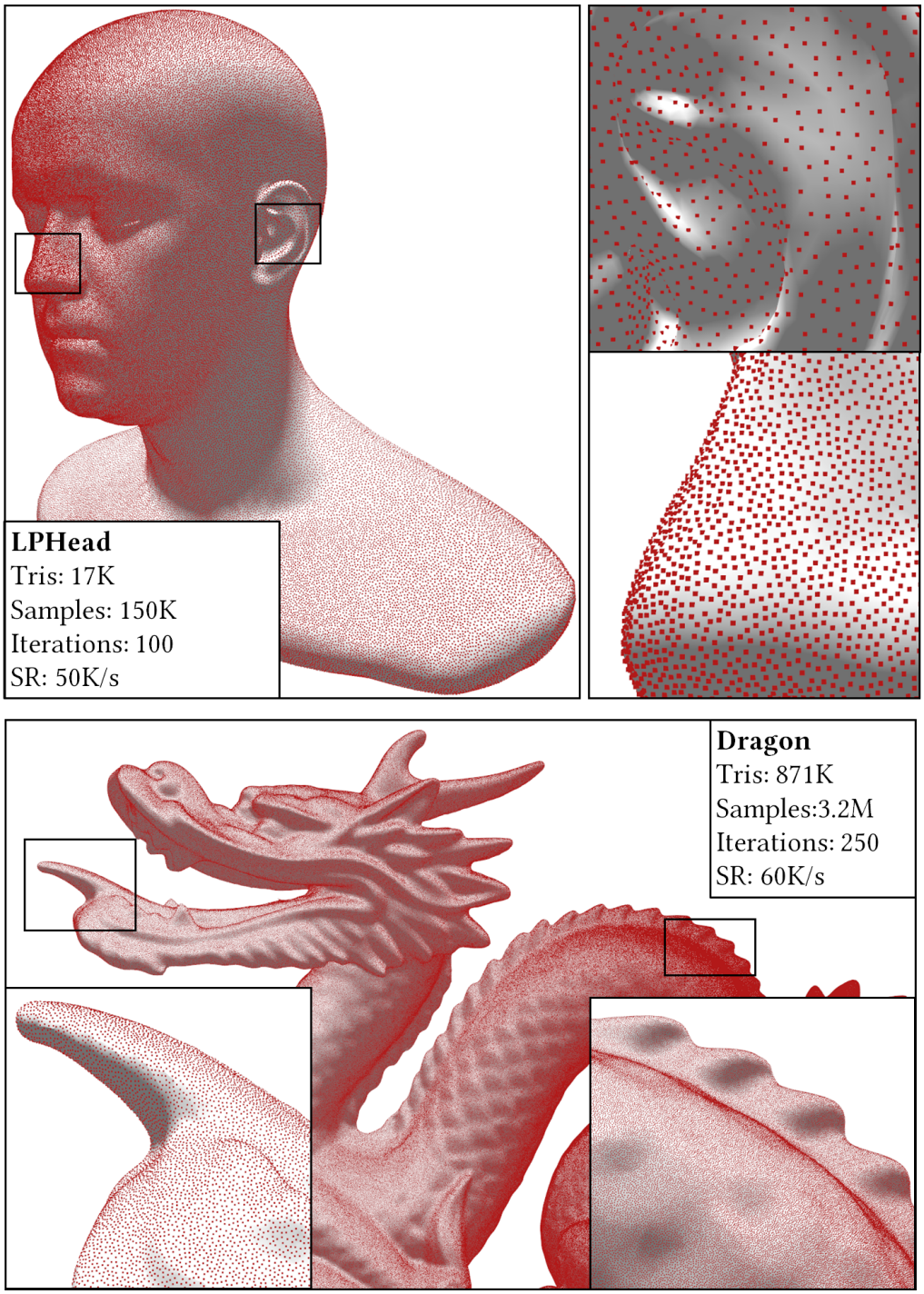


Figure 3.20: Adaptively sampling a fading function. The density map on the mesh surfaces is gradually increased or decrease.

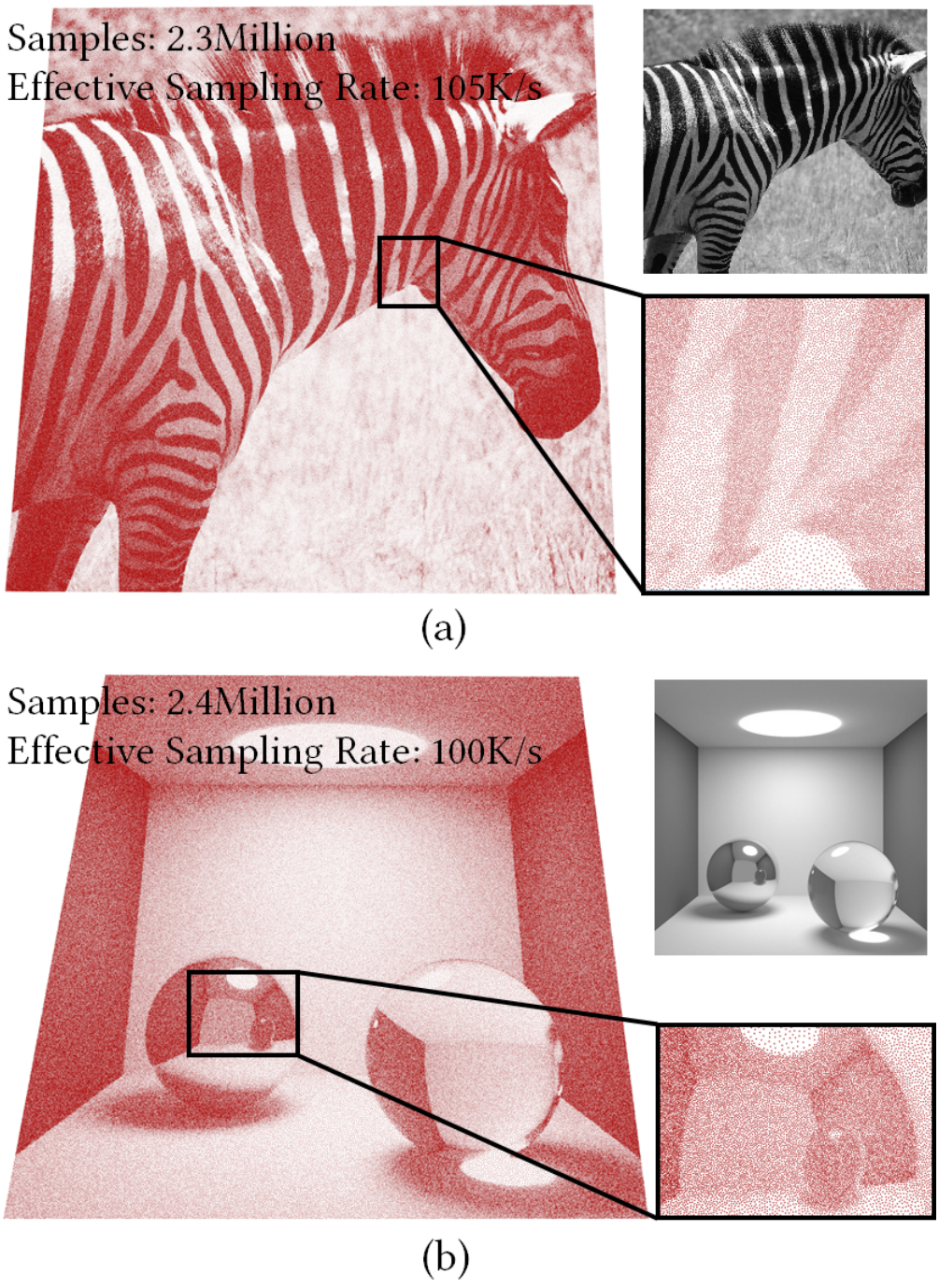


Figure 3.21: Image stippling of a Zebra image and a rendered results of a Cornell Box. The slight tilting of the image shows that the image is actually a texture on two triangles in 3D space, which indicates that the image stippling itself is a special case for our progressive sample projection to deal with.

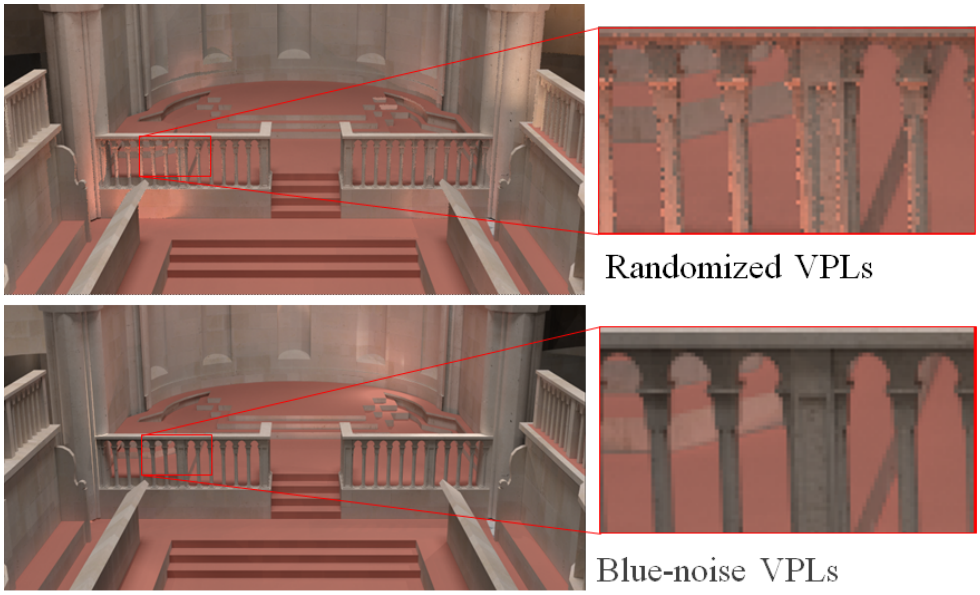


Figure 3.22: Blue-noise VPLs generated by rejection sampling. Both image rendered with 2048 VPLs spreaded across the scene. We generate the blue-noise stationary site points with our progressive projection based sampling, and the results shows that it can alleviate some singularity problem, though not by a significant deal.

Blue-noise VPLs

Virtual Point Light is an important image synthesis method that treat each illumination vertex as a virtual point light that can enlighten the scene. For detailed introduction of such a rendering technique, reader may refer to [Dachsbacher et al., 2014]. We use a technique to guide the generation of virtual point lights in the scene to only generate VPLs on the pre-sampled blue-noise stationary pattern called "site points" in this way with rejection sampling. The rendered results appear to have better illumination condition than the randomly generate VPL, which have strong artifacts caused by false singularity. See Figure 3.22 for more details.

Fur geometry placement

The fast generation with only a few iterations can be suitable for many applications. Figure 3.23 gives a result of placing fur geometry on object.

3.4.5 Limitations: Recap the Previous Work

We emphasize on the advantages in performance and memory footprint that our technique brings, but still, some obvious limitations are comparing with the previous work. They will be discussed in this sub-section.

Overall limitations

There is an obvious limitation that our sampling strategy will need the proper 2D sample pattern to start with, which means some pre-processing steps will be needed for the generation of such a pattern. One can store smaller pattern and use KDRT described in Chapter 2 to reduce the storage: the hard-coded

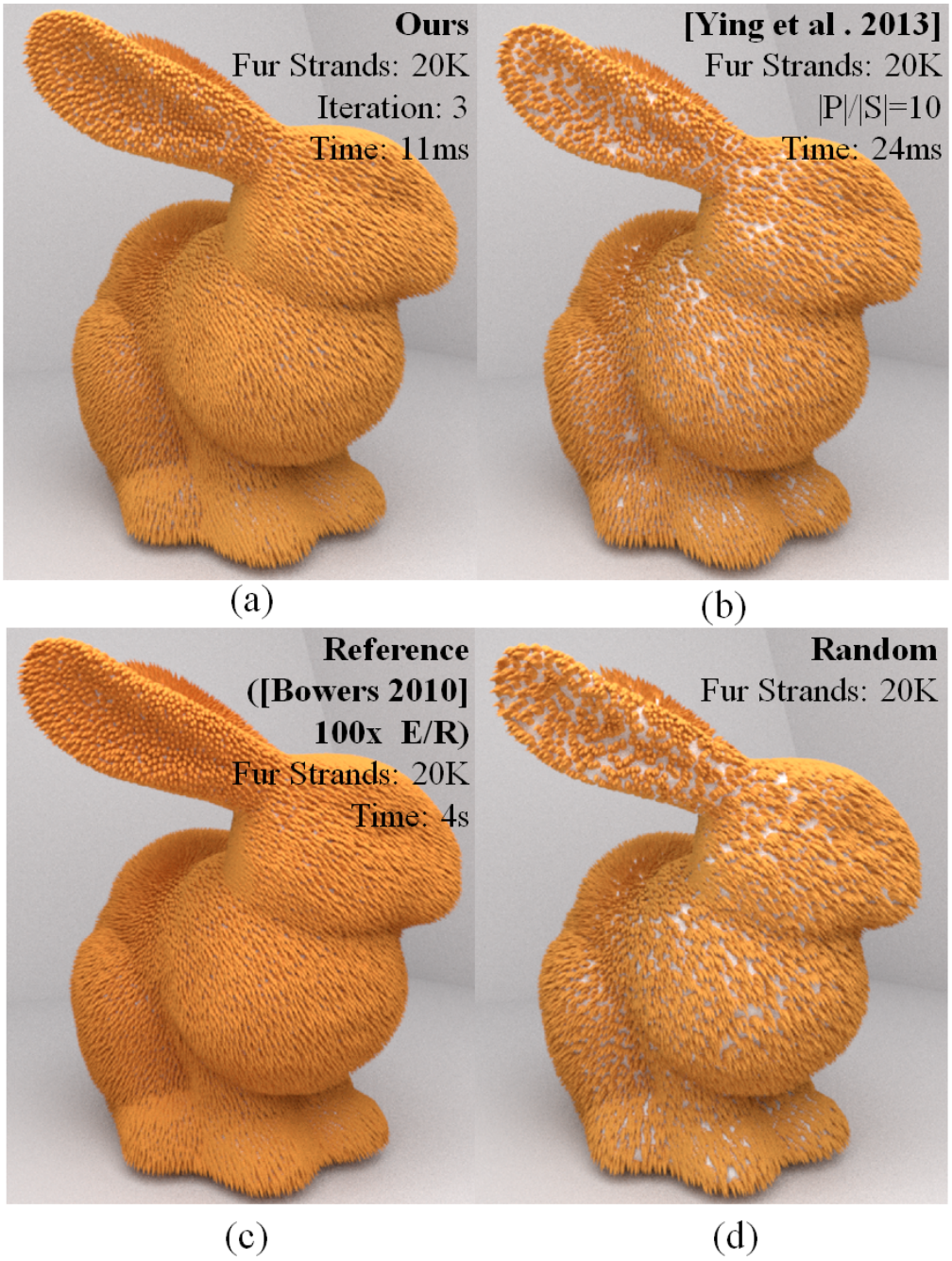


Figure 3.23: Fur geometry placement on 3D object. Our method can approach a near reference quality with only 3 iterations.

small pattern can be used as the tiling pattern in KDRT to tile the 2D sampling pattern we need in PSP. In this way, the storage space, as well as calculation consumption, will be reduced significantly. But the pre-processing step will still be a part of the algorithm, which makes the PSP technique heavy: that it's better to be used when there are a massive amount of samples will be generated, instead of only a few.

Another limitation of the algorithm is the fact that it could be hard to determine the number of iterations needed to get results that meet demands. We can provide an empirical number of 30 iterations to reach a good sampling quality, but as previous sections discussed that it is heavily scene-dependent. We did not

provide an intrinsic way to

Comparing with elimination with phase-group based parallelization ([Bowers et al., 2010])

As this is a method based on elimination from sample pool, thus the whole process will be finished without iterations. It will need less pre-processing, and the whole process will be less heavier without introducing a parallel ray tracer in the implementation.

Comparing with elimination with priority-based parallelization ([Ying et al., 2013])

Their strategy can process all samples at the same time, so the parallelism will be higher than ours. Though the parallelism will not be as effective as the priority-chain reaction reflected as recursive chain execution for some threads. But still, there will be times that the priority chain being not as serious. With those situations that their methods can reach a high performance as well. But empirically this will rarely happen. Another concern would be that some modern GPU architecture will prefer parallelism than workload balance. In such cases, it could be possible that this algorithm can also have high effectiveness. Whether the effectiveness will be higher than our strategy on such GPU architecture can be a direct future work to test.

3.4.6 Future Work

As we described in the previous section that one obvious future work that we didn't do using this technique is multi-class sampling. To do multi-class sampling, we can use k set of ORBGs (k being the number of set) to do the projection, and increase the priority number to keep the correctness in the elimination. It can be done with some quite minor modification of the algorithm.

3.5 Conclusion

In this chapter, we present *progressive sample projection*, a new parallelization paradigm of dart-throwing: Project samples from orthogonal 2D planes onto the surface, then rotate the plane for more iterations to get high-quality result progressively. Our method can generate usable Poisson-disk samples extremely fast with few iterations, while can also be used to produce high-quality samples with an adequate amount of iterations. Our method can outperform previous methods in terms of generation speed under the same quality metric, with memory usage bounded. We believe that many interactive applications can benefit from the proposed method.

Chapter 4

Efficient Parallelism in Distance-Constrained Relaxation

After a decent Poisson-disk sampling pattern is created using methods introduced before, the samples will often need to go through a relaxation step to be optimized for applications demanding much more strict quality metrics. These applications include remeshing ([Yan and Wonka, 2013]) and high quality stippling ([Balzer et al., 2009]).

Also, it is important to generate Blue Noise samples with a user-specified state. For example, the power spectrum can be customized, the final number of samples should be able to specify, and a quality metric can be pre-defined as well. Relaxation-based methods can better satisfy these requirements: the power spectrum of samples can be modeled as an optimization problem of a function defined on each sample points (see [Wachtel et al., 2014b],[Zhou et al., 2012]).

In this chapter, we are going to introduce an efficient way that can properly accelerate and strengthen a relaxation method based on a simple yet powerful constraint energy function: a constraint-based on an averaged distance between samples, to generate high-quality blue noise samples with fully parallelism to ensure sampling rate. This method can be extended naturally to multi-class blue noise sampling, as well as adaptive sampling. The previous is crucial in applications such as object placements, while the second makes the fundamental of most image stippling algorithm.

We will also introduce the activeness based relaxation, which will utilize second-order information defined on each point as activeness to dynamically re-distribute tension forces between each pair. This new approach can help exploit full parallelism from the distance-constrained relaxation method, while it can also help the sample set to converge to a better equilibrium to help avoid local balance.

We will first introduce how other related relaxation-based ideas evolved and developed, then introduce the idea and algorithm on efficient parallelism for radius-constraint blue noise sampling. Then we'll test the idea in some related applications

4.1 Relaxation Based-Blue Noise Sampling: Introduction

In this section, we'll give an introduction to relaxation-based blue-noise sampling methods. Methods introduced here may not directly be used for improvements or comparison in the following sections but can help to give readers a more comprehensive understanding of the idea.

A natural way to consider blue noise sampling is to morph a previously existed sample pattern to fit the power spectrum for specific usage. The original sample set can be in a rough state of blue-noise, or even completely randomly generated in the domain. This set of methods will then move each sample to minimize an energy function defined on each point that represents the final state of requirements, or set a repulsive or absorptive force rule for each sample to get an equilibrium state, or simply set a rule to separate samples apart so they would appear blue-noise from a general perspective.

The first group of methods will be named as **Optimization methods**, the second group of methods is **Equilibrium methods**, and the third group will be called **Separation methods**. In the rest of the section, we will discuss selected state-of-the-arts methods in these categories as an introduction to the related literature.

4.1.1 Optimization Methods

Though blue noise sampling seems to be a straight forward sampling setting, to obtain a high-quality sample pattern is not trivial. It will take much more efforts than imagined. If the toleration of sampling rate is high, which means the hardware can have a good amount of time to generate a high definition setting of blue-noise, then the family of **optimization methods** serve as a good approach.

Classic approaches in this category work like this: they often define an energy function on each point, which will be based on a first or second-order characteristic of the point cloud, often came from a predefined blue-noise profile. Then all points will be moved based on a simple gradient descent or more complex methods to another position, where the energy function value being optimized. The iterations will go on until a convergence condition is met.

The most famous classic methods belong to this category is centroidal Voronoi tessellations (CVT) based method [Du et al., 1999]. To optimize based on the CVT often start with a random sample distribution, define an energy function on each point, then move samples to optimize the energy function to finally achieve blue-noise results. The centroidal Voronoi tessellations define the energy function related to its Voronoi centroid. The energy function is often formulated as

$$E_{CVT}(X) = \sum_{n=1}^{N} \int_{V_i} \rho(\boldsymbol{x}) \left\| \boldsymbol{x} - \boldsymbol{x_i} \right\|^2 d\boldsymbol{x} \tag{4.1}$$

where $V_{i=1}^{n}$ being the Voronoi diagram and $\rho(\boldsymbol{x})$ is the density function defining the underlying sampling domain. A famous algorithm to minimize E_{CVT} is Lloyd's algorithm. Figure 4.1 illustrated how this algorithm works. The total energy will be minimized if sample x_i kept moving to the centroid of the Voronoi region [Lévy and Liu, 2010]. This method can generate blue-noise sample pattern efficiently comparing with other more complex optimization-based methods, but as we can get from Figure 4.1 that the regularity of generated pattern is not suitable for many usages that need more alias control.

To solve the regularity artifacts caused by the strict constraint of Lloyd's algorithm (the constraint of points position being the centroid of Voronoi region),

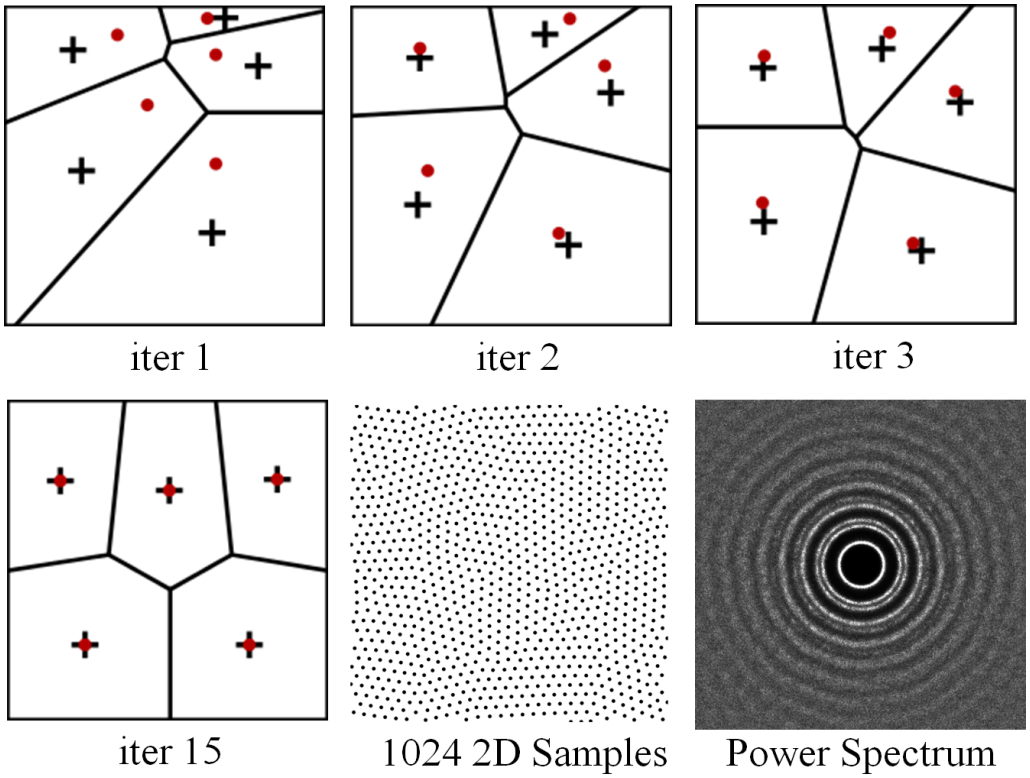


Figure 4.1: Optimization based method[Du et al., 1999], Lloyd's Algorithm as an example.

[Balzer et al., 2009] proposed a much loose constraint on Voronoi region's capacity for each sample. The capacity-constrained point distribution requires the capacity, or the effective area (area combining underlying density function) being the same for each sample. The energy function can be formulated as

$$E_{CapVT}(X) = \sum_{n=1}^{N} (\int_{V_i} \rho(\boldsymbol{x}) d\boldsymbol{x})^2 \tag{4.2}$$

It turns out that using capacity constraint can greatly reduce regularity artifacts caused by Lloyd's algorithm. Balzer proposes CCVT method (capacity-constrained Voronoi tessellation) in [Balzer et al., 2009] a discretize optimization method that with complexity at least of $O(n^2)$ which is not acceptable when considering the sampling rate. He proposed a downhill simplex algorithm in [Balzer, 2009], but it is still inefficient for applications with large scale as the convergence is difficult to reach [Balzer, 2009].

Various improvements over CCVT has proposed afterward. [Xu et al., 2011] proposed capacity-constrained Delaunay triangulation to address complex problems in CCVT by modifying on Delaunay triangulations of samples instead of a discretized Voronoi diagram. [Chen et al., 2012] combined the centroidal Voronoi tessellation with CCVT and proved the combined function being differentiable on the sampling domain, then used a second-order optimization algorithm based on L-BFGS [Nocedal and Wright, 2006].

Besides using CVT and CapCVT energy functions, While other optimization-based algorithm also try to define a proper optimization on functions to satisfy two important criteria: (1)Samples should appear blue noise and (2)samples should not appear to be too regular. One famous approach is proposed

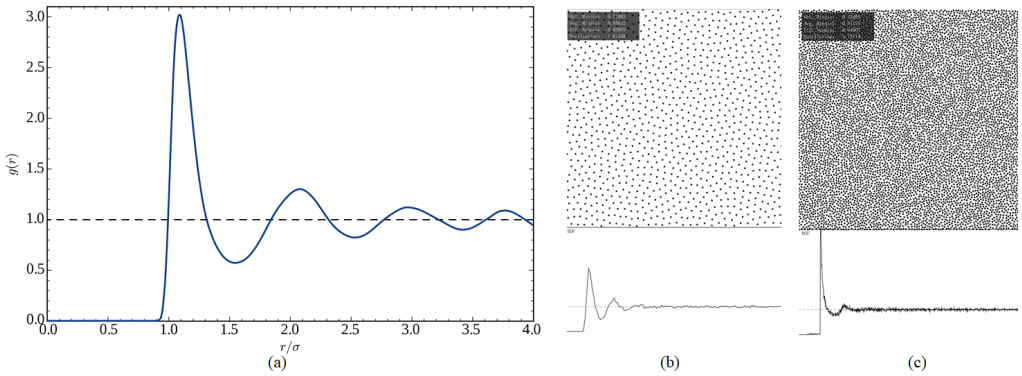


Figure 4.2: Pair correlation function (PCF). (a). PCF from a typical Lennard-Jones model fluid [Jones, 1924]. (b). PCF of a 2D CCVT point set. (c) PCF of a 2D maximal Poisson-disk point set.

by [De Goes et al., 2012] that formulate the problem to be an optimal transport problem, and use a continuous algorithm to optimize the optimal transport function. They also use a power diagram instead of the previously used simple Voronoi diagram.

Another group of methods focuses on how to adopt the state of power spectra of sample sets to a specified profile. Pioneer research is done by Zhou et al in [Zhou et al., 2012]. They try to model the problem as calculating current differential domain statistics from the target function and applying gradient-domain to morph each point so that their characteristics would match more and more towards a final power spectrum state. A group of methods makes more advance in this direction. [Öztireli and Gross, 2012] try to utilize a concept called **pair correlation function** (or radial distribution function) to describe sample set statistics. From [Illian et al., 2008] we know that a second-order characteristic is recommended to describe and represent the distribution of a stationary point process, among which the pair correlation function can provide the most information. This is a function that describes how density varies, as a function of distance from a particular reference particle, then make an average of them all, see Figure 4.2 for some examples of PCFs.

4.1.2 Equilibrium Methods

Generally speaking, this group of methods is also an optimization-based method. The difference is that this group of methods define a repulsive or absorptive force analogy on each point and run the iteration to get a final equilibrium state. As most of these methods behave quite differently from typical methods that try to minimize an Energy function, so we put a separate section to introduce this category of relaxation-based methods to generate blue-noise samples.

It is a natural intuition that blue-noise samples appear to be particles that attract or repel each other, so modeling them as physics particles as an analogy came to be an obvious insight. A pioneer work [Schmaltz et al., 2010] first try to make all repelling forces between equally charged particles to create a homogeneous distribution that best suit a blue noise profile for the application of halftoning. Apart from a global approach that tries to model a much strict electrostatic model on each sample, [Jiang et al., 2015] also tackling this approach by inspiration from Smoothed Particle Hydrodynamics. Particles going through SPH appear to have a high-quality blue-noise spectrum, thus they tailored the

SPH method just for blue-noise sampling and managed to increased the performance by a significant scale. Also, another N-body modeling is proposed in [Wong and Wong, 2017] that treat samples as electrically charged particles to minimize the electrostatic force that each experienced.

The PCF statistics mentioned above can provide a repulsion analogy from an integration of the PCF function, as stated in [Heck et al., 2013]. They utilized PCF to create a proper definition of repulsion around each point by taking integration of the PCF function.

$$F_i = \sum_{j \neq i} f(\|x_i - x_j\|) \frac{x_i - x_j}{\|x_i - x_j\|^2} \tag{4.3}$$

$$f(r) = \int_0^r g(x)dx - \int_0^r g_t(x)dx \tag{4.4}$$

That the force of each point i will be determined by the integration of PCF $g(r)$ by measuring its distance from the target profile $g_t(x)$. It's clear that the integration $G(r)$ just measures the average point density within radius r, which provide a detection of sample density around point i and if it is close to the target $G(r)$ accordingly.

Forces that are defined on different criteria and conditions can all benefit from an iterative scheme like this. But the demands they share in common is how to quickly iterate the current sample set to a final stationary state, which is the equilibrium state that satisfies blue-noise as well as a non-regularity criterion. Our research will focus mostly on how a quick iteration can be obtained when trying to reach the final equilibrium state.

4.1.3 Separation Methods

Separation methods tend to separate samples that are too close together apart. While this action is kept continuously, the final state of the sample set will appear blue-noise. The category of separation idea came from the [Schlömer et al., 2011]. This is a very simple method and easy to understand. The outline of the algorithm is that every time pick a sample from the current iteration, delete it from the sample set, then re-insert it into the set, but this time only insert the sample to the largest gap in the whole sample set. It can be proved that the minimum distance from one another will be maximized by this process.

If the iteration of optimization methods or equilibrium methods is too aggressive that each sample will be moved along step size to satisfy the blue-noise feature directly, rather than using small step size to gradually approximate the final states, then it can be equivalently considered a separation algorithm already. The most famous method will be the push-pull algorithm [Ahmed et al., 2016]. It is a serial algorithm that takes three constraint condition into account for each sample point: Each point will be in charge to move its neighbors. It will move near-by neighbor points beyond conflict radius, move neighbors that are far away towards the sample itself to minimize coverage radius while maintaining a tracking on the constraint on Voronoi diagram's capacity to reach a blue-noise profile similar to what's revealed in CapCVT. It also has a better

But the problem with the push-pull algorithm is that it is hard to be made parallel. The author stated in [Ahmed et al., 2016] that empirically a parallel version of the push-pull algorithm is hard to converge and is slow overall. Here also lies the problem of how to achieve an efficient parallel paradigm to get the full sampling rate out of this method. In the following section, we'll give an

introduction of a core problem that alliterative method should manage to solve when executed in parallel, how some other methods managed to tackle it, and how our algorithm alleviates the canceling out the problem by using novel second-order information defined on each sample.

4.2 Distance Constrained Relaxation: An Overview

Previous methods focused on much more complex constraint which they hope could get the results to appear more blue-noise feature while maintaining random. But with our observation, what actually conduct the sample pattern into is actually determined by the initial state of sample set before relaxation. Even a complex optimization based method can fall into many problems during the journey towards convergence.

4.2.1 Problem of Iterative Algorithm with Per-sample Constraint in Sample Relaxation

The current process for the most iterative algorithm is finding a constraint, then focus on meeting the constraint over the global point cloud through a long iteration. We will state later in this chapter that even with a very simple constraint such like the distance between samples, high-quality blue-noise samples can be obtained through carefully designed iterations and performance can be exploited through our efficient parallelism of this iterative scheme. The simple distance constrained blue-noise sampling can generate high-quality blue-noise samples fast, kept the iterations low, while preserving a good quality over many other methods comparing radius statistic and quality of the final application.

While doing parallel iterative computations on each point, traditional methods constraining a certain energy function often stuck on a local minimum, or stay at a false equilibrium constraint tension for a long time of iteration. Mostly the reason is caused by three reasons as follows:

Propagation Speed of Tension

Tension is an analogy represent the differentials calculated on each sample in the downhill style optimization. Figure 4.3 shows that in such a relaxation scheme, tensions will propagate and relaxed during the iteration. The relaxation method takes in an input of uniformly generated samples or jittered grid samples, calculate energy function on each sample, propagate tensions to each neighbor, then repeat the iteration to get final results. The propagation speed of the differentials of energy function determined the speed of how fast the sample set can be converged. The step size λ provide a straightforward way to adjust a propagation speed, but a large step size is not well-defined when all tensions are computed in parallel and added to samples simultaneously. With a conservative step size however, the convergence can take really long, and the tensions are not evenly distributed across the whole sample plane.

Local Balance and Canceling

Besides the most general propagation speed issue, a much more serious problem is the local balance or local equilibrium problem. Sometimes a sample have clear tension with its neighbors calculated by the energy differential, but when it is doing parallel computation, all tensions around the points canceling each other out, leaving only a slight differential vector being valid for the current processed

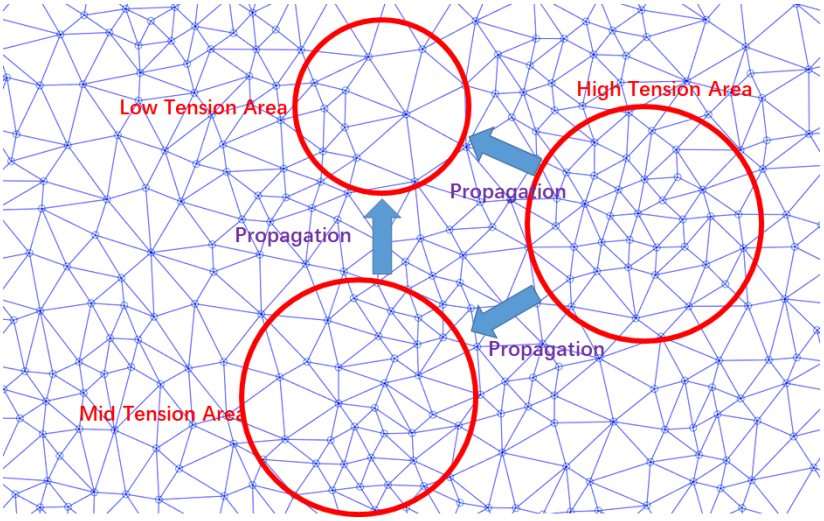


Figure 4.3: The speed of tension propagation on the point cloud (which is being Delaunay triangulated in the figure) is the key to achieve convergence.

sample: though hopes are that the local equilibrium can be solved over time during iterations, but combining the previous problem, the convergence could be extremely slow for the point cloud to realize the problem by itself. What's worse is that chances are that some local balance may never be resolved by passing simple first order energy differentials.

In Figure 4.4 we can see that even tensions have been propagated to the incident sample, the effective displacement vector calculated through this process might be extremely small, which will for sure harm the convergence speed.

It is obvious that the problem of local balance may be alleviated or aggravated by choosing which entity to move. For example, when the tensions that cause by not satisfying the constraint existed but no effective displacement vector can be effectively generated, we can inverse the problem from *moving the current entity* to *moving the neighbors*. This will be a fundamental inspiration of our parallel method to solve the problem efficiently. Notice that if we choose a symmetric way to decide which sample to be the dominant one in charge of the moving, the second strategy might still cause another canceling-out at the neighbor's position, thus being ineffective. We will later present a method to provide a non-symmetric way to distribute the dominance of samples activity by better utilizing second-order information gathered during the iterative sampling process.

Oscillation

Finally, there might be another issue that prevents our constraint-based method from fast convergence, which is oscillation. Oscillation often happened when a local equilibrium has several stationary sub-state, and the iteration step is relatively high. See figure 4.5 for an example.

This will appear less often than the previous two categories, but when such state is entered, it will need a long time before the unstable state is finally reached, and all samples begin to find their correct new position. To alleviate this problem, we propose a varying step size for each sample calculated accordingly by their tension history to avoid oscillation around the same spot. The following section will provide more details on how these problems are being resolved.

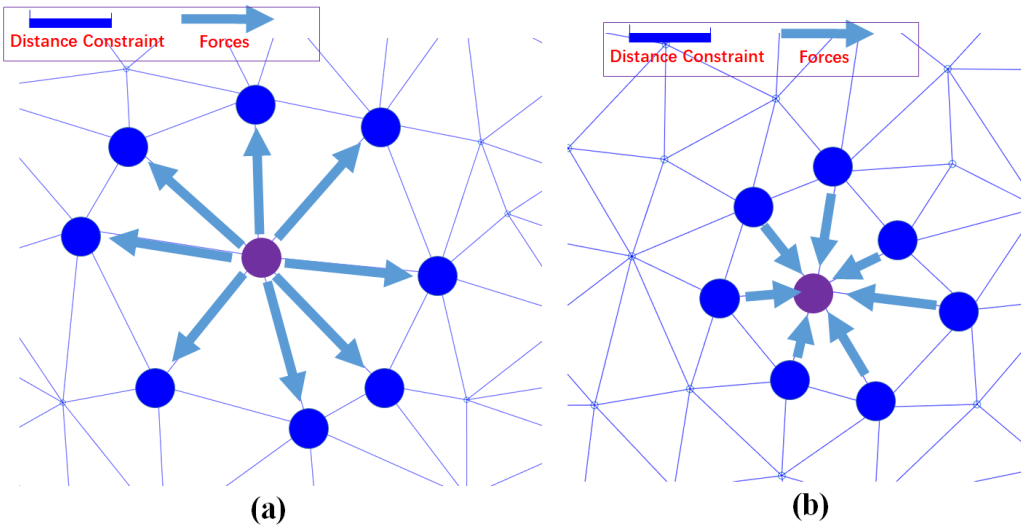


Figure 4.4: Examples of how local balance is formulated. (a). Samples will be pulled by the differentials from neighbors, but all forces are canceled out. (b). Samples will be pushed by the differentials from neighbors, but all forces are canceled out as well. The effective force for each iteration will stay low, even though there is tension around the current processing sample point.

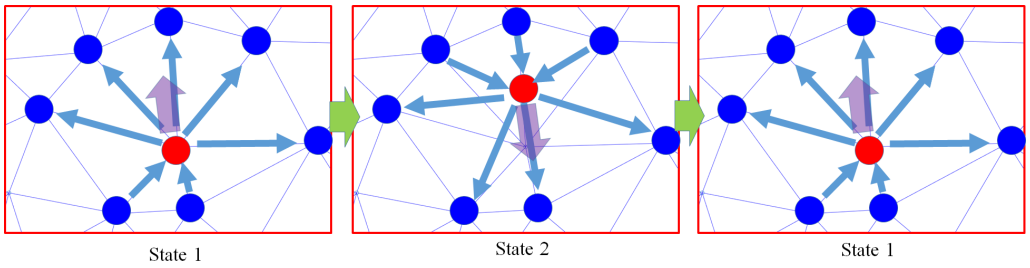


Figure 4.5: Assume that all neighbors of the current processing sample are all settled somehow with a relatively stable tension vector operated on them. The current sample may have two stable states in this case. The sample will jump back and forth at the two-state, if no parameter bounding is specified.

4.2.2 Constraint: A discussion

There are various constraint to take into consideration while designing the energy function on each sample while doing the iterative process of point cloud relaxations. Some famous examples are capacity[Balzer et al., 2009], centroidal[Du et al., 1999], a combined constraint CapCVT[Chen et al., 2012], a combined metric of conflict radius, coverage radius and capacity[Ahmed et al., 2016], a pair-correlation function's gradient[Öztireli and Gross, 2012], etc..

Most of these constraints (except pair-correlation based methods) have one thing in common: their energy function will have a global minimum on regular hexagonal arrangements of samples. Which lead to a concern that they might generate the regular pattern. But as they have introduced randomness and also tolerance of the constraints. What's most important is their initial states of samples: they came from uniformly generated samples or randomly jittered grid samples, that introduced the original random seed in the iterative process. Because of the constraints tolerance and randomness of initial states, samples will

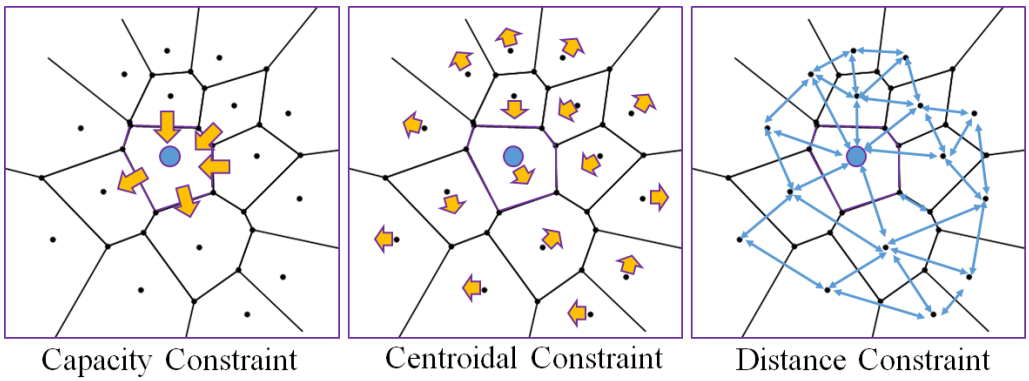


Figure 4.6: Different constraints used to synthesis blue-noise samples with relaxation-based method from a random initial state. The energy function defined in the sample set will be minimized by constantly morphing the sample sets to meet the constraints. Distance constraints can also achieve a local minimum fast.

eventually be optimized to a local minimum instead of the global minimum, which is what most relaxation based algorithms seeking for. Some algorithm like [Balzer et al., 2009] consider their local minimum with the same value evaluated from the energy function with global minimum. Others, however, hope the algorithm converge at a local minimum instead of the global minimum.

It is now clear that why most of the time CCVT[Balzer et al., 2009] and CapCVT[Chen et al., 2012] are better than CVT[Du et al., 1999] most of the time in the irregularity. The main reason is that they have more local minimum states. During experiments, we find that even a simple constraint such as to minimize the variance of lengths of all Delaunay triangulation in of all samples seems to have a really good spread of local minimum, as long as the radius/length constraints are set with enough tolerance, and the initial state being random. We will make some comparison with setting different parameters for each.

4.2.3 Distance Constraint

From Poisson-disk sampling, we get that the simple regulation of sample distance can produce great blue-noise sampling pattern. From this perspective, it is natural to also use distances between samples as constraints in generating blue-noise samples through relaxation-based techniques. To use distance as a constraint is not new, [Ahmed et al., 2016] used the $r_{conflict}$ and $r_{attraction}$ to apply a regulation on sample sets. But their methods are hard to make parallel, as they lack a strategy to solve the canceling-out problem in the parallel scheme addressed before. Some equilibrium-based methods can also be considered as a distance-constraints method.

To formulate the constraints, we consider all connections between the Delaunay triangulation on the convex hull of the point sets as an Edge set $\mathcal{E}$, the distance constraint method will minimize the sum of all differentials of all length of the edges with an ideal radius r_{avg} noted as D:

$$D = \frac{1}{N}\sum_{i=1}^{N} \|(e_i - r_{avg})\|^2, e_i \in \mathcal{E} \tag{4.5}$$

But this is impractical because of two reasons: (1). The r_{avg} is not well-

defined. For different profiles, there will be different settings of the r_{avg}. and (2). The algorithm is hard to converge without tolerances. The r_{avg} is a definitive value with zero tolerances, which will cause serious oscillation with even the slightest step size, this is not practical in real applications. It's common to add tolerances with respect to the conflict radius and attraction radius.

Here we will be using radius and distance interchangeably. We follow the definition of [Ahmed et al., 2016], but with a difference that the "radius" used in this chapter will always refer to the distances between two samples, instead of describing the radius of the sample disks. **Conflict Radius** is the smallest distance between any two sample points in the sample set $\mathcal{S}$.

$$r_f = \min_{\mathbf{x_i}, \mathbf{x_j} \in \mathcal{S}, i \neq j} E(\mathbf{x_i}, \mathbf{x_j}) \tag{4.6}$$

And our definition of coverage domain will be different. We use the Delaunay triangulation of the sample set's longest triangle edge to represent the coverage radius. So we will call the radius as **Attraction Radius** is the longest edge length of the sample set's Delaunay triangulation. That if the Delaunay neighbor is far away than this radius, they will start to have absorptive forces between each other.

$$r_a = \max_{(\mathbf{x_i}, \mathbf{x_j}) \in Edge(DT(\mathcal{S}))} E(\mathbf{x_i}, \mathbf{x_j}) \tag{4.7}$$

Based on the new constraints, samples within r_f will have high energy, and samples away from each other than r_c will contribute energies too. So the distance constraint with tolerance will be a piecewise sum:

$$D = \frac{1}{N_1 + N_2} \Big(\sum_{\forall e_i < r_f}^{N_1} \|(r_f - e_i)\|^2 + \sum_{\forall e_j > r_a}^{N_2} \|(e_j - r_a)\|^2 \Big), e_i, e_j \in \mathcal{E} \tag{4.8}$$

Notice that the conflict radius is actually just the regulation of Poisson-disk sampling: that a blue-noise sample set can be synthesized with the dart-throw algorithm only regulated by the conflict radius. The second regulation is valued in maximal Poisson-disk sampling methods, where they claimed that when no gaps exist in the domain, the samples will appear to have better characteristics in various applications. They have also been taken into considerations in relaxation based algorithms such as in [Ahmed et al., 2016], but not in the context of an apparent way or with better parallelization taken into consideration. Figure 4.7 shows the difference between these two different definitions of energy function based on distance constraints.

Also, notice that we did not make it generalized to adaptive sampling. When adaptive sampling is taken into consideration, the r_f and r_a should be multiplied by an underlying density map that is defined by the user. For example, in image stippling, all samples will be fetching their corresponding radius relative to the density they get from the bit-map.

4.2.4 Optimize based on Distance Constraints

When we have defined the energy function on the sample set, the optimization will be much more clear: during each iteration, we manage to reduce the energy by each moving of the sample. This process often introduces Lagrange multipliers to minimize the objective function, but our constraint is simple enough without needing to introduce extra parameters. The differential of each point will be their

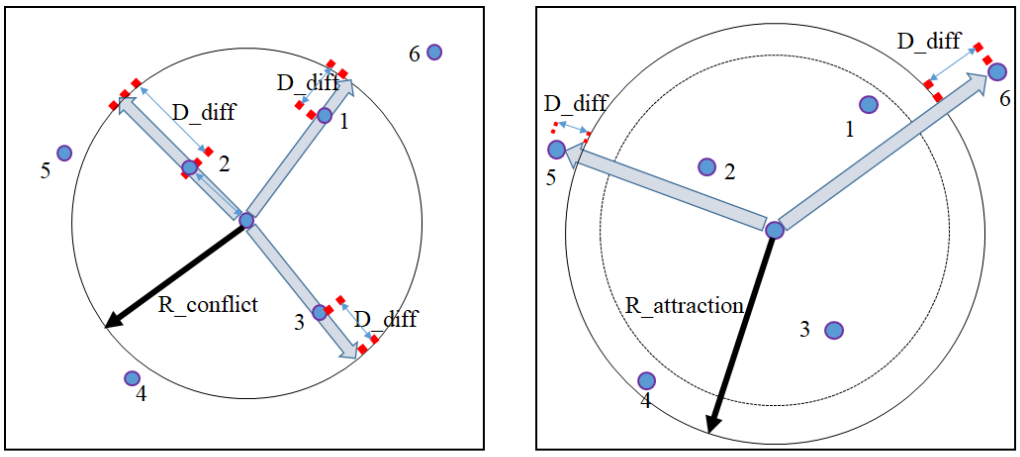


Figure 4.7: Explain the r_f and r_a. Consider points 1 6 on the sample plane are going to test their constraints against samples in the middle. Sample 1, 2, 3 will conflict mid samples as they violate the conflict radius constraints. So they should be modified in the following iterative process to gradually met the condition. While sample 5 and 6 have a longer distance than the r_a, thus they begin to gain absorptive force between the midpoints. Samples will then be morphed accordingly by the norm of their differential vector comparing the specified radius parameters. Notice that sample 4 just drop in between the r_a and r_f, which will remain stationary for this turn. This is the tolerance introduced to avoid serious oscillation of sample and enlarge the local stationary equilibrium so that sample will be easier to converge during iteration while keeping the random seeds they bear from the initial states.

difference vector comparing r_f and r_a, thus the gradient of a sample considering the number of its neighbor set $\mathcal{N}(x_i)$ will be:

$$\nabla_x D = \frac{2}{N_1 + N_2}\Big(\sum_{\forall e_i \in \mathcal{N}_x < r_f} -\mathbf{e_i} + \sum_{\forall e_j \in \mathcal{N}_x > r_a} \mathbf{e_j} \Big) \tag{4.9}$$

Which is a very simple optimization based only on the connecting edge vectors between the sample and its neighbors. The direction is also simple to determine, that all samples close to the current sample will add a force pushing the current sample away, while neighbors that are far away will add an absorptive force.

To bound the effective force acted on each sample for fast convergence, we need to bound the displacements δD_i to be proportional to the differential vector (or edge vector) to its neighbor. The displacement **Force** vector is:

$$F = \lambda\Big(\sum_{\forall e_i \in \mathcal{N}_x < r_f}^{N_1(\mathcal{N}_x)} -(r_f - \|\mathbf{e_i}\|)\frac{\mathbf{e_i}}{\|\mathbf{e_i}\|} + \sum_{\forall e_j \in \mathcal{N}_x > r_a}^{N_2(\mathcal{N}_x)} (\|\mathbf{e_j}\| - r_a)\frac{\mathbf{e_j}}{\|\mathbf{e_j}\|} \Big) \tag{4.10}$$

All the differential vector (or edge vector) will be pointing towards their neighbors $\mathcal{N}$.

But a simple theory doesn't always mean a good numeric solution, that this constraint also suffers from what we talked above: tension propagation speed, local equilibrium, and oscillation. In the next section, we will try to alleviate these problems with second-order information defined on each sample.

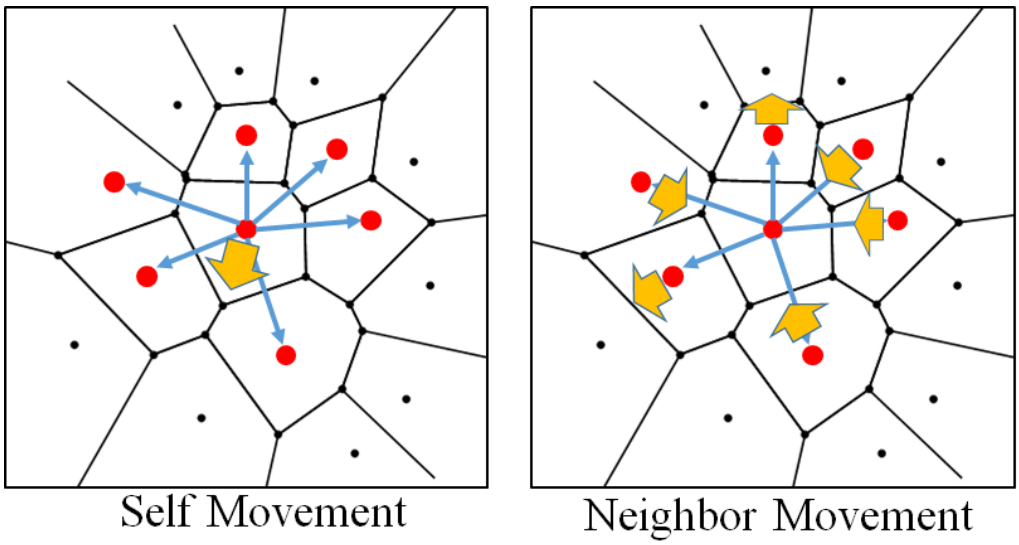


Figure 4.8: The differential vector can be used to generate displacement vector on both sides equivalently. The distribution of how the force of displacement generation will be a key concept in our algorithm. Sometimes it's better to make self movements, while other times move neighbors will optimize the local neighborhood much faster. A balancing mechanism between these two will be proposed in this chapter.

4.3 Methodology

In this section, we are going to introduce the methodology of our efficient parallel distance constraint method to generate a blue-noise sample pattern. The discussion will be in a 2D domain, it can be transferred to 3D or mesh surfaces with some extension.

4.3.1 Observation: Self-movements vs. Neighbor-movements

From previous sections, we know that almost all the optimization based underlying mechanism will have trouble executing in parallel. A small enough step size for each iteration can alleviate the problem but introduced serious damage on performance.

The inspiration of our algorithm came from insights of experiments of the distance-constrained method. When the gradient vector is being optimized, it can equivalently move the current sample itself, or move neighbors of the sample. When each sample is being processed, the gradient vector will calculate a downhill direction to minimize the distance energy, the direction will then dot product with a step vector to get the displacements. During the parallel computing, we know that each sample will affect other neighbors as well in the same iteration, thus the same differential can be distributed on both samples so that we don't have to calculate twice the same displacement vector for each sample.

Here comes the key point in our algorithm: should we distribute the displacement vector for the calculation evenly, or weight and test to see if there exists a better distribution. From a top view, it would be like should the samples move based on the gradient vector calculated directly from the neighbor, or should sample move its neighbors in each iteration equivalently. See Figure 4.8.

Figure 4.9 shows how our first intuition of finding such a force distribution method to break the balance come from adding a second-order information for

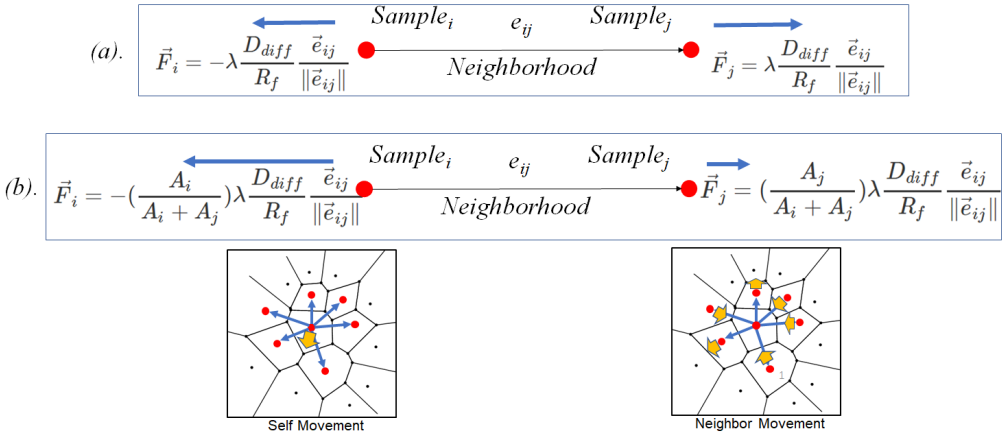


Figure 4.9: Force vector can be distributed arbitrarily between neighbor samples. (a). By default, neighbor samples will apply symmetric even forces from each other. But the force vector of neighbor samples can be distributed arbitrarily, To move themselves more, or move neighbors more, create asymmetric. With an importance value A defined on each sample for the distribution.

each sample involved. Samples will be compared with its neighbors to see weather the force will be distributed more towards itself or towards its neighbors.

On the local equilibrium problem that caused the convergence speed to slow down significantly, we propose to calculate an **Activeness** value on each sample to decide how the distribution of forces should be in the process towards equilibrium. Our goal is to accelerate tension propagation, and also keep the distribution of samples being irregular. What's more, it's possible to reach an equilibrium of local minimum with a better radius statistics based on the new movement strategy.

Now, our next goal would be to decide how this value A could be defined on each sample, so that we can have a value representing a sample's different weight in the distribution of displacement vector, in order to create asymmetries between samples to avoid local balance.

4.3.2 Utilize Activeness

Observation: Activeness of Samples

We define a second-order function evaluated on each sample named **Activeness**. Samples with high activeness will be in charge of moving the other neighbors rather than move themselves, while samples with low activeness will follow other samples order to move around. To define the activeness of samples, we observe that samples that are more easily stuck on a local balance are samples that have been in local constraints that effective forces tensions canceled each other out on the incident sample, though they still stretching or pushing the sample individually. Thus these samples shall have a high motivation to be actively deciding other sample's displacement vectors. On the other hand, there are situations where the tensions on the sample might not be high but the effective force executed on the specific sample is effective which results in an effective displacement vector, the sample should just follow the orders of its neighbors and reduce its activeness during the iteration process.

A simple analogy would be that when a leader in the team is putting a democratic strategy along the work process. If members have a different opinion over

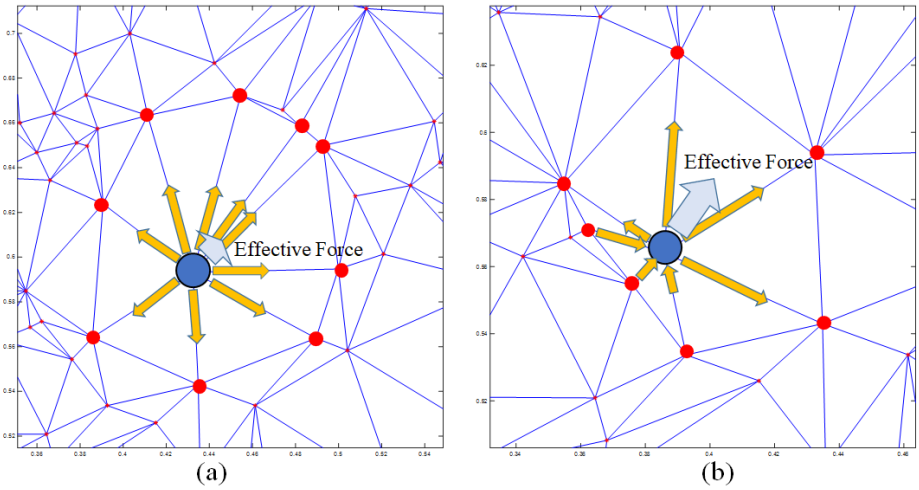


Figure 4.10: Explain the activeness for different conditions and how activeness help resolves the local balance. (a). Tensions on the sample are really high, as almost all neighbors of the current processing sample are acting absorptive force on the point. But the final effective force is relatively small. The sample is clearly not in a constrained state, thus the activeness of the central blue point will be increased, so that the movement displacement vector will be more distributed across its neighbors rather than itself, thus help to resolve the local balance. (b). The central blue point is being dragged by an effective force of large scale, thus the activeness will be low as long as the direction and scale keep clear, it will be moved around by neighbors instead of self displacements.

the execution of a certain act, which might lead projects to different directions that totally contradict each other, the leader has to be in charge and proceed with his own will. While if members have similar ideas to push the process of the projects along a planned route, the leader should comply with public opinion in order to achieve the final goal (which is the convergence in our case) faster.

An explanation of such a concept is shown in 4.10.

Definition

We define the activeness of a sample denoted as α:

$$\alpha_i = \frac{\sum_{j \in \mathcal{N}_i} (\|F_j\|)}{\| \sum_{j \in \mathcal{N}_i} F_j \|} \tag{4.11}$$

That α will be the sum of all the norms of tension forces added to this sample, over the norm of all added forces. In other word, the alpha will be defined by its current tensions and how effective they are in generating displacements. The more forces there are, means the sample is being pulled or pushed by other neighboring samples, the more activeness it will gain. Meanwhile, if the forces generated by its neighbor will be effective to create a big enough displacement vector, that means the tensions added on the current sample is not evenly distributed, and will not generate a local equilibrium (or local balance) on the current iteration, which means that the activeness of current sample doesn't need to be high to know where it will be going to. But if the tensions are high, but effective forces are low, that means the local balance is acting on the current sample, which

means that the current sample needs to be in charge of moving its neighbors to break the current situation.

When samples need to distribute tension force vector across each other, the displacement vector calculated by the pair will distribute the total displacements to the point by linearly comparing their α value:

$$F_1 = F\frac{\alpha_1}{\alpha_1 + \alpha_2}$$
$$F_2 = F\frac{\alpha_2}{\alpha_1 + \alpha_2}$$

4.3.3 Algorithm based on Delaunay Triangulation Neighborhood

Here we will discuss two variations of our algorithm. We tested the algorithm on a Delaunay based neighborhood and a KNN (K-nearest neighbor) based neighborhood. The Delaunay Triangulation of samples will be our base algorithm to run as it is robust to run algorithms on the faithfully connect triangulations. As it is the theoretically correct way to run based on how our differential is calculated. The algorithm will be shown in Algorithm 29.

4.3.4 kNN Neighborhood

As a much faster alternative from the Delaunay triangulation based neighborhood, the KNN neighborhood can be a less strict but much faster method. Instead of the carefully designed connectivity and neighborhood relationships in the original Delaunay triangulation, this method tries to find nearest K neighbors as the indication of how forces are calculated. To avoid attraction clustering, a KNN neighborhood can only include repulsive vector displacements. Meanwhile, the KNN neighborhood can be really sensitive to the radius settings for each sample, that if the radius setting is too strict, as the KNN neighborhood has a strong motivation to find the global minimum of the constraint, the results will tend to have regular patterns.

The KNN algorithm will be similar to algorithm 29, with the only difference that the neighbors are calculated from k-nearest neighbor search instead of using their Delaunay triangulated neighbors, while the attraction force is not well-defined in the KNN neighborhood, we will only use repulsive force to do the updating. All the other parameters are defined as the same. a

4.3.5 Resolve Oscillation

Although the Alpha value solved the false local balance, it did not address the oscillation problem, which is more related to the history of the force vector for each sample. We set a similrity parameter δ, and each time we will do a test on the force history comparing with the currently calculated force and displacements of the current iteration. If the norm of the dot product of the current force and minus history force is higher than 0.9 (comparing the longer vector of current force and minus history force), that means the sample is oscillating around, and jumping back and force between two stationary points. We then enforce the alpha of current point to be infinity, in order to stabilize the current points while moving its neighbor instead. In this way, the oscillation problem will be significantly reduced during the whole execution. See Figure 4.11 for some reference of this problem.

Enforcing Distance Constraints with Delaunay Triangulation Neighborhood;

Input : Initial sample set $\mathcal{S}$, conflict radius r_f, attraction radius r_a, step size λ

Output: Enforced sample set $\mathcal{S}$

Construct Delaunay Triangulation DT of $\mathcal{S}$;
Set initial α array for each sample to be 1;
while *Not converged* **do**
  forall *samples* x_i *in* $\mathcal{S}$ ***in Parallel*** **do**
    Initialize vector F;
    forall *neighbor samples* x_j *of* x_i **do**
      $e_{ij} \leftarrow x_j - x_i$;
      if $\|e_{ij}\| < r_f$ **then**
        $diff \leftarrow r_f - \|e_{ij}\|$;
        $Alpha \leftarrow \frac{\alpha_i}{\alpha_j + \alpha_i}$;
        $F_i \leftarrow F_i - \lambda(1 - Alpha)\frac{diff}{r_f}\frac{e_{ij}}{\|e_{ij}\|}$;
        $F_j \leftarrow F_j + \lambda(Alpha)\frac{diff}{r_f}\frac{e_{ij}}{\|e_{ij}\|}$;
      end
      if $\|e_{ij}\| > r_a$ **then**
        $diff \leftarrow \|e_{ij}\| - r_a$;
        $Alpha \leftarrow \frac{\alpha_i}{\alpha_j + \alpha_i}$;
        $F_i \leftarrow F_i + \lambda(1 - Alpha)\frac{diff}{r_a}\frac{e_{ij}}{\|e_{ij}\|}$;
        $F_j \leftarrow F_j - \lambda(Alpha)\frac{diff}{r_a}\frac{e_{ij}}{\|e_{ij}\|}$;
      end
    end
  end
  forall *samples* x_i *in* $\mathcal{S}$ **do**
    $\alpha_i \leftarrow \frac{\sum_j^N \|F_j\|}{\|\sum_j^N F_j\|}$;
  end
  $\mathcal{S} \leftarrow \mathcal{S} + F$;
  Updating DT;
end
Psudocode of Updating Force with Delaunay Triangulation Neighborhood

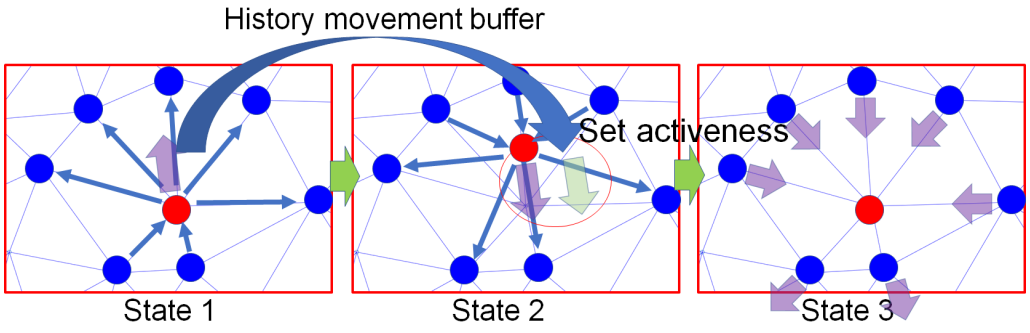


Figure 4.11: Oscillation problem. Samples will be jumping back and force in on stationary states. To resolve oscillation, we detect the force history by comparing similarities.

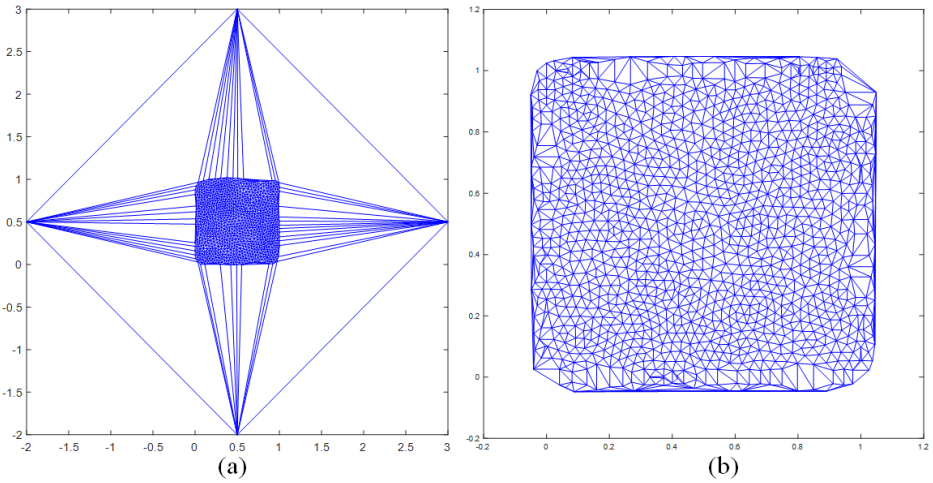


Figure 4.12: Delaunay triangulation of samples with extra pining points. (a). Use extra stable samples away from the central domain. This is a simple solution, but will still leave samples at the boundary behave in unordered manner. (b). Use extra mirrored samples to make the domain periodic. Samples inside the sampling domain will have constraints beyond the grid, thus regulated its behaviors.

4.4 Implementation and Discussion

4.4.1 Implementation Details

We implement the algorithm on a consumer level PC, with CPU of i7-4930K, and NVIDIA GTX1080 GPU. The Delaunay based algorithm is based on a CPU version, while the kNN based neighborhood is implemented on GPU.

Delaunay Triangulation Neighborhood

We use the CGAL's standard fast Delaunay Triangulation algorithm in our implementation.

Note that straight forward Delaunay Triangulation works on the convex hull point cloud. But with a distant constraint, we have to convert the point cloud to a convex hull point cloud. To do this, we copy and flip the boundary points in the sampling domain to ensure that all triangulation happened inside the sampling domain will be in a sub convex hull set. There is also a simple solution of adding four pin samples far away from the sample plane to maintain convex. See Figure 4.12 for some demonstration.

KNN Neighborhood

Notice that our algorithm can run on a k-Nearest neighbor algorithm instead of Delaunay triangulation to make it faster. Which is one advantage over the push-pull algorithm in [Ahmed et al., 2016]. As we don't need a strict connection of related Delaunay triangle neighbors to constrain any capacity related parameters, which made our methods much more easy to implement and faster to execute.

As we only have a simple constraint of distance. We will show the difference between using each method to find valid neighbors. The k-nearest neighbor is more ad-hoc and not robust, but it can be extremely fast when executed on GPU. While Delaunay triangulation based neighbor searching can provide more

stable results and fewer iterations until convergence. The reader can choose the appropriate methods of searching nearby samples.

4.4.2 Choice of Parameters

r_f

According to the maximal conflict radius in the domain is the radius of the corresponding hexagonal arrangements of points. $r_{max} = \sqrt{\frac{(2\sqrt{3})}{3}N}$ then we can set the conflict radius and attraction radius accordingly. The r_f is the most important parameter in the algorithm. We will test how our algorithm works under different settings of r_f.

As we only utilized distance as the main constraint for the final convergence, a strict r_f means a much more regular final pattern as there is no tolerance for randomness placements of sample points. Meanwhile, a tolerance r_f could cause gaps because the tension force may not be evenly distributed in the first place, thus causing the uneven results.

We find that with our Delaunay based distance-constraint methods, a good interval will be $0.8r_{max} < r_f < 0.97r_{max}$. For kNN based distance-constraint, it will partly depend on how K is selected, but generally speaking $0.9r_{max} < r_f < 0.99r_{max}$ can provide samples without apparent gaps, though a larger r_f sometimes generate regular bias artifacts.

r_a

The choice of r_a will have more room than r_f. r_a means samples in the DT with more than this radius will have attraction forces. As the blue-noise sampling is mainly focused on solving conflicts, the attraction algorithm is not as important, as it's only a mean to converge algorithm faster, while seek to leave no big gaps. In practice, we use $1.2r_{max} > r_a > 1.01r_{max}$.

Step size λ

This will be a tricky part, as this is a parameter that will greatly affect convergence speed, thus have to be controlled to test if our algorithm works or is it because of the dynamic λ that secretly affect the results. During our test, we use a stable λ setting as $0.05r_{max}$ to run the whole algorithm to eliminate the effects. But in practice, we recommend using a decaying λ that bounded to the minimum differential vector of the neighborhood. The dynamic λ will help converge and find a better local minimum.

K

In the k-nearest neighbor search, we find that the choice of K can be quite interesting. When K=1, all samples will only test with its nearest neighbor. What's good about it is that the algorithm will be almost ensured to converge (though currently no theoretical proof of the convergence, experiments shows that the convergence for K=1 is really good for many applications). The problem is that this algorithm will leave gaps with higher radius tolerance. Good thing is that the canceling-out problem will not happen under such cases.

Iter

The number of iteration is probably a parameter that needs to be tuned if the algorithm fails to reach convergence state, or an undefined state is acceptable sometimes in lower demanding applications. In such cases, we set a detector to check how variance is changed during iterating. If the variance changing kept less than 0.1 percent for continuous ten iterations, and the percentage of stable samples is higher than 99 percent, then we consider a loose convergence is reached.

δ

Which is the vector similarity that is used to control oscillating. The anti-oscillating mechanism will detect oscillating by comparing force with its minus adjacency history force. If they are of great similarity, then the sample is considered oscillating and its activeness will be forced to be infinity. We set the similarity to a cosine similarity of more than 95 percent (in cosine similarity, that means 0.95).

Figure 4.13 shows the DT based relaxation with a relatively loose radius tolerance. Figure 4.14 on the other hand, shows a relatively strict tolerance parameter. From the results we can see how this tolerance parameter will affect the final quality of the sampling.

4.4.3 Results and Analysis

Figure 4.16 and Figure 4.17 shows the comparison of sample patterns generated with previous methods and our methods. Evaluated with power spectrum, radial mean and anisotropy of each group of samples, our algorithm can generate comparable quality blue-noise samples with some of the best previous methods.

Discussion of GPU implementation and Parallelism

We implemented the DT-based method on CPU and kNN based method completely on GPU for GPU friendly execution. Comparing with the CPU-based implementation of Delaunay Triangulation, the k-nearest neighborhood on GPU can have much more parallelism. We claim that these parallelism with our activeness-based sample tuning activities are more effective because the distance variance can converge faster and also can converge to a even better global equilibrium, see 4.4.3 for more details. Figure 4.18 shows the performance comparison with previous work.

Distance Variance Comparison

We claimed the new method can converge faster, and most of the time can converge to a better local minimum comparing with using other strategies to determine how activeness going through the tensions. There can be the various distribution of forces among each sample. We compare with the following four: only move self, only move neighbors, move self and neighbor simultaneously with the same weight, and do a 0.2 vs. 0.8 distribution, which means just slightly distributed to self move, while moving neighbors with a larger force.

Here we show some results of five different strategies:

- **Ours**: methods used our second-order activeness to determine the distribution of tensions across neighbor points dynamically.

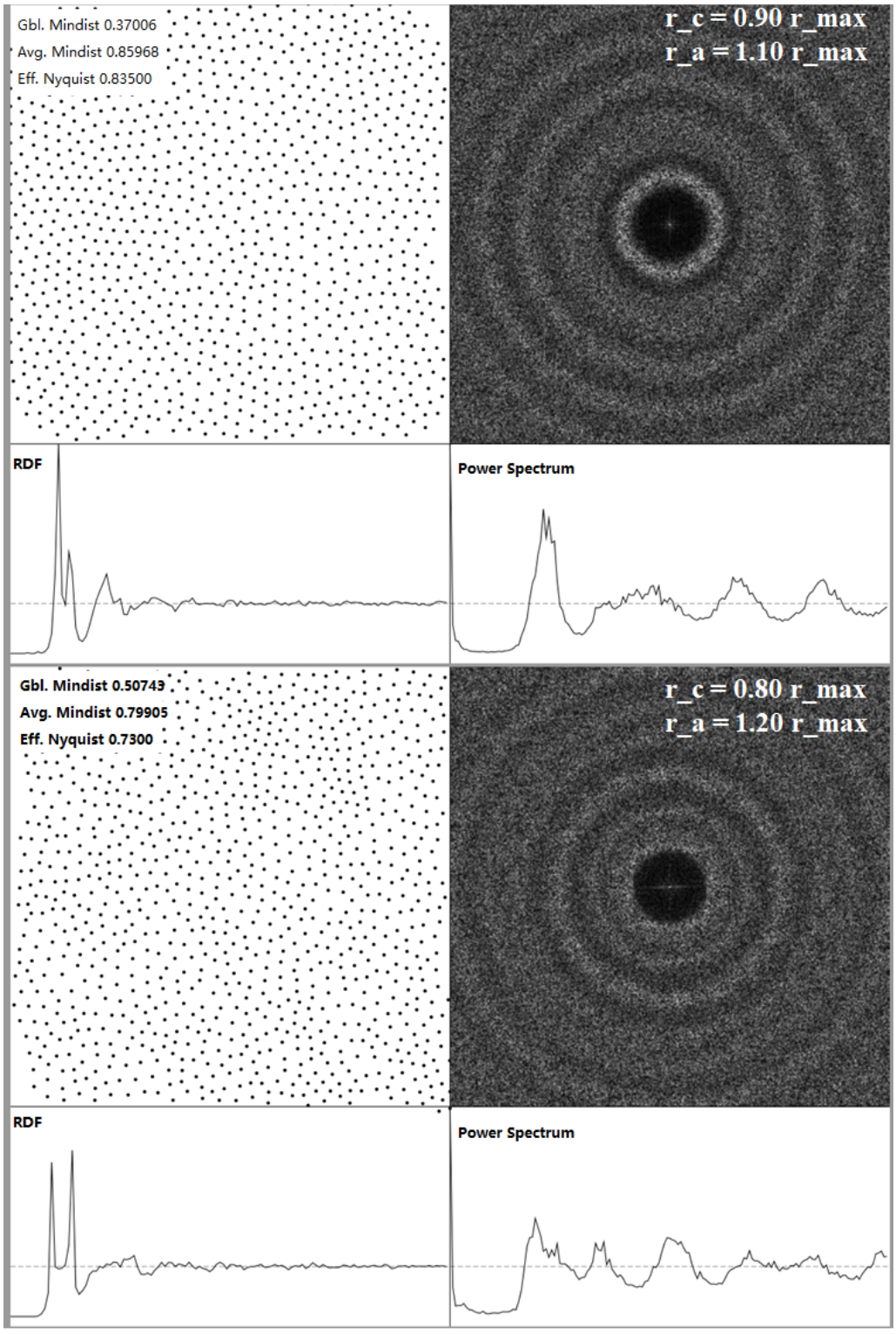


Figure 4.13: Delaunay based relaxation, with $r_c = 0.9r_{max}, r_a = 1.10r_{max}$ and $r_c = 0.8r_{max}r_a = 1.2r_{max}$. These are quite loose regulations, and we can see that the quality of our generated results OK but have more room to improve.

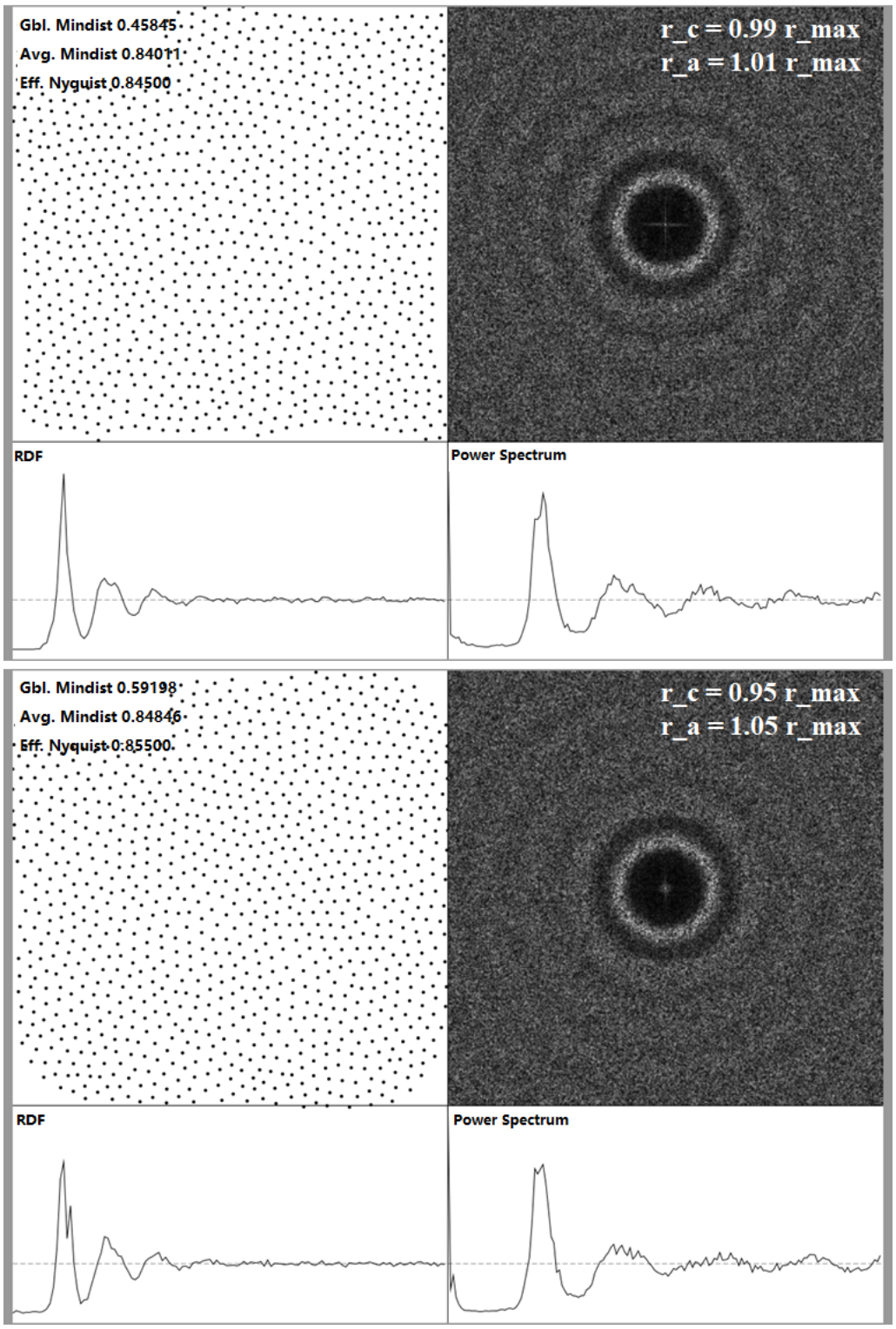


Figure 4.14: Delaunay based relaxation, with $r_c = 0.95 r_{max} r_a = 1.05 r_{max}$ and $r_c = 0.99 r_{max} r_a = 1.01 r_{max}$. These are quite strict regulations, and the constraint became faster. Notice that with extreme condition the generated results may appear to have some regular pattern, but most of the case it performs well.

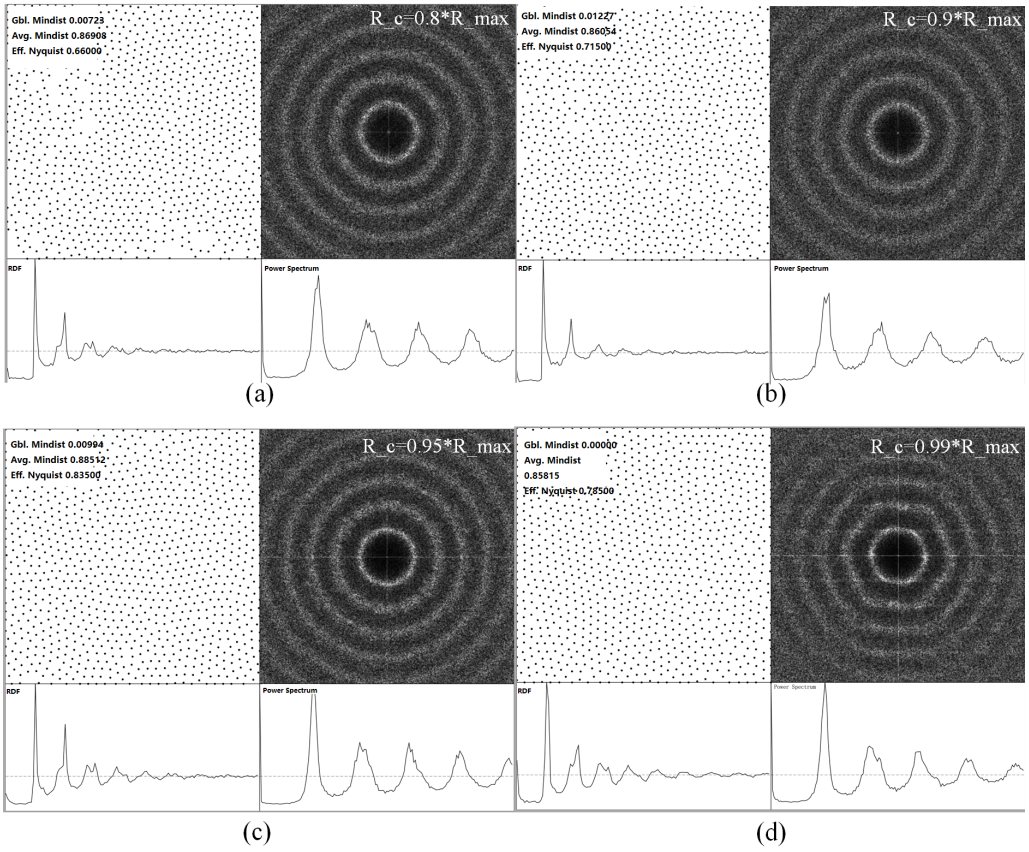


Figure 4.15: Distance-constrained relaxation based on kNN neighborhood. Lower radius regulation will leave gaps in the image, and if it is too strict, the regular artifact will soon be revealed. We recommend a $0.95r_{max}$ setting based on our experiments.

- **Average Alpha**: distribute alpha evenly on the edge of two neighboring samples.
- **Dominant Alpha**: The current processing sample will be in charge to move its neighbors, add all forces computed from the edge vector to its neighbor.
- **Zero Alpha**: The current processing sample will have no activeness, thus it adds force vector to itself. It is symmetric to the dominant alpha in a parallel environment.
- **0.8-0.2 Alpha**: the current processing sample will distribute the tension force in an 80-20 manner, Consider this strategy as moving itself drastically while moving neighbor slightly.

From Figure 4.19, we can see that our strategy is most the time the best strategy to use in such cases. Sometimes it may not perform significantly better due to different settings of step-size, radius, and so on. But it did perform better most of the time. Furthermore, the proposed alpha-based force distribution method most of the time will force the sampling to converge at a better state, where variance will be lower globally without introducing regularity. This is sometimes more important than the increase in convergence speed.

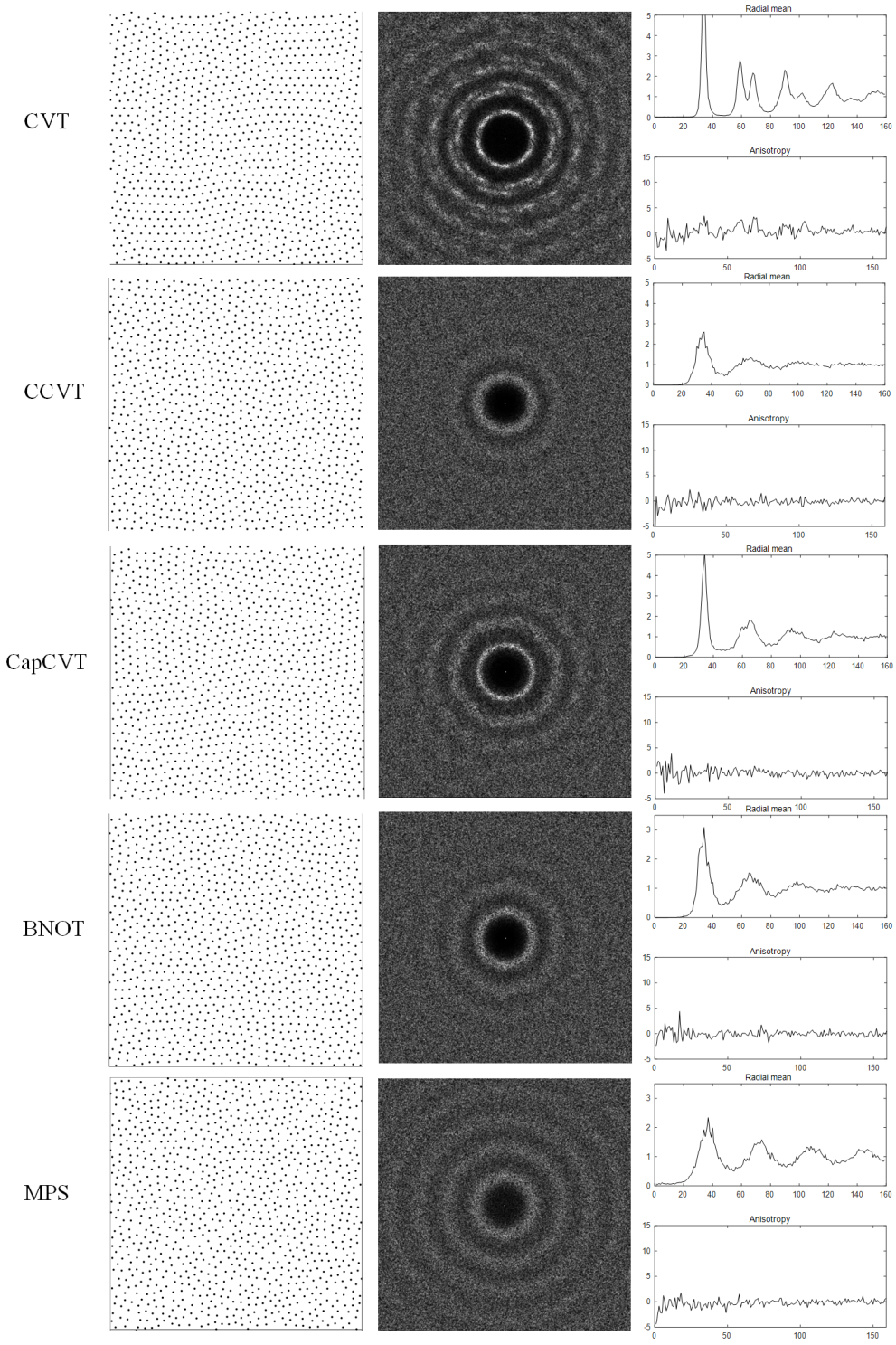


Figure 4.16: Samples, power spectrum, radial mean and anisotropy of CVT [Du et al., 1999], CCVT [Balzer et al., 2009], CapCVT [Chen et al., 2012], BNOT [Öztireli and Gross, 2012] and MPS [Ebeida et al., 2012]

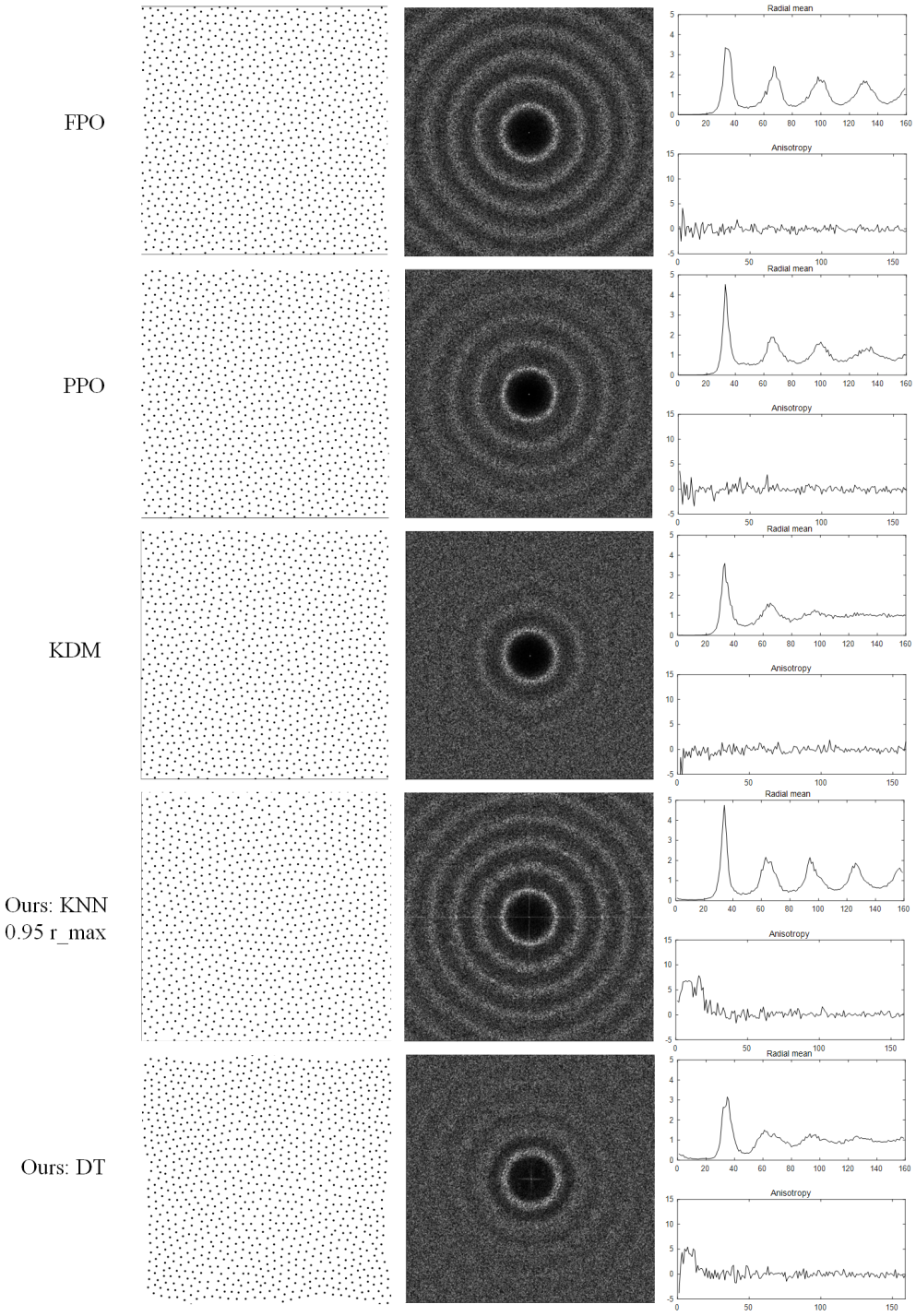


Figure 4.17: Samples, power spectrum, radial mean and anisotropy of FPO [Schlömer et al., 2011], PPO [Ahmed et al., 2016], KDM [Fattal, 2011], our KNN method and our Delaunay triangulation based method.

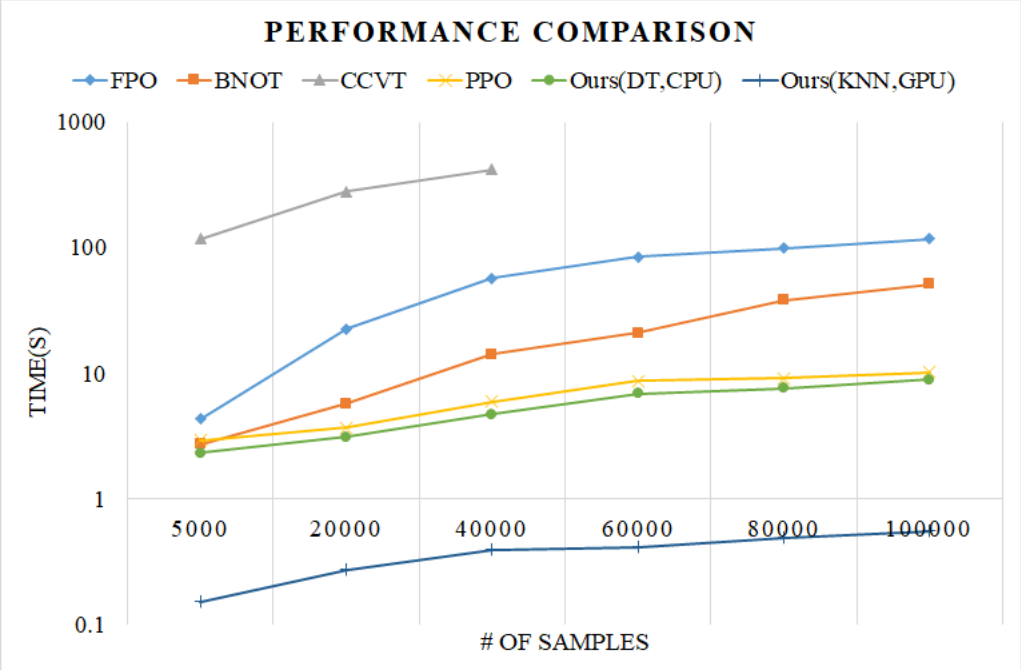


Figure 4.18: Some performance comparison with related work. The setting of our Delaunay triangulation based CPU method is $r_f = 0.9r_max, r_a = 1.1r_max$. The setting of our KNN method is $r_f = 0.95r_max$. We use default setting for most other related methods. Our KNN algorithm is efficient on GPU, while the DT algorithm can generate better distributed sample set.

On the other hand, Figure 4.20 shows that with extreme environments such as situations where radius regulation is loose, a more drastic solution might find its usage here. An explanation of this is that the 80-20 distribution methods equivalently enlarged its test radius, which means a larger step size. Large step size in each iteration will be beneficial in a loose parameter setting. Thus it may perform better than the rest. While dominant alpha and zero alpha suffer from canceling out and local equilibrium problems, and average alpha and our methods don't have an equivalently larger step size, which will not perform really well at such a setting. But we know from previous sections that the loose radius setting will not guarantee a good blue-noise sampling, thus narrowed its usage.

Overhead

Compare with the same distance-constrained method, the main overhead of our algorithm is the calculation of Alpha array for each sample. The extra array being kept will be updated during each calculation of forces, which will not introduce noticeable overhead. Notice that we calculate alpha for the next iteration, not for the current one, thus force value will not be gathered twice.

4.4.4 Tuning the Previous Method

As we mentioned in the relationship part of DCR and KDRT/PSP, that the DCR algorithm can be used as a tuning method to morph the Poisson-disk sampling pattern style to a high quality relaxation style sampling pattern which is proved to have much better quality in Monte-carlo integration based algorithm and mesh reconstruction (Remesh) applications. DCR can be used to do such kind of tuning

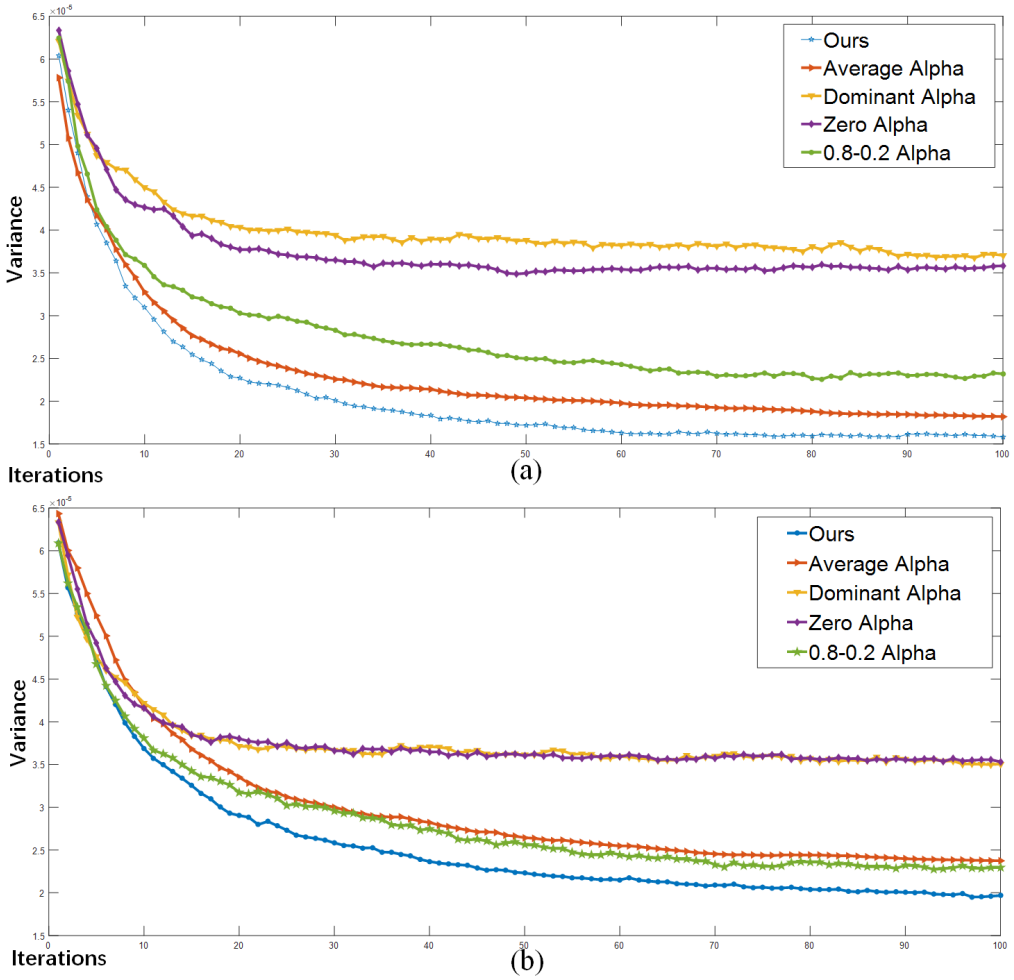


Figure 4.19: Situations where our algorithm performs significant better than other strategy of force distribution in parallel computing process during iteration. The x axis is iterations, y axis is variance of distances from each sample's nearest neighbor. From a distance-constrained perspective, the lower the better.(a). $r_f = 0.95r_{max}, r_a = 1.05r_{max}$. (b). $r_f = 0.9r_{max}, r_a = 1.1r_{max}$. These radius setting provide the best trade-off between sparsity and regularity. From the graphs we can see that our algorithm should be used in the considered good settings of radius status, as it can converge faster while also converge to a better local equilibrium considering distance-constrained.

on the sample pattern generated with KDRT or PSP. We will discuss how this process is proceeded, and show some result analysis of the tuning results.

We observe the iterations of convergence from an initial state of uniform random sampling, or from a Poisson-disk sampling pattern is different. Which is quite easy to explain that the Poisson-disk sampling pattern will be much more near the global equilibrium. Our activeness-based algorithm also helped the relaxation process.

We tested the tuning results in a Delaunay Triangulation quality on a 2D Poisson-disk Sampling scenario. To do analysis of triangle quality, we measure three different properties of the triangle: Percentage of obtuse triangles after tuning, percentage of acute triangles with less than 30 degree angles, and observe the variance of edges in Figure 4.21.

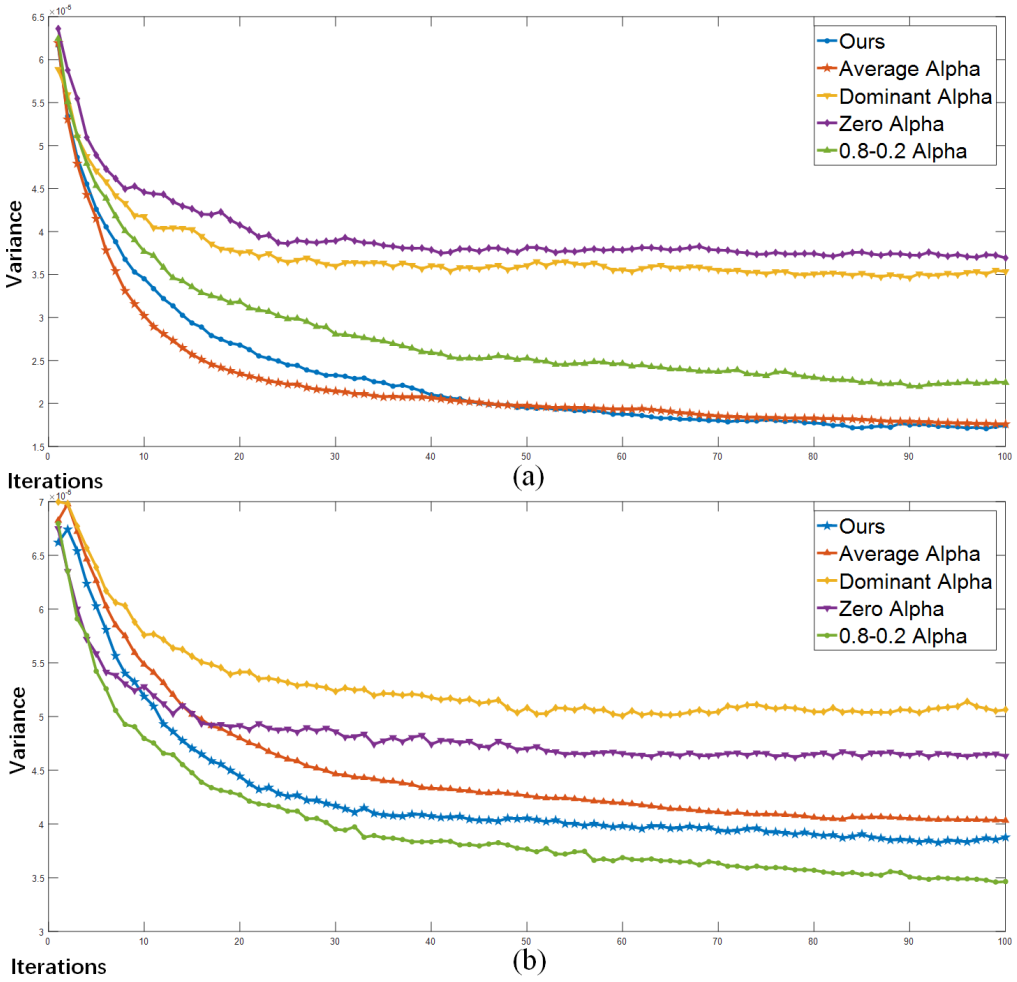


Figure 4.20: Situations where our algorithm performs similar or lower than other strategies. The x axis is iterations, y axis is variance of distances from each sample's nearest neighbor. From a distance-constrained perspective, the lower the better. (a). $r_f = 0.85r_{max}, r_a = 1.15r_{max}$. (b). $r_f = 0.8r_{max}, r_a = 1.2r_{max}$. These are sample set with quite loose regulations. In this case, the 0.2-0.8 Alpha means that the sampling is better to

	Avg. Iterations Before Convergence	
Initial Sample Set	**w/ α**	**w/o α**
by KDRT	132	146
by PSP(on 2D)	157	158
by Random	276	299

4.4.5 Limitations: Recap the Previous Work

One obvious limitation of this work comparing with previous work, is that the method will not have a way to decide the best tolerance in deciding r_c and r_a relatively. This could be a parameter that can be solved by sweeping the possible tolerance, but currently we could not provide a way to do theoretical analysis towards this direction.

Comparing with one of the best relaxation based model in relaxation: Push-Pull algorithm stated in [Ahmed et al., 2016], although we have achieved better speed, but their method can have a sole convergence condition. Also, if we

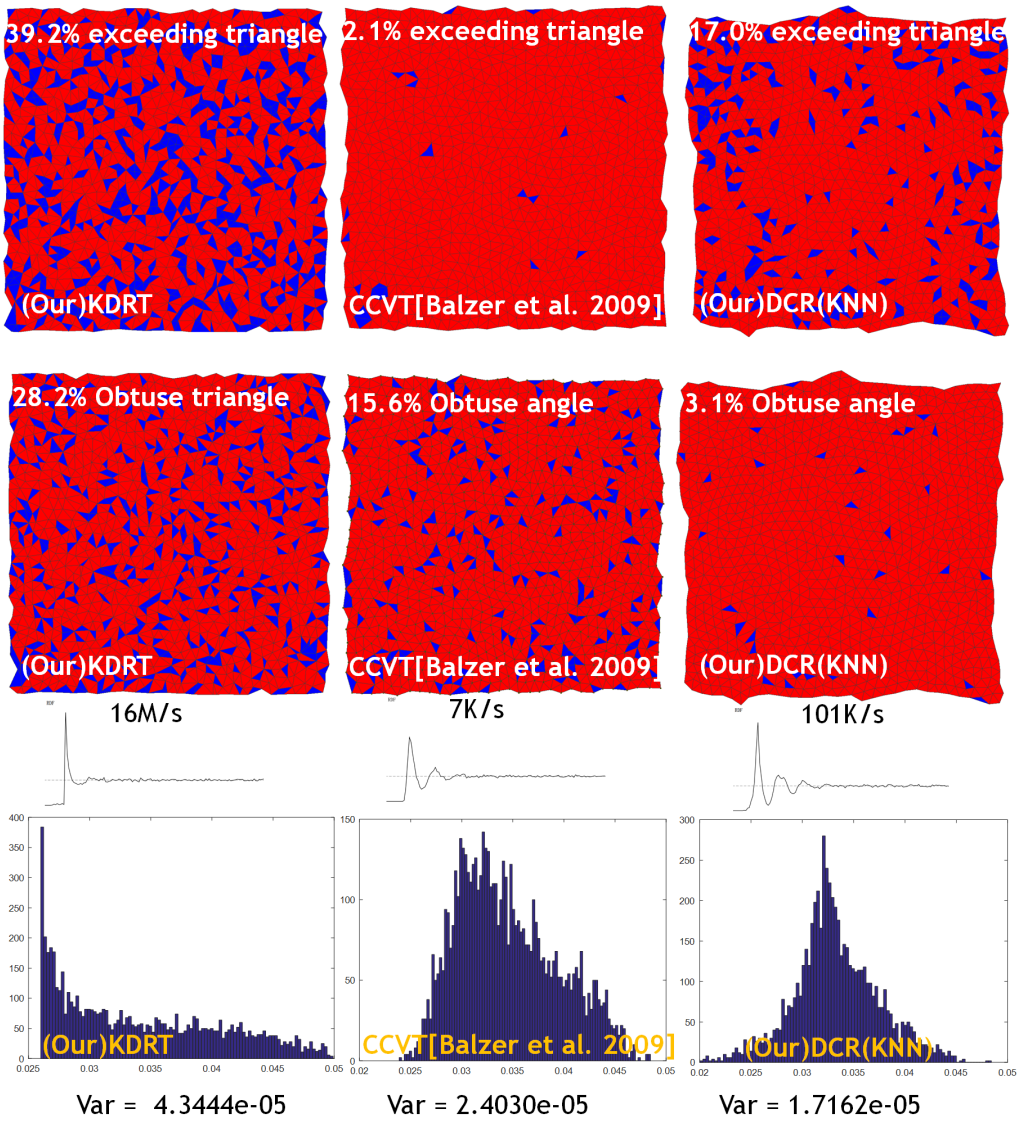


Figure 4.21: DCR tuning the KDRT for better performance in Re-meshing. The first row shows the percentage of triangles beyond 4 σ from the average triangle area in blue. The second row depict all obtuse triangles in blue, and the last row shows the variance of Delaunay triangulation edges. The DCR tuning results in much better quality measured by obtuse angle percentage and overall variance. CCVT performs better in triangle area as it is a method constraining the area of the Voronoi diagram. DCR performs better in distance variance and obtuse angle percentage. Both revealed much better quality metrics comparing with maximal Poisson-disk sampling pattern generated with KDRT.

consider regularity, the KNN based algorithm will generate regularity artifact, while it is alleviated in push-pull algorithm.

Comparing with the classic CCVT method [Balzer et al., 2009], our method will not be constraining on triangle area, so the quality mesh showed in figure 4.21 that CCVT can converge a better quality mesh than our method with the measurement of evenly distributed Delaunay triangle mesh average area.

Comparing with the FPO (farthest point optimization) method [Schlömer et al., 2011], although our method can generate samples faster with constrained distance, the FPO on the other hand have the advantage that it can generate

progressively better sampling pattern with less iterations, which is a valuable property in various applications.

4.4.6 Future Work

The performance of DCR is great for many applications such as mesh reconstruction, but we didn't propose a DCR that is suitable for mesh surface currently. It could be a quick and important future work to do.

The activeness-based parallelization, which solved local balance and oscillation, can be applied to many other relaxation based algorithms, such as CapCVT([Chen et al., 2012]).

One important limitation of DCR is the setting of tolerance parameter can be arbitrary, which prevent it from establishing a robust experimental results on measuring the relationship between tolerance parameters and the irregularity artifacts. To explore more in this direction can provide valuable insights on how to develop precise descriptions of the theoretical analysis on DCR. Meanwhile, the theoretical analysis of how the second-order heuristic activeness help to alleviate problem in local balance and oscillation can also be a great future work for us to do.

4.5 Conclusions

In this chapter, our contribution is of two part: first, we proposed distance-constrained blue-noise sampling, which is a very simple and fast blue-noise sampling algorithm. We can tackle the iteration of neighbors with Delaunay triangulation or kNN neighbors, and they both have their advantages and disadvantages. Second, we proposed to use the activeness second-order information defined on each point to accelerate the iteration and improve convergence status. The activeness value can dynamically distribute tension forces among pairs of samples to best suit their parallel computing conditions. We tested various evaluations of our algorithm including power spectrum, anisotropy and radial mean statistics, which showed that it can generate decent blue-noise samples.

The main limitation of our algorithm is that the radius and step-size specification will affect some final results, and there is no well-defined way to automatically set these parameters. When the radius regulation is loose, our algorithm may not be the best choice as it is not drastic enough to spread samples quickly across the domain. The kNN based distance-constrained relaxation may be really fast to use and is GPU friendly,

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we talked about mining effective parallelism from various groups of blue-noise sampling methods. All the techniques introduced before can be considered as how parallelisms are being exploited from the previously For extremely fast tile-based techniques, we proposed KDRT, which proposed to divide the sample space into randomly subdivided building blocks, then use these building block to clip from a pre-generated pattern, and finally use our conquering algorithm to combine the final results, and eliminate conflicts.

With this generated sample pattern, we can further use our Progressive Projection Sampling technique to generate Poisson-disk samples on mesh surfaces from rotating orthogonal projectors towards the mesh surfaces with real-time ray tracing engine. As the parallelism is carefully designed so that there are no intra-batch conflicts, which greatly increased parallelism, and thus can be very GPU friendly to execute. The samples can be further optimized by using a relaxation-based method to get a sample set with high quality, or we can synthesis samples from scratch. We proposed the activeness second-order information extracted from local tension characteristics to help alleviate three problems that will harm the parallelism of such a relaxation scheme. From comparisons with other parallel schemes and applications results, we conclude that our algorithm pushed forward the high-performance blue-noise sampling by designing effective parallelism, through randomized tiling, progressive projection, and relaxation with high efficiency. They are all friendly for GPU execution as well as there are no irregularities introduced in the process. Tiling, projection and kNN distance-constrained relaxations all have embarrassingly parallel execution in the process.

Our results show state-of-the-art performance comparing with related work, and also show its easy implementation and robustness in execution. The KDRT and Progressive Projection methods can produce a huge amount of samples in milliseconds and can be easily applied on applications such as image stippling, adaptive sampling and more. The distance-constrained technique itself is an easy and robust way to do blue-noise synthesis which can be useful in the most relaxation-based method.

5.2 Limitations and Future Work

Detailed limitations for each technique are listed at the end of each chapter. Here we'll discuss limitations from the whole problem-solving perspective.

The limitation of the thesis is that we considered parallelism and performance as a priority. In various applications, there may be some specific requirements for

different metrics to evaluate the sample quality. Through our progressive projection sample has increased quality during sampling, it is still a Poisson-disk sampling method with characteristics approximately near the maximal Poisson-disk sampling. If a much more high-quality sample set is needed, an extra refinement process might be needed.

Our methods didn't consider a general power spectrum like stated in [Zhou et al., 2012] and [Wachtel et al., 2014b]. But the activeness based distance-constrained relaxation method can have the potential be generalized combining with pair-correlation function to generate a general power spectrum.

Another limitation of the thesis is that these methods will suffer from a curse of dimensionality, which means that the performance of sampling will reduce Euclidean space. The main reason is that our algorithm will rely on a global spatial search data structure, and most such data structure is not effective in high dimensional space. To solve such a problem, we can consider combining our algorithms with the family of methods called advancing front algorithm (Mitchell et al. [2018]), as this group of methods can be good in solving high dimensional sampling problem.

Future work of the work will be to generalize all techniques multi-class sampling and high dimensional sampling. Some applications are not discussed thoroughly in the thesis, which should be supplemented in the future. More hardware-specific optimization can also be studied to better utilize GPUs and distributed computing systems.

In KDRT, the tiling process is KD-tree based, hence very easy to extend the method to high dimensions. Elimination and insertion involve higher dimensional neighbor search, which may suffer from a curse of dimensionality. With our experience and early-stage experiments, this method can at least maintain good quality and performance in 3D space sampling. Also, we see the potential of adaptive sampling with this method, as the KD-tree based tiling has the nature of recursive and subdivisions. It would be really interesting to see this method being used on applications such as image stippling.

One future work in PSP will be to use hardware-based ray tracing API instead of using GPGPUs to do that. With the newly proposed ray tracing APIs by NVIDIA and other IC producers and their real-time ray tracing solutions based on ASIC or FPGA hardware, tracing rays can be even much more effective than the current version.

Also, we would recommend future scholars proceeding this road to explore more relaxation possibilities during the iterative process, that despite the second-order information, more information that could predict the tension's local balance more accurately can be evacuated inspired by the activeness. It is also interesting to dig into the theoretical background inside the activeness-based heuristic, and prove why introducing this asymmetric can effectively resolve the local balance problem.

There are many more applications that can benefit from our high-performance blue-noise sampling algorithm. Also, we would recommend future scholars proceeding this road to explore more in the point cloud-based learning algorithms. As we discussed in the introduction chapter in the thesis, that the geometry-based learning has become an active and important field to explore. The generation of blue-noise samples can be applied directly to the generation of the point cloud from geometry, and also in the hierarchy abstract of the whole point cloud structure, as stated in [Qi et al., 2017b]. PointNet++, PointCNN and many other neural networks that directly consume point cloud can benefit from high-performance blue-noise sampling methods.

There are many other possibilities to explore this topic. If you proceed more through the performance track, you will encounter the curse of dimension that prevents the algorithm to be applied on high-dimensional applications. As we mentioned in the previous chapter, that this can be tackled from the advancing fronting based method, but these methods will still be not as effective as we want. Dealing with high dimensional conditions now have growing importance with the rising of point cloud-based learning we discussed, so dealing with high dimensional blue noise sampling is also a very interesting and important future work for future scholars to tackle.

References

Abdalla GM Ahmed, Hui Huang, and Oliver Deussen. Aa patterns for point sets with controlled spectral properties. *ACM Transactions on Graphics (TOG)*, 34(6):212, 2015.

Abdalla GM Ahmed, Jianwei Guo, Dong-Ming Yan, Jean-Yves Franceschia, Xiaopeng Zhang, and Oliver Deussen. A simple push-pull algorithm for blue-noise sampling. *IEEE transactions on visualization and computer graphics*, 23(12): 2496–2508, 2016.

Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-603-8. doi: 10.1145/1572769.1572792. URL `http://doi.acm.org/10.1145/1572769.1572792`.

Michael Balzer. Capacity-constrained voronoi diagrams in continuous spaces. In *2009 Sixth International Symposium on Voronoi Diagrams*, pages 79–88. IEEE, 2009.

Michael Balzer, Thomas Schlömer, and Oliver Deussen. *Capacity-constrained point distributions: a variant of Lloyd's method*, volume 28. ACM, 2009.

Maurice Stevenson Bartlett. *An introduction to stochastic processes: with special reference to methods and applications*. CUP Archive, 1978.

Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999.

John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM Transactions on Graphics (TOG)*, volume 29, page 166. ACM, 2010.

Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *SIGGRAPH sketches*, page 22, 2007.

Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. Variational blue noise sampling. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1784–1796, 2012.

Michael F Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. *Wang tiles for image and texture generation*, volume 22. ACM, 2003.

Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, 1986.

M. Corsini, P. Cignoni, and R. Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):914–924, June 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2012.34.

Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, volume 33, pages 88–104. Wiley Online Library, 2014.

Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. Blue noise through optimal transport. *ACM Transactions on Graphics (TOG)*, 31(6):171, 2012.

Mark AZ Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. *ACM Siggraph Computer Graphics*, 19(3):69–78, 1985.

Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM review*, 41(4):637–676, 1999.

Fredo Durand. A frequency analysis of monte-carlo and other numerical integration schemes. 2011.

Mohamed S Ebeida, Andrew A Davidson, Anjul Patney, Patrick M Knupp, Scott A Mitchell, and John D Owens. Efficient maximal poisson-disk sampling. In *ACM Transactions on Graphics (TOG)*, volume 30, page 49. ACM, 2011.

Mohamed S Ebeida, Scott A Mitchell, Anjul Patney, Andrew A Davidson, and John D Owens. A simple algorithm for maximal poisson-disk sampling in high dimensions. In *Computer Graphics Forum*, volume 31, pages 785–794. Wiley Online Library, 2012.

Mohamed S Ebeida, Scott A Mitchell, Muhammad A Awad, Chonhyon Park, Laura P Swiler, Dinesh Manocha, and Li-Yi Wei. Spoke darts for efficient high dimensional blue noise sampling. *arXiv preprint arXiv:1408.1118*, 2014.

Mohamed S Ebeida, Ahmad A Rushdi, Muhammad A Awad, Ahmed H Mahmoud, Dong-Ming Yan, Shawn A English, John D Owens, Chandrajit L Bajaj, and Scott A Mitchell. Disk density tuning of a maximal random packing. In *Computer Graphics Forum*, volume 35, pages 259–269. Wiley Online Library, 2016.

Raanan Fattal. Blue-noise point sampling using kernel density model. In *ACM Transactions on Graphics (TOG)*, volume 30, page 48. ACM, 2011.

Manuel N Gamito and Steve C Maddock. Accurate multidimensional poisson-disk sampling. *ACM Transactions on Graphics (TOG)*, 29(1):8, 2009.

Rafael C Gonzalez, Richard E Woods, et al. Digital image processing [m]. *Publishing house of electronics industry*, 141(7), 2002.

Simon Green. Particle simulation using cuda. *NVIDIA whitepaper*, 6:121–128, 2010.

Jianwei Guo, Dong-Ming Yan, Xiaohong Jia, and Xiaopeng Zhang. Efficient maximal poisson-disk sampling and remeshing on surfaces. *Computers & Graphics*, 46:72–79, 2015.

Daniel Heck, Thomas Schlömer, and Oliver Deussen. Blue noise sampling with controlled aliasing. *ACM Transactions on Graphics (TOG)*, 32(3):25, 2013.

Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. Deep-learning the latent space of light transport. *arXiv preprint arXiv:1811.04756*, 2018.

Janine Illian, Antti Penttinen, Helga Stoyan, and Dietrich Stoyan. *Statistical analysis and modelling of spatial point patterns*, volume 70. John Wiley & Sons, 2008.

Cheuk Yiu Ip, M Adil Yalçin, David Luebke, and Amitabh Varshney. Pixelpie: Maximal poisson-disk sampling with rasterization. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 17–26. ACM, 2013.

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. Blue noise sampling using an sph-based method. *ACM Transactions on Graphics (TOG)*, 34(6):211, 2015.

John Edward Jones. On the determination of molecular fields.ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):463–477, 1924.

Nima Khademi Kalantari and Pradeep Sen. Fast generation of approximate blue noise point sets. In *Computer Graphics Forum*, volume 31, pages 1529–1535. Wiley Online Library, 2012.

Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita. Deterministic blue noise sampling by solving largest empty circle problems. *The Journal of the Institute of Image Electronics Engineers of Japan*, 40(1):6–13, 2011.

Tero Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 33–37. Eurographics Association, 2012.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. *Recursive Wang tiles for real-time blue noise*, volume 25. ACM, 2006.

Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM Transactions on Graphics (TOG)*, 24(4):1442–1461, 2005.

Ares Lagae and Philip Dutré. A comparison of methods for generating poisson disk distributions. In *Computer Graphics Forum*, volume 27, pages 114–129. Wiley Online Library, 2008.

Thomas Leimkühler, Gurprit Singh, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel. End-to-end sampling patterns. *CoRR*, abs/1806.06710, 2018. URL `http://arxiv.org/abs/1806.06710`.

Bruno Lévy and Yang Liu. L p centroidal voronoi tessellation and its applications. In *ACM Transactions on Graphics (TOG)*, volume 29, page 119. ACM, 2010.

Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018a.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018b.

Zonghui Li, Tong Wang, and Yangdong Deng. Fully parallel kd-tree construction for real-time ray tracing. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 159–159. ACM, 2014.

Jae S Lim. Two-dimensional signal and image processing. *Englewood Cliffs, NJ, Prentice Hall, 1990, 710 p.*, 1990.

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Don P Mitchell. Spectrally optimal sampling for distribution ray tracing. In *ACM Siggraph Computer Graphics*, volume 25, pages 157–164. ACM, 1991.

Scott A Mitchell, Mohamed S Ebeida, Muhammad A Awad, Chonhyon Park, Anjul Patney, Ahmad A Rushdi, Laura P Swiler, Dinesh Manocha, and Li-Yi Wei. Spoke-darts for high-dimensional blue-noise sampling. *ACM Transactions on Graphics (TOG)*, 37(2):22, 2018.

Harald Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of number theory*, 30(1):51–70, 1988.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

A Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Transactions on Graphics (TOG)*, 31(6):170, 2012.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778803. URL `http://doi.acm.org/10.1145/1778765.1778803`.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. Variance analysis for monte carlo integration. *ACM Transactions on Graphics (TOG)*, 34(4):124, 2015.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017b.

Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. Projective blue-noise sampling. *Comput. Graph. Forum*, 35(1):285–295, February 2016. ISSN 0167-7055. doi: 10.1111/cgf.12725. URL `https://doi.org/10.1111/cgf.12725`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

John C Russ. *The image processing handbook*. CRC press, 2016.

Thomas Schlmer and Oliver Deussen. Accurate spectral analysis of two-dimensional point sets. *Journal of Graphics, GPU, and Game Tools*, 15(3):152–160, 2011. doi: 10.1080/2151237X.2011.609773. URL `http://dx.doi.org/10.1080/2151237X.2011.609773`.

Thomas Schlömer and Oliver Deussen. Accurate spectral analysis of two-dimensional point sets. *Journal of Graphics, GPU, and Game Tools*, 15(3): 152–160, 2011.

Thomas Schlömer, Daniel Heck, and Oliver Deussen. Farthest-point optimized point sets with maximized minimum distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 135–142. ACM, 2011.

Christian Schmaltz, Pascal Gwosdek, Andrés Bruhn, and Joachim Weickert. Electrostatic halftoning. In *Computer Graphics Forum*, volume 29, pages 2313–2327. Wiley Online Library, 2010.

Hugo Steinhaus. *Mathematical snapshots*. Courier Corporation, 1999.

Kartic Subr and Jan Kautz. Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration. *To appear in ACM TOG*, 32:4, 2013.

Xin Sun, Kun Zhou, Jie Guo, Guofu Xie, Jingui Pan, Wencheng Wang, and Baining Guo. Line segment sampling with blue-noise properties. *ACM Trans. Graph.*, 32(4):127–1, 2013.

Stanley Tzeng, Anjul Patney, Andrew Davidson, Mohamed S Ebeida, Scott A Mitchell, and John D Owens. High-quality parallel depth-of-field using line samples. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 23–31. Eurographics Association, 2012.

Robert A Ulichney. Dithering with blue noise. *Proceedings of the IEEE*, 76(1): 56–79, 1988.

Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando de Goes, Mathieu Desbrun, and Victor Ostromoukhov. Fast tile-based adaptive sampling with user-specified fourier spectra. *ACM Trans. Graph.*, 33(4):56:1–56:11, July 2014a. ISSN 0730-0301. doi: 10.1145/2601097.2601107. URL `http://doi.acm.org/10.1145/2601097.2601107`.

Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando De Goes, Mathieu Desbrun, and Victor Ostromoukhov. Fast tile-based adaptive sampling with user-specified fourier spectra. *ACM Transactions on Graphics (TOG)*, 33(4):56, 2014b.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601199. URL `http://doi.acm.org/10.1145/2601097.2601199`.

Tong Wang and Reiji Suda. Fast maximal poisson-disk sampling by randomized tiling. In *Proceedings of High Performance Graphics*, HPG '17, pages 16:1–16:10, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5101-0. doi: 10.1145/3105762.3105778. URL `http://doi.acm.org/10.1145/3105762.3105778`.

Tong Wang and Reiji Suda. Fast generation of poisson-disk samples on mesh surfaces by progressive sample projection. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):30, 2018.

Li-Yi Wei. Parallel poisson disk sampling. In *ACM Transactions on Graphics (TOG)*, volume 27, page 20. ACM, 2008.

Li-Yi Wei. Multi-class blue noise sampling. *ACM Transactions on Graphics (TOG)*, 29(4):79, 2010.

Li-Yi Wei and Rui Wang. Differential domain analysis for non-uniform sampling. *ACM Trans. Graph.*, 30(4):50:1–50:10, July 2011. ISSN 0730-0301. doi: 10.1145/2010324.1964945. URL `http://doi.acm.org/10.1145/2010324.1964945`.

Kin-Ming Wong and Tien-Tsin Wong. Blue noise sampling using an n-body simulation-based method. *The Visual Computer*, 33(6-8):823–832, 2017.

Yin Xu, Ligang Liu, Craig Gotsman, and Steven J Gortler. Capacity-constrained delaunay triangulation for point distributions. *Computers & Graphics*, 35(3): 510–516, 2011.

Dong-Ming Yan and Peter Wonka. Gap processing for adaptive maximal poisson-disk sampling. *ACM Transactions on Graphics (TOG)*, 32(5):148, 2013.

Dong-Ming Yan, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka. A survey of blue-noise sampling and its applications. *Journal of Computer Science and Technology*, 30(3):439–452, 2015.

John I Yellott. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, 221(4608):382–385, 1983.

Xiang Ying, Shi-Qing Xin, Qian Sun, and Ying He. An intrinsic algorithm for parallel poisson disk sampling on arbitrary surfaces. *IEEE transactions on visualization and computer graphics*, 19(9):1425–1437, 2013.

Cem Yuksel. Sample elimination for generating poisson disk sample sets. *Comput. Graph. Forum*, 34(2):25–32, May 2015a. ISSN 0167-7055. doi: 10.1111/cgf.12538. URL `http://dx.doi.org/10.1111/cgf.12538`.

Cem Yuksel. Sample elimination for generating poisson disk sample sets. *Comput. Graph. Forum*, 34(2):25–32, May 2015b. ISSN 0167-7055. doi: 10.1111/cgf.12538. URL http://dx.doi.org/10.1111/cgf.12538.

Sen Zhang, Jianwei Guo, Hui Zhang, Xiaohong Jia, Dong-Ming Yan, Junhai Yong, and Peter Wonka. Capacity constrained blue-noise sampling on surfaces. *Comput. Graph.*, 55(C):44–54, April 2016. ISSN 0097-8493. doi: 10.1016/j.cag.2015.11.002. URL https://doi.org/10.1016/j.cag.2015.11.002.

Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. Point sampling with general noise spectrum. *ACM Transactions on Graphics (TOG)*, 31(4):76, 2012.