N° d'ordre NNT : xxx

PHD FROM UNIVERSITÉ DE LYON
developped within
Université Claude Bernard Lyon 1

École Doctorale 512
Infomaths
Specialty : Computer Science

Publicly defended the 07/03/2018, by :
Hélène Perrier

Anti-Aliased Low Discrepancy Samplers for Monte Carlo Estimators in Physically Based Rendering

In front of the following jury :

Abstract (English)

When displaying a 3-D object on a computer screen, this 3-D scene is turned into a 2-D image, which is a set of organized colored pixels. We call *Rendering* the process that aims at finding the correct color to give those pixels. This is done by simulating and integrating all the light rays coming from every direction that the object's surface reflects back to a given pixel. Unfortunately, a computer cannot compute an integrand directly. We therefore have two possibilities to solve this issue:

1. We find an analytical expression to remove the integrand (statistic based strategy).

2. We numerically approximate the equation by taking random samples in the integration domain and approximating the integrand value using Monte Carlo methods.

Here, we focused on numerical integration and sampling theory. Sampling is a fundamental part of numerical integration. A good sampler should generate points that cover the domain uniformly to prevent bias in the integration and, when used in Computer Graphics, the point set should not present any visible structure, otherwise this structure will appear as artefacts in the resulting image. Furthermore, a stochastic sampler should minimize the variance in integration to converge to a correct approximation using as few samples as possible. Many different samplers exist that we roughly regroup into two families:

- Blue Noise samplers, that have a low integration variance while generating unstructured point sets. The issue with those samplers is that they are often slow to generate a point set.

- Low Discrepancy samplers, that minimize the variance in integration and are able to generate and enrich a point set very quickly. However, they present a lot of structural artefacts when used in Rendering.

The goal of this PhD was the study of the theoretical characteristics and practical impact of samplers combining both the Blue Noise and Low Discrepancy properties.

Abstract (French)

0.1 Problème

Lorsque l'on affiche un objet 3-D sur un écran d'ordinateur, on transforme cet objet en une image 2-D, c.a.d en un ensemble de pixels colorés. Les couleurs observées par l'oeil humain sont fonction du déplacement de la lumière dans le monde et de la façon dont cette lumière intéragit avec les différents matériaux des différents objets. On appelle *Rendu* la discipline qui consiste à trouver la couleur à associer aux pixels pour représenter une scène donnée. Pour ce faire, on va placer une caméra virtuelle dans une scène 3-D, découper le film de cette caméra en pixel, et calculer la quantité de lumière qui arrive sur chaque pixel du film (nommé aussi plan image) de cette caméra.

Si on nomme I le pixel du plan image qui nous intéresse, on calcule sa couleur en résolvant l'équation (équation simplifiée) suivante:

$$I = \int_S L(p, v) V_p(v) dp \tag{0.1}$$

Ce que cette équation signifie c'est que pour trouver I, on va intégrer toute la lumière que chaque point p de la scène S renvoie dans la direction v, v étant la direction menant de p à I. Le tout est pondéré par une fonction binaire V_p determinant si un point p de S est visible ou non depuis I. Cependant, pour déterminer la quantité de lumière que p renvoie vers v, on utilise la fonction L, qui elle même intègre la totalité de la lumière que p reçoit depuis tous les autres points de la scène et calcule quelle fraction de cette lumière p renvoie vers v.

On a donc déjà deux intégrales dans cette version très simplifiée de l'équation. Malheureusement, l'ordinateur ne sait pas calculer d'intégrales; on a donc deux solutions possibles pour faire un rendu:

1. Trouver une expression analytique qui permette de supprimer l'intégrale de l'équation (approche basée statistique).

2. Approximer numériquement l'équation en tirant des échantillons aléatoires dans le domaine d'intégration et en en déduisant la valeur de l'intégrale via des méthodes dites de Monte Carlo.

La difficulté d'une représentation statistique de la surface réside dans le fait qu'une représentation statistique globale n'est pertinente que dans des cas très restreints (distribution gaussienne, pas de problèmes de visibilité, uniformité de l'objet ...).

Lors de cette thèse nous nous sommes donc plutôt intéressés à la deuxième solution, celle de l'intégration numérique, basée sur la théorie de l'échantillonnage. Ce sont ces méthodes qui constituent le noyau dur de cette thèse, et qui ont données lieu au développement de deux nouveaux échantillonneurs. Notre première méthode génère des échantillons 2-D ayant une excellente distribution spatiale. Elle est étendue dans notre deuxième méthode, qui peut produire des échantillons N-D ayant une distribution de qualité équivalente. Cette seconde méthode permet également de contrôler la quantité d'échantillons localement et en offre la possibilité de rajouter incrémentalement des échantillons tout en préservant la qualité de la distribution.

Dans la suite, nous présenterons tout d'abord la théorie de l'échantillonnage, avant de détailler nos contributions.

0.2 Théorie de l'échantillonnage

Dans le cadre du Rendu, si on ne peut pas calculer analytiquement l'intégrale, on va approximer sa valeur en simulant un certain nombre n de rayons lumineux. Cela revient donc à un problème d'échantillonnage; le problème étant de choisir correctement ces rayons pour avoir la meilleure approximation possible de l'intégrale. Pour ce faire, on va utiliser des méthodes dites de Monte Carlo et essayer d'approximer I grâce à l'équation (0.2).

$$E(I) = \sum_{i=1}^{n} p_i f(x_i) \tag{0.2}$$

Dans cette équation, x_i est le $i^{ème}$ échantillon, et f est la fonction à échantillonner. Par rapport à notre équation (0.1), $f(x_i)$ correspondrait à $L(x_i, v)V_p(v)$. On a également $\sum_i = 1^n p_i = 1$ pour normaliser le tout. Plus n est grand, plus on est proche de la valeur réelle pour I.

On note que suivant le choix des échantillons x_i le résultat de cette équation varie, plus ou moins suivant la quantité d'erreur mesurée. Dans le cas du rendu, le choix d'un échantillonneur est essentiellement fonction de cette erreur. En effet, comme on intègre chaque pixel individuellement, plus cette erreur est grande, plus on va observer de différences entre deux pixels voisins. Cela crée des artefacts dits de *bruit* dans l'image finale. Il a été montré que le fait d'avoir une distribution spatiale des x_i de type *bruit bleu* est optimale pour réduire cette variance. Cette distribution, essaye de maintenir la même distance entre chaque point et tous ses voisins.

Une autre solution pour réduire ce bruit est d'utiliser des échantillonneurs dits *basse discrépance*. La discrépance est une mesure mathématique qui permet d'évaluer l'uniformité d'une distribution spatiale de points. Plus la discrépance est basse, plus la distribution est uniforme. Pour être qualifié de *basse discrépance*, un échantillonneur doit voir sa discrépance baisser avec l'augmentation du nombre N de points à une vitesse $O\left(\frac{\log(N)^{D-1}}{N}\right)$, où D est le nombre de dimensions. Ces échantillonneurs présentent des distributions de points très structurées, et ont pour but d'avoir la distribution la plus uniforme possible. Malheureusement, lors d'un rendu, ils créent une sorte de grille très régulière dans l'image. C'est ce que l'on appelle l'*aliassage*. Ils ont cependant l'avantage pour la plupart d'être des séquences de points. Cela signifie que l'on peut rajouter des points à un ensemble existant tout en préservant la basse discrépance.

Les travaux qui aujourd'hui constituent le noyau dur de cette thèse, consistent à concilier ces deux catégories d'échantillonneurs pour avoir une l'erreur la plus faible possible en intégration sans créer d'aliassage dans l'image finale.

Nous avons pu produire deux travaux sur ce domaine. Le premier présente un premier échantillonneur en 2-D qui ne présente pas d'aliassage tout en ayant une faible erreur lors de l'approximation de l'intégration. Il est décrit Section 0.3. Section 0.4 présente notre deuxième contribution. Nous y présentons un échantillonneur toujours à faible variance et anti-aliassé, mais pour de plus hautes dimensions. Il permet également de contrôler localement la quantité d'échantillons et offre la possibilité d'augmenter incrementalement le nombre d'échantillons.

0.3 Premier échantillonneur 2D

Ce premier échantillonneur était la première tentative pour concilier à la fois une discrépance basse et une distribution *bruit bleu*. En utilisant des ensembles de points 1-D connus pour être basse discrépance, on peut construire un ensemble 2-D où chaque ligne et chaque colonne est elle même un ensemble basse discrépance. Il a été prouvé qu'un tel ensemble 2-D est lui même un ensemble basse discrépance. Ensuite, en réarrangeant correctement cet ensemble 2-D, on peut changer la distribution des points. Cela nous permet d'obtenir une distribution de points *bruit bleu* (ou d'autres distributions), sans changer la discrépance de l'ensemble initial.

Pour réarranger l'ensemble de points, on permute les coordonnées x des points puis les coordonnées y. Intervertir des points formant un ensemble basse discrépance 1-D ne change pas la discrépance de l'ensemble, du coup notre ensemble 2-D résultant est toujours composé de lignes et de colonnes qui sont elles même basses discrépance. Ce qui fait que notre ensemble final est basse discrépance, tout en ayant une distribution de points différente. Pour obtenir une distribution ciblée, on précalcule et on stocke l'ensemble de permutations sur x et y à appliquer et l'échantillonneur n'a plus qu'à lire ces données et à faire la permutation. Tout ceci est illustré Figure 0.1.

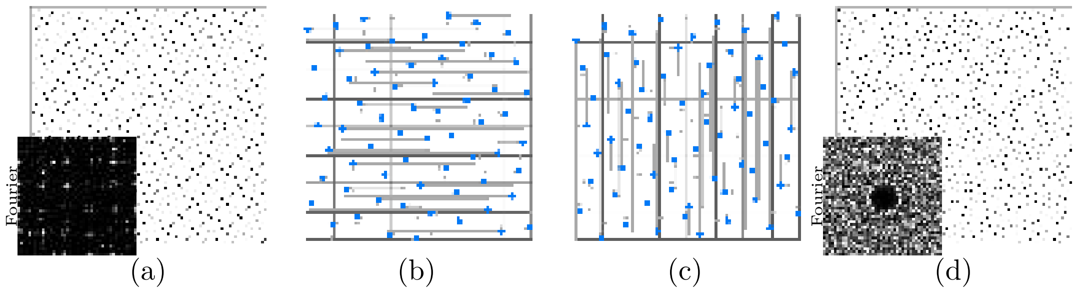


Fig. 0.1.: (a) Présente l'ensemble de points 2-D initial de notre méthode. Cet ensemble est permuté sur chaque coordonnées (b,c) Pour donner un ensemble qui soit bruit bleu à l'arrivée (d). On peut noter que l'ensemble de départ a un spectre de Fourier très différent de celui de l'ensemble d'arrivée. On peut également noter que le spectre de Fourier de l'ensemble d'arrivée présente une croix en son centre. Cette croix est symptomatique de la basse discrépance et n'est en aucun cas un handicap aux capacités de notre échantillonneur en terme d'intégration.

Cette méthode a le mérite d'avoir réussi à prouver que la basse discrépance et la distribution *bruit bleu* étaient parfaitement compatibles. Elle est cependant limitée dans ses applications pratiques. Elle ne génère que des points 2-D, n'est pas séquence, et ne permet pas de contrôler localement la densité des points.

0.4 Second échantillonneur N-D et Séquentiel

Ce second échantillonneur est lui aussi *bruit bleu* et basse discrépance. Il exploite le fait que si un ensemble est *bruit bleu* dans chaque sous partie (suffisamment grande) du domaine, il le sera également sur le domaine entier.

Notre méthode part donc d'une séquence basse discrépance qu'elle subdivise en tuiles carrées. Ensuite, on applique nos permutations locales, sur chaque tuile. Nous avons prouvé formellement dans le papier que ces permutations nous permettent de préserver la discrépance de l'ensemble entier malgré le fait qu'elles soient locales. Parmi toutes les permutations locales possibles, nous avons précalculé celles qui correspondaient au mieux à une distribution *bruit bleu*, et nous les avons stockées.

De plus, grâce à des permutations globales, dont nous avons également prouvé qu'elles préservent la discrépance, nous pouvons rendre notre échantillonneur séquence. Cela signifie que si nous avons un ensemble de points répartis dans des tuiles à un niveau l, nous pouvons resubdiviser ces tuiles au niveau $l+1$ pour rajouter des points tout en préservant la position des points déjà présents.

Enfin, comme notre système est basé sur des tuiles, on peut choisir de ne subdiviser qu'une partie des tuiles pour changer localement la densité des points.

Tout ceci est illustré Figure 0.2.

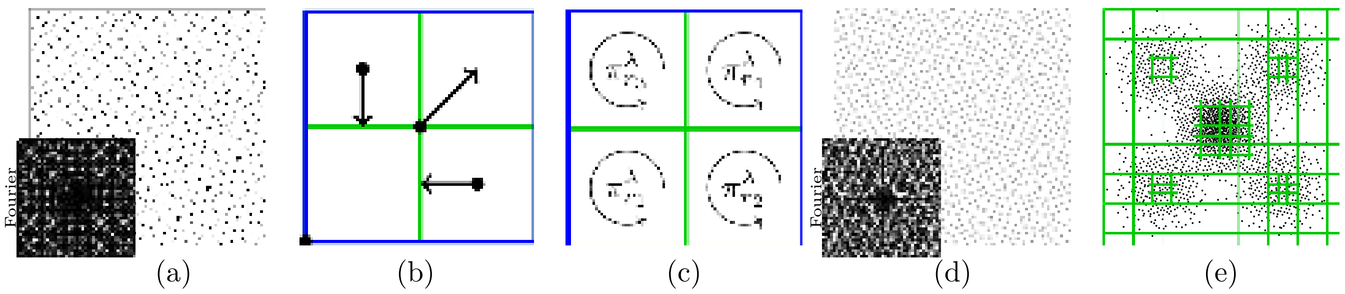


Fig. 0.2.: (a) Présente l'ensemble de points 2-D initial, répartis dans des tuiles à un niveau l. On permute individuellement chaque tuile (b,c) pour obtenir un ensemble qui soit bruit bleu à l'arrivée (d). Une tuile contient toujours le même nombre de points. Pour rajouter des points, on subdivise les tuiles. Cette subdivision peut être faite localement pour faire varier la densité de points (e).

Acknowledgement

Tout d'abord, je m'excuse auprès de tout ceux qui ont (ou qui vont) lire ce document ... Un grand merci aux rapporteurs et membres du jury qui eux n'ont pas eu le choix !

Pour le reste je suis pas sûre de savoir quoi marquer ici, j'ai toujours été nulle avec les mots ... Je n'ai rien contre faire vos éloges individuels, mais vous êtes tous tellement géniaux que ça augmenterai massivement le nombre de pages de ce document et après j'aurai toutes les associations contre la déforestation sur le dos. Je ne tiens pas non plus à faire une liste de noms parce qu'on fini toujours par oublier quelqu'un et c'est super vexant. Du coup je veux juste dire à tous les potos de labos (oui David et Victor vous êtes aussi mes potos), à tous mes amis et à toute ma famille à quel point je vous aime fort. Sans vous ces 3 années n'auraient été qu'un long cauchemard, et j'aurai probablement tout laissé tomber au bout d'une seule année sans regrets. Grâce à vous non seulement j'ai tenu suffisamment longtemps pour écrire ces lignes, mais maintenant que c'est fini, je pars en plus avec un gros pincement au coeur en pensant que je ne vous verrai plus (ou plus assez à mon gout !). Un grand merci pour tout ce que vous m'avez apporté (burger, body combat, équitation, jeux de société, séances de bitch/gateaux, robot de cuisine trop bien, et j'en passe ...) Je finirai avec cette citation tirée d'un chef d'oeuvre cinematographique moderne

Friends have a way of making even the worst of times into something pretty great.

(My Little Pony : Friendship is magic)

I love you guys !

PS: Ah oui, je remercie aussi tous les chats qui m'ont ronronné dessus !

Contents

Context 1

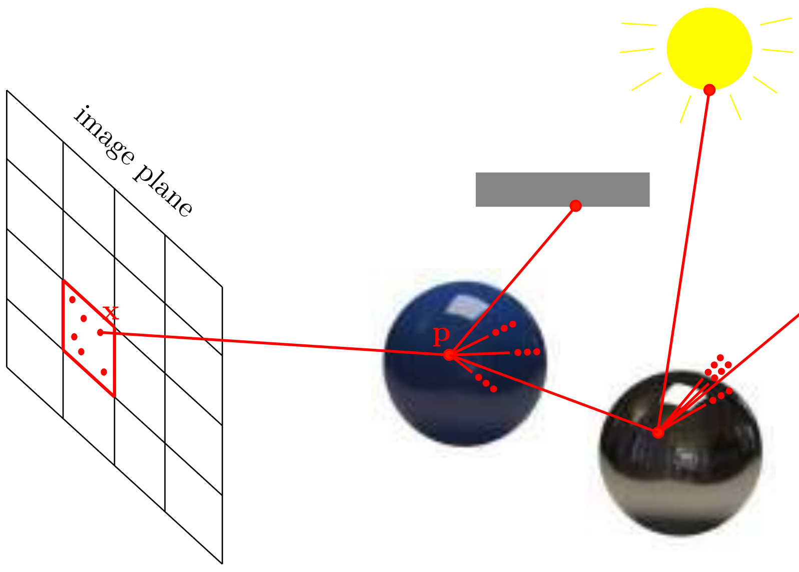
Nowadays, computer generated images are ubiquitous. They appear in movies, commercials, video games and have become almost undistinguishable from real pictures. This "realisticness" is partly due to the modern technologies. They increase so drastically our processing power, that it is now possible to try simulating the physical light field in the scene. The issue is that a physical light field in the real world is something that is infinite and continuous. Computers cannot handle any of those properties, and therefore, to simulate this light field, we need to rely strongly on sampling and discretization.

To generate a synthetic image we need to first be able to sample this simulated light field (meaning being able to measure the amount of light in a particular point of the scene), and second, we need to be able to reconstruct a 2-D image from those sampled values. Our eyes do this reconstruction thanks to a set of cells covering our retinas. Each of those cells receives a discrete portion of the real light field in front of us. Those values are then sent to the brain that reconstructs an image from it. To create a synthetic image from a simulated light field, we replicate this process of sampling a set of discrete values from the simulated light field and use tools such as convolutions or Fourier transform to reconstruct an image from it. Interested readers can find a detailed description of this process in appendix A on page 168.

Our main focus will then be on how to simulate and sample this light field. As every object in a scene receives light and reflects light, evaluating the amount of light that reaches a particular point in a scene is very computationally expensive. It requires evaluating all the light incoming to this point from every other point on every possible objects in the scene. Mathematically, this translates as a recursive integration, which cannot be solved analytically. It requires to be approximated, using numerical integrations methods, which strongly relies on sampling. In this document, we will focus on the use of *Monte Carlo estimators* as our tool to solve this numerical integration. Such light field simulation will be described Section 1.1, with a focus on Monte Carlo estimators in Section 1.1.2 on page 4.

1.1 The Rendering process

To render an image, we need to simulate the flow of light bouncing within a scene. This implies recursively integrating for every single point of the scene the light incoming from every possible direction. To solve this set of complex integrands, we use Monte Carlo estimators, that rely on sampling to approximate the result of an integration. In the following, we first detail how we simulate the flow of light and the different integrands that need to be accounted for. Then, we see how we can use Monte Carlo estimators to approximate those integrands.



The light field in the scene is represented as a set of discrete light rays. The amount of light within each ray is computed by sending this ray from the image plane towards the scene, and bouncing it against the geometry until it reaches the light source.

Fig. 1.1.: The rendering process.

1.1.1 Path-Based rendering

Generating a synthetic image first requires a description of the scene to be rendered. This description contains a set of objects present in the scene (often referred to as the *geometry* of the scene), each of them being associated with a material, a color, a texture, a motion, etc. A scene description also contains a set of light sources and a description of the virtual photo camera taking the picture (its position in the scene, its lens and focal, its looking-at point etc.). This camera defines a virtual image plane in the scene, usually discretized into pixels.

To render a scene, we compute the amount of light incoming from the scene to each pixel of the image plane. This set of discrete radiance values will then be the input for a reconstruction process which will generate the final image (as described in Appendix A on page 168). This radiance can be seen as the amount of light reaching a point $\mathbf{x}$ from every possible directions. To evaluate this, we use path tracing algorithms. As this is a very active field of study and as it is a bit out of scope of this document, we will only present the simple naive algorithm here (Fig. 1.1). Interested readers can refer to [T+12; Pha+16; Vea97] for more details.

Path tracing algorithms [Pha+16] aim at solving the following equation, first devised by Kajiya in 1986,

$$I(\mathbf{x}) = L_e(\mathbf{x}) + \int_S \rho(\mathbf{p}, \mathbf{x}) I(\mathbf{p}) d\mathbf{p}. \tag{1.1}$$

Given here in its simplest form, this equation, known as the *Rendering Equation*, evaluates the amount of light I that reaches a point $\mathbf{x}$ on the image plane. To do so, it sums the intensity of the emitted light of $\mathbf{x}$ with the amount of light $\rho(\mathbf{p}, \mathbf{x})$ that each point $\mathbf{p}$ of the whole geometry surface S reflects towards to point $\mathbf{x}$. One of the difficulties with this equation is that $\rho(\mathbf{p}, \mathbf{x})$

is not a continuous function, it includes a visibility term, which is a binary factor with values being either 1 or 0, that accounts for the visibility discontinuities within the scene.

Note that light is composed as a set of wavelengths, which determines its color. Since the colors, materials, or geometry of the objects in the scene can affect each wavelength differently, this equation needs to be solved for each of them. However, this problem is out of the scope of this document and, to simplify the notation, we will assume that we are always working with a single given wavelength.

The issue that interests us most here, is that Equation (1.1 on the preceding page) relies on recursive evaluation of complex integrands. As we assume the scene to be a continuous signal, those integrands translate as infinite sums. It is thus impossible to evaluate them directly (or only in very specific cases [BN12]). Instead, we will rely on sampling-based numerical integration methods. More specifically, we will use Monte Carlo estimators which will be described in details in section 1.1.2 on the following page. For now, we just describe Monte Carlo estimators as a way to approximate the integrand of a function by evaluating N points of this function, and averaging them. Following this, Equation 1.1 on the preceding page becomes

$$I(\mathbf{x}) \approx L_e(\mathbf{x}) + \sum_{i=1}^{N} \rho(\mathbf{p}_i, \mathbf{x}) I(\mathbf{p}_i) \tag{1.2}$$

where $\mathbf{p}_i$ is the i^{th} sample. The quality of the approximation is now dependent on the error from the Monte Carlo estimators, which is related to the way the samples $\mathbf{p}_i$ are chosen (see Section 1.1.2 on the following page). Note that in this whole document, we only focus on unbiased estimators, meaning estimators that will converge to the true value of the integrand when using a infinite number of samples. Therefore we will often implicitly ignore bias. Note that other estimators could have been used, but we focus on estimators that do not assume prior knowledge of the function to integrate. For more details please refer to [Gla95; HH64].

To evaluate each $I(\mathbf{p}_i)$, we combine Monte Carlo estimators with *Path tracing* algorithms. This means that we represent our light field as a set of discrete light rays. Then, we evaluate the light field value for each ray by simulating how it interacts with the scene.In the real world, rays are coming from the light before reaching our eyes. In this algorithm, we do the opposite. The rays are thrown from the image plane and bounce within the scene either a finite amount of times or until they reach the light.

For each pixel of the image plane, we will sample a certain amount of points $\{\mathbf{p}_i\}$. Each $\mathbf{p}_i$ will be used as the starting point for a light ray. Each ray is then tested against the scene to compute the intersection point $\mathbf{p}_i^{'}$ between this ray and the geometry. Once this intersection point is found, we sample a bouncing direction of the ray and we compute the amount of light that bounces in this direction. To do so we use a Bidirectional Reflectance Distribution Function (BRDF). This function takes as input an incoming light direction (the direction of the ray) and an outgoing light direction (the sampled direction) and returns the amount of light that was reflected. This BRDF function is specific to each possible material of each object of the scene. Then, we once again test the bounced ray against the scene to find its new intersection point with the geometry, sample a bouncing direction, compute the reflected light, etc. This recursion ends when the ray hits a light source, or when a maximal number of recursion is achieved (Fig. 1.1 on page 2).

Following this recursive algorithm, the evaluation of a single sample can become quite costly. The challenge is thus to have an approximation as good a possible with as little samples as possible. This is the focus of the next section, along with the theory of Monte Carlo estimators.

1.1.2 Monte Carlo estimators

Monte Carlo estimators are used to numerically approximate a complex integrand. They rely on a set of samples, that are used to evaluate the function to integrate. Then, the values of this function for each sample are averaged to give an approximation of the integration.

More formally, to use classic Monte Carlo estimators, we first need to generate n independent, uniformly distributed random variables $\{\mathbf{x}^{(i)}\}_n$ in Ω, that we will refer to as *samples*. Then, an estimation of the mean value of a function f can be obtained via the unbiased estimator:

$$\mathcal{I}_n := \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}^{(i)}). \tag{1.3}$$

Note that this function f is defined on a s-dimensional domain Ω. Thus, all the samples $\mathbf{x}^{(i)}$ must be generated within this same domain (following [Gla95] for notations). In most cases, $\Omega = [0,1)^s$ but this is not mandatory.

The integration error Δ when using such an estimator is defined as

$$\Delta := |\mathcal{I}_n - \mathcal{I}| \tag{1.4}$$

where $\mathcal{I} := \int_\Omega f(x)dx$ is the true integration value. Note that this error depends on the sampling pattern used, the number of samples used, and the integrand function.

For stochastic samplers, we often use instead of this error the Mean Square Error (MSE), that will be defined just a bit below. First, we denote

$$Bias := |\mathcal{I} - \langle \mathcal{I}_n \rangle\,| \tag{1.5}$$

and

$$Variance := \left\langle (\mathcal{I}_n)^2 \right\rangle - \langle \mathcal{I}_n \rangle^2 \tag{1.6}$$

where $\langle . \rangle$ represents the expectation of a random variable.

From [Dur11; SK13], the MSE can then be expressed as

$$\text{MSE}(\mathcal{I}_n) = \text{Bias}^2 + \text{Variance}. \tag{1.7}$$

In this thesis we only work with unbiased stochastic point sets, our MSE depends thus entirely on the variance of $\mathcal{I}_n$. For classical Monte Carlo estimator with pure random samples, this variance is

$$\sigma^2 := \frac{\sigma_f}{n}, \tag{1.8}$$

where σ_f is the variance of the integrated function f. As the variance is function of the number of samples, the more samples we use the less error we observe (for unbiased point sets).

With a MSE in $O(1/n)$, the error in integration Δ is in $O(1/\sqrt{n})$. Thus, to divide this error by 2, we need to use 4 times more samples, which is rather unsatisfying. Furthermore, even though when taking a infinite number of samples in a pure random way we are guaranteed that we will cover the whole domain uniformly, there is no guarantee, if unlucky, that when taking a given number n of samples, they will not all clump into a small portion of space. For all those reasons, using pure random samples is close to be the worst scenario for unbiased Monte Carlo integration. Therefore, several improvements were developed, reducing the variance of this estimator by alleviating this clumping.

One of the most popular is the use of stratified sampling (Section 3.1 on page 36). This sampling subdivides the domain into k equal regions and generates random samples within each region. This sampling has provably a better variance than pure random sampling [HH64], proportional to the number of regions k and the way they are chosen. The exact variance function when using stratified sampling is quite complex and thus we will not give it here. Interested readers can find it in [Gla95], Section 7.4.3, Equation (7.35).

Another improvement, which interests us more here, is the usage of samples that are no longer randomly chosen. By choosing such samples in order to guarantee an optimal covering of the domain, you may reduce the error of the approximation. This category of estimators are called *Quasi Monte Carlo* (QMC). We point that such estimators were not originally designed for computer graphics applications but for financial and simulations contexts.

When using deterministic sets of samples, we can no longer rely on the MSE value, as the approximation $\mathcal{I}_n$ is not longer a random variable. The quality of QMC estimators is thus only measured from their error, which is

$$\Delta = \frac{k\sqrt{\log(n)^b}}{n} \tag{1.9}$$

for constants b and k [Gla95].

However, even though such samplers lead to a lower error and a better approximation for the color of a single pixel, this is not enough to ensure a better final image. An image is composed from a whole set of pixels, and therefore from a whole set of integrands, all solved independently. The way those integrands are correlated or not also impacts the final rendering.

1.2 Sampling for Monte Carlo Rendering

There are two specificities when using Monte Carlo estimators for rendering. The first one is that neighbouring pixels should be decorrelated, to avoid structural artefacts in the final image (Fig. 1.2 on the next page). The second is that the integrand to solve is recursive, therefore, even for a single pixel, we need to solve several integrands all related to each other. It is interesting to note that in most situation, we are working with recursive 2-D integrands, since we are mostly sampling directions or points on a surface.

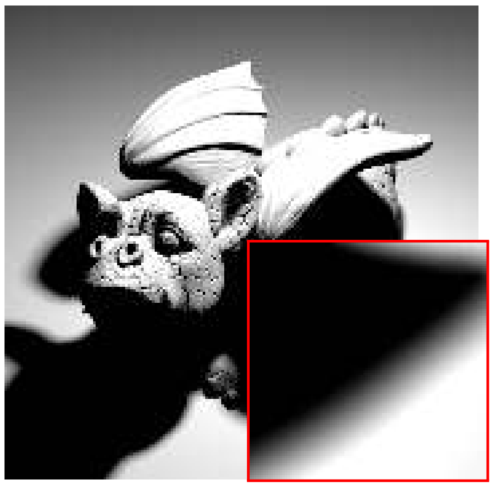
The structural artefacts appear mostly when using QMC estimators for rendering. QMC sampling patterns were originally devised for applications in finance and simulation. They have been proven to lead to a very low integration error, along with being easy to implement and computationally efficient, even in very high dimensions. Those properties are very often due to the analytical definition of such samplers. However, such analytical definition often implies that they present a recursive and regular spatial organisation of samples. This structure correlates the integration error over neighbouring pixel, which leads to disturbing structures in the final image. This is studied in more details in Section 2.1 on page 14 and in Section 3.3 on page 57. Instead, when using random samplers, this structure turns into noise [DW85], which is directly related to the variance in the estimator. It can never be completely removed but it can become imperceivable to the human eye, as the variance reduces (Fig. 1.2 on the facing page). Roughly, using QMC estimators leads to higher artefacts but a faster convergence and MC estimators presents a slower convergence but with lesser artefacts.

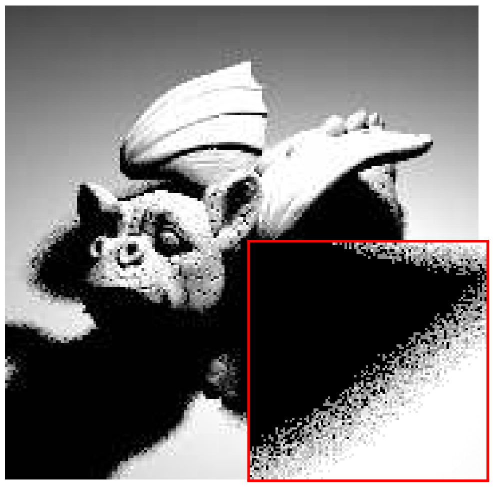
The second specificity of Monte Carlo Rendering is that even though we are solving a recursive multi dimensional integrand, not all dimensions are directly correlated. Assume that we are sampling a light ray that has N bounces. We will need one 2-D sample for the light plane. Then, we will need N 2-D sample for the bouncing directions. We can also need a 1-D sample for the motion blue, or another 2-D sample for the lens. The point is that we do not necessarily need a s-D sampling pattern. One can also use several 2-D sampling patterns, while being careful to decorrelate them and remove any structure. However, when using N 2-D sampling patterns, this decorrelation is simulated through random shuffling of samples, which harms the quality of the approximation of the integrand. This will be detailed in more specifics in Chapter 4 on page 74.

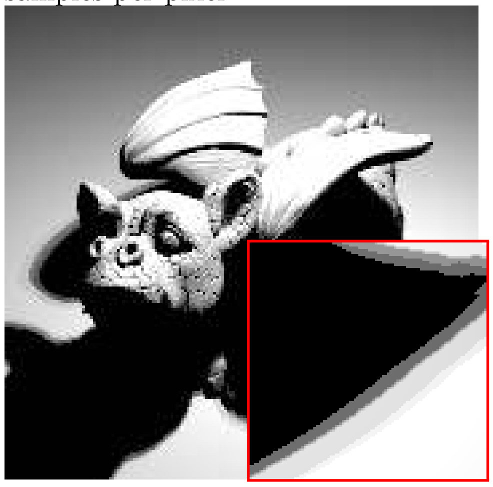
1.3 Contributions

In this PhD, we focused on developing proofs and algorithms to generate sets of s-D samples (also referred to as *point sets*, *distributions of samples*, or *sampling patterns*) that will be as optimal as possible regarding Monte Carlo rendering.

We tried to address 3 major issues,


(a) Image rendered with 512 random samples per pixel


(b) Noise artefacts from using 4 samples per pixel with a random sampler


(c) Aliasing artefacts from using the same 4 samples for each pixel

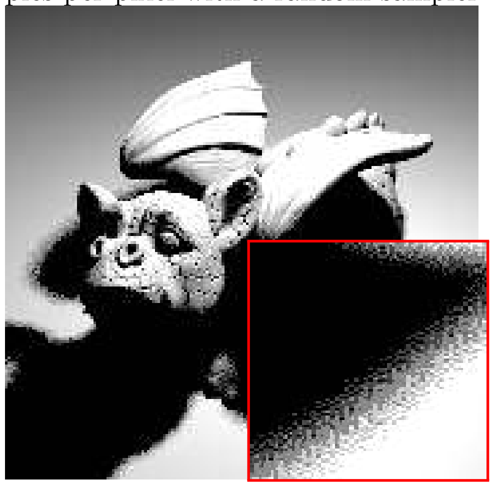

(d) Aliasing artefacts from using 4 samples per pixel with a structured sampler

Fig. 1.2.: Different artefacts observed when rendering a scene using different sampling strategies.

- Removing the bias in integration by guaranteeing that our point sets are uniformly distributed (Figure 1.2 on the preceding page (c)).

- Removing any structure in our spatial distributions to remove all *aliasing* in the final image (Figure 1.2 on the previous page (d)).

- Finally, reducing as much as possible the variance in Monte Carlo/Quasi Monte Carlo estimation (Figure 1.2 on the preceding page, (b)).

To assert all those issues, we tried to combine the uniformity and efficiency properties of QMC samplers with the irregular samples placement of stochastic point sets. Joining those properties apply additional constraints on the spatial samples organisation, which were though to be unreconcilable. We showed that they are in fact reconcilable, by providing several algorithms generating such point sets, along with their formal proofs. Some possible naive approaches to generate such samplers are presented in chapter 4 on page 74 and our main contributions are presented in chapter 5 on page 93 and in chapter 6 on page 107.

As a preliminary, Section 1.4 will present a few basic notations and definitions regarding point processes. Those definitions will be further developed in Chapter 2 on page 14 where we will see in deeper details how the quality of a sampling pattern regarding Monte Carlo Rendering can be assessed.

1.4 Basic definitions

When using a MC or QMC estimator, we need to generate a set of variables $\mathbf{x}^{(i)}$. We denote *point set* or *point pattern* a set P_n of n indexed s-D points

$$P_n = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(n)}\}$$

where all $\mathbf{x}^{(i)} \in \Omega$, with $\Omega \subset \mathbb{R}^s$ (Fig. 1.3 on the facing page). Note that as P_n is a set, we implicitly assume that there are no duplicates in this set, more formally, we assume that $\forall i, j$, $\mathbf{x}^{(i)} \neq \mathbf{x}^{(j)}$. Point sets are generated by *samplers*, which we will denote $\mathcal{S}$.

We denote $\mathcal{S}(\Omega)$ all point sets generated from the sampler $\mathcal{S}$ in the domain Ω. Note however, that in most rendering situations, the samples are defined in the unit hypercube $[0, 1)^s$, before being rescaled to the integration domain (which may be the surface of the light, a pixel, a direction hemisphere, ...). This is why all along this document, we will ease the notations by implicitly assuming that Ω is the unit hypercube (formally defined as $\Omega = [0, 1)^s$), unless specified otherwise.

Deterministic samplers generate from a given set of parameters a unique set P_n (note that the value n is a parameter of $\mathcal{S}$). Typically, such samplers are used with QMC estimators. On the contrary, stochastic samplers can generate several point sets from a given set of parameters. A point set P_n generated from a stochastic sampler is referred to as a realisation of this sampler. We assume that if we denote $\mathcal{A}$ the set of point patterns that present a given characteristics, we have $\mathcal{S}(\Omega) \in \mathcal{A}$, meaning that a stochastic sampler generates different sets but always with similar characteristics (see Chapter 2 on page 14 for a list of criteria that may characterize a sampling pattern). Such sets are typically used for MC estimators.

In her book, Illian [Ill+08] presents a thorough statistical analysis for samplers (which she refers to as *point processes*). We note that when $\mathcal{S}$ is stochastic, $\mathcal{S}(\Omega)$ is a random variable. For simplicity, we will assume that this is also true for deterministic samplers, with all realisations of $\mathcal{S}(\Omega)$ being equal for a given set of parameters. We can then differentiate different types of point processes. A point process can be *finite*, *infinite*, *isotropic*, *stationary* and/or *ergodic*.

Finite/Infinite point processes A sampler $\mathcal{S}$ that can generate any n number of samples over any domain Ω is an infinite point process. Finite point processes are point processes defined over a single given Ω. They are mostly related to real world situations, for example, the repartitions of the seeds around a tree, or the number of pores in a Swiss cheese.

Stationary and Isotropic point processes A point process is said to be *stationary* if it is invariant by translation, and isotropic if it is invariant by rotation. More formally, if we assume $\mathbf{P}$ a *probability measure*, $\mathcal{S}$ is stationary if $\forall \mathbf{x} \in \mathbb{R}^s$

$$\mathbf{P}(\mathcal{S}(\Omega)) = \mathbf{P}(\mathcal{S}(\Omega - \mathbf{x})) \tag{1.10}$$

and *isotropic* if $\forall \alpha \in [0, 2\pi]$

$$R_\alpha \mathcal{S}(\Omega) = \{R_\alpha \mathbf{x}^{(1)}, R_\alpha \mathbf{x}^{(2)}, \cdots, R_\alpha \mathbf{x}^{(n)}\} \tag{1.11}$$

and

$$\mathbf{P}(\mathcal{S}(\Omega)) = \mathbf{P}(R_\alpha(\mathcal{S}(\Omega))) \tag{1.12}$$

with R_α being the rotation matrix of angle α. Note that point sets organized on a regular lattice are not isotropic.

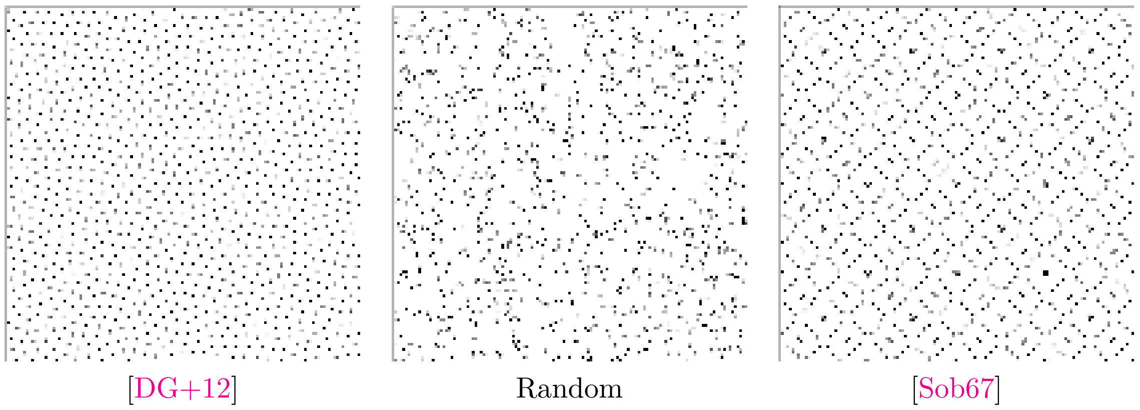


Fig. 1.3.: Various 2-D point sets P_{1024} generated from different samplers $\mathcal{S}$.

Density of a point processes We also define the *density* of a point set as the average number of samples inside a region R of volume V_R around a sample $\mathbf{x}$.

$$\lambda(x) = \frac{R(x)}{V_R} \tag{1.13}$$

This density is constant for isotropic and stationary point processes. A sampler generating sets with a non constant density is sometimes called a *non-uniform* sampler.

Ergodic point processes The ergodicity of a sampler $\mathcal{S}$ refers to samplers that are so statistically invariant that the analysis of a single realisation is statistically meaningful. We point out that a deterministic sampler is necessarily ergodic.

Sequentiality We will end this section by defining the notion of sequentiality for a sampler. A sampler is said to be a *sequential* sampler if it does not require a prior knowledge of the number of samples to generate. Such samplers are able to iteratively add n new samples within an already existing set, while preserving the characteristics of the set. We denote *sequence* a sampler that is both sequential and present highly uniformly distributed samples (low discrepancy). We will see how we can characterize this uniformity in Chapter 2 on page 14.

Finally, as many of the samplers we will study rely on binary manipulations over positive integers, we define as

$$x = \sum_{k=0}^{N} a_k b^k \tag{1.14}$$

the representation of a integer $x \in \mathbb{N}$ in base b, where $a_k \in \mathbb{Z}/\mathbb{Z}$. Each a_k value will be denoted a *digit* of x, or a *bit* of x if $b = 2$. The value N is the number of digits needed to encode x in base b. Note that each a_k value belongs to the domain $\mathbb{Z}/b\mathbb{Z}$. For simplicity, we will also denote x from the list of digits of its b-ary representation as

$$x = (a_N, \cdots, a_0)_b. \tag{1.15}$$

Similarly, if a value x belongs to the set $[0, 1)$, it will be denoted as

$$x = \sum_{k=0}^{N} a_k b^{-k-1}, \tag{1.16}$$

and

$$x = (.a_0, \cdots, a_N)_b. \tag{1.17}$$

Finally, we will denote $(a_N, \cdots, a_0)$ or $(a_0, \cdots, a_N)$ the vectors made from the list of digits of a value x in base b.

Note that any value $x \in \mathbb{N}$ can be turned into its equivalent $y \in [0, 1)$ (and vice versa) with the following operations

$$y = x/b^N \tag{1.18}$$

$$x = y \cdot b^N. \tag{1.19}$$

This means that all those notations are roughly equivalent, and in this document, when there is no ambiguity, we will often implicitly switch between them. Table 1.1 on the next page summarizes all the notations that will be used in this document.

In the rest of this document, we will rely on those definitions to present the different characteristics of a set of samples that impact the quality of the final rendered image (Chapter 2 on page 14). We will then present existing samplers (Chapter 3 on page 35), which will let us underline where do our contributions fit in (Chapter 4 on page 74, Chapter 5 on page 93, and Chapter 6 on page 107). Finally, we will present how those samplers behave in practice through experimental results (Chapter 7 on page 133). Note that is this PhD, we study both sequential and non-sequential samplers. However, only infinite and stationary point processes are considered, with most of them being also isotropic.

Chapter 1	
s	The total number of dimensions
$\mathbf{x}$	A point in s-D
$\mathbf{x}^{(i)}$	The i^{th} sample of a set P_n
x_i	The i^{th} coordinate of $\mathbf{x}$
$x_j^{(i)}$	The j^{th} coordinate of $\mathbf{x}^{(i)}$
P_n	A point set of size n
Ω	The domain of definition of $\mathcal{S}$
$\mathcal{S}(\Omega)$	A point set generated from the sampler $\mathcal{S}$
Chapter 2	
$\mathcal{F}_f$	The Fourier transform of a function f
$\mathcal{F}_f^{-1}$	The inverse Fourier transform of a spectrum f
$\mathcal{P}_f$	The Power spectrum of f
$\mathcal{D}_*$	The l_∞ discrepancy
$\mathcal{D}$	The l_∞ extreme discrepancy
$\mathcal{D}_o$	The l_∞ isotropic discrepancy
$\mathcal{T}_2$	The l_2 discrepancy
$\mathcal{D}_2$	The generalized l_2 discrepancy
Chapter 3	
$(a_N \cdots a_0)_b$	The b-ary representation on a value $x \in \mathbb{N}$
$(.a_0 \cdots a_N)_b$	The b-ary representation on a value $x \in [0,1)$
$(a_N \cdots a_0)$	A vector of values with $a_k \in \mathbb{Z}/b\mathbb{Z}$

Tab. 1.1.: Table of the notations we will use throughout this document.

2 Characterizing a set of samples

There are several ways to characterize a point set. Depending on the application, one can focus on visual quality, stochasticity, ease to control, etc. In our case, we will use those sets for Monte Carlo rendering. We will thus characterize our samplers with characteristics related to the integration error of their MC estimator. This leads us to three main characteristics. First, the uniformity of the point set (Section 2.1), second, its Fourier spectrum, (Section 2.2 on page 20) and third, its spatial organisation (Section 2.3 on page 25). We chose those measures specifically, since they are known to be related to the integration error and integration variance (Section 2.4 on page 28), and therefore are related to the expected quality of the final rendered image.

2.1 Uniformity

A set of samples can be characterized by how close it is from the perfect uniform distribution. In other words, how well the samples are spread out over a given domain. There are several ways to measure this uniformity.

Some tests rely on a supposed lattice structure of the distribution, like the *spectral test* [CM67]. This test, defined only for lattices, measures the largest distance d_s between adjacent parallel hyperplanes that together cover the whole point set. The smaller this distance, the better the uniformity of the lattice (Fig. 2.1).

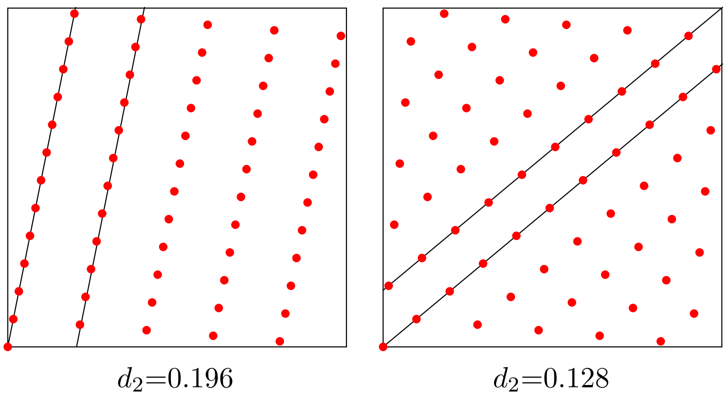


Fig. 2.1.: Spectral test in 2-D on Korobov lattices. Figure from [Lem09]

Other tests apply on point sets quantized on a grid of a given resolution (Figure 2.2 on the next page left), namely the test of *Congruent Boxes* (Figure 2.2 on the facing page right). Roughly, the domain is partitioned according to a grid in base b, and if each box of this grid contains the same number of samples, the point set is considered uniform [Lem09]. This test is used to determine if a point set is a *digital net*. Such nets will be of importance and will be presented in more details in Section 3.3.2 on page 59.

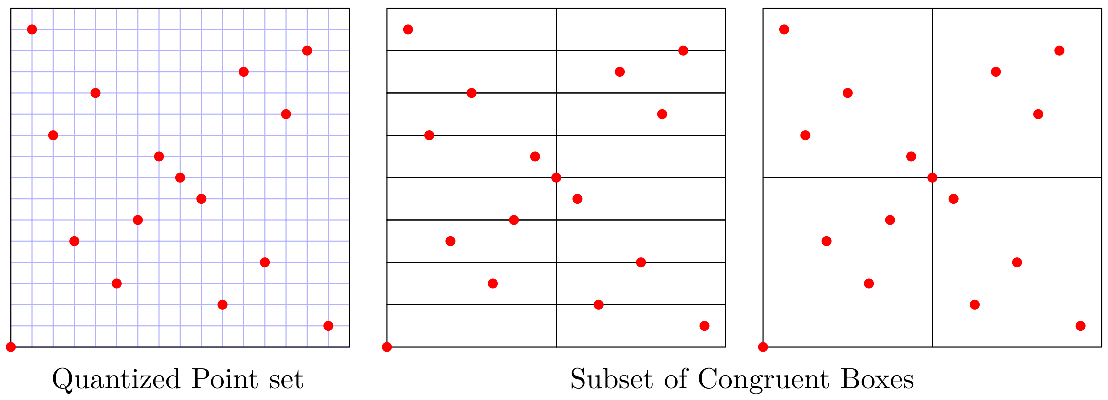


Fig. 2.2.: Congruent boxes in 2-D on the Sobol sequence [Sob67]

The most common measurement of uniformity is the *discrepancy* of a point set. One of the main asset of this measure is that it is not specific to a type of point set. It quantifies the distance between a distribution of samples and the uniform distribution. In 1-D, we compute the distances between the Cumulative Distribution Functions (CDF) of the uniform distribution, and the CDF of a given 1-D point set P_n, both defined in the domain $[a, b)$. Note that in most situations, we have $a = 0$ and $b = 1$. The CDF of a point set (Fig. 2.3 on the following page) is defined as

$$F_n(u) := \frac{1}{n} Card(\{x, x \in P_n, x < u\}) \tag{2.1}$$

with $u \in [a, b)$.

The uniform distribution, denoted $\mathcal{U}$ is defined as the distribution where each possible random variable is equally probable. A 1-D illustration of the distribution $\mathcal{U}(a, b)$, which is the uniform distribution of all the values in $[a, b)$, is given Figure 2.4 on the next page.

We quantify the l_∞ distance between the distribution of a given point set to $\mathcal{U}$ using the Kolmogorov-Smirnov test. The distance $N(P_n)$ between a point set P_n and the uniform distribution is then

$$N(P_n) := \sup_u |F_n(u) - F(u)| \tag{2.2}$$

where $\sup_u$ returns the maximum distance and where $F(u)$ is the CDF of the uniform distribution (Fig. 2.5 on the following page). Note that we could also quantify similarly the l_p distance $\mathcal{D}_p(P_n)$ between the distribution of our point set and $\mathcal{U}$,

$$N(P_n) := \sum_x |F_n(x) - F(x)|^p. \tag{2.3}$$

Those definitions apply to 1-D point sets. We will see in the following sections the definition of the l_∞ star discrepancy (also referred to as *star discrepancy* [Nie92]) for 2-D point sets, and the l_2 star discrepancy for s-D point sets.

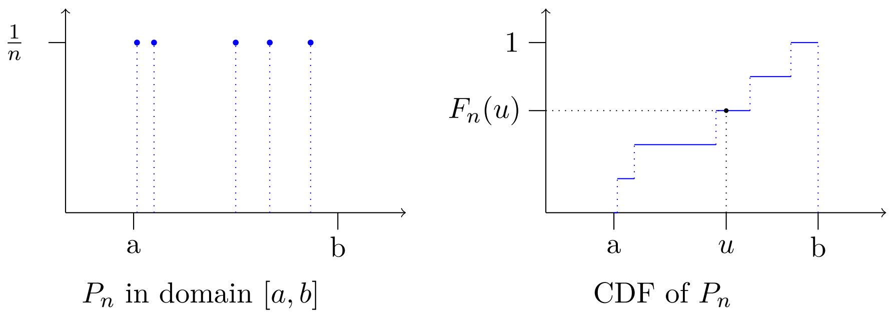


Fig. 2.3.: The CDF of a pointset P_n

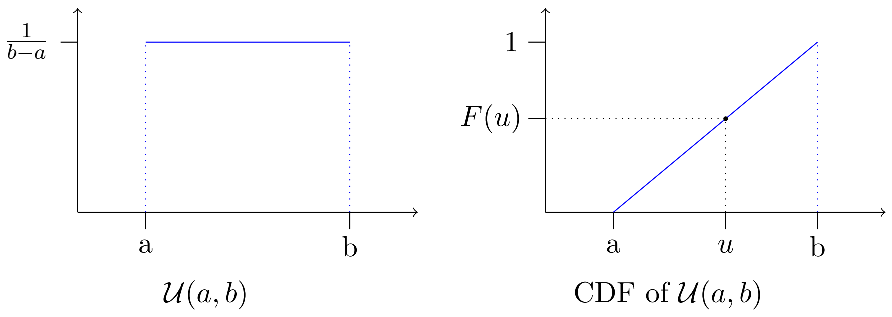


Fig. 2.4.: The Uniform distribution $\mathcal{U}(a, b)$

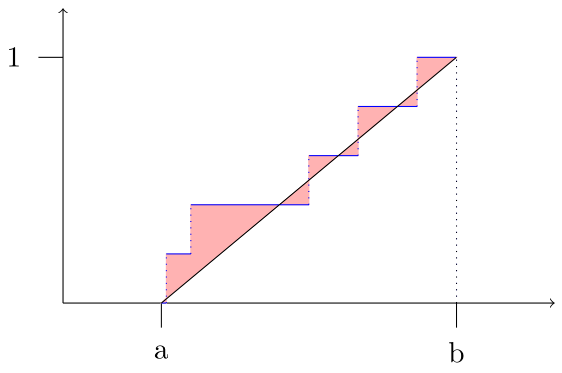


Fig. 2.5.: The Kolmogorov-Smirnov difference (in red) between the CDF of a point set (in blue) and the CDF of the uniform distribution (in black).

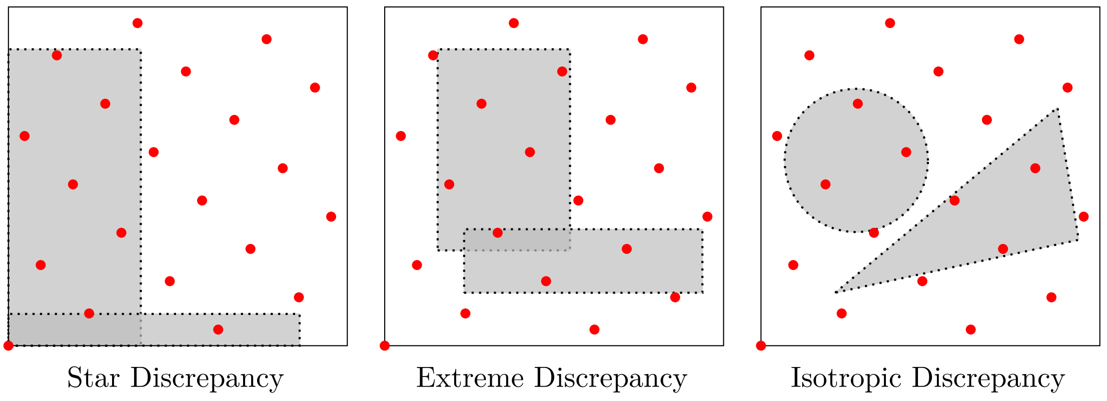


Fig. 2.6.: Different measures for discrepancy in 2-D.

2.1.1 l_∞ discrepancy

One of the most common of all the existing discrepancy measure is the l_∞ *star discrepancy*. This measure considers all axis-parallel hyper-rectangles $B(\mathbf{v})$ in the domain $[0,1)^s$ with a corner at the origin of the domain and the other one at position $\mathbf{v}$ (Figure 2.6, left). We call such hyper rectangles *anchored boxes*. We then define the function

$$\alpha(P_n, \mathbf{v}) := \frac{Card(P_n \cap B(\mathbf{v}))}{n} \tag{2.4}$$

that counts the number of samples that fall inside an hyper-rectangle and expresses it as a fraction of the total number of samples n. We then compare it to the true area of $B(\mathbf{v})$, $v_1 \cdots v_s$.

This leads us to the final formula for the l_∞ star discrepancy,

$$\mathcal{D}_*(P_n) := \sup_{\mathbf{v}\in[0,1)^s} |v_1 \cdots v_s - \alpha(P_n, \mathbf{v})|. \tag{2.5}$$

One can note the existence of other discrepancy measures than the l_∞ star discrepancy. First, the l_∞ *extreme discrepancy* that computes the discrepancy using unanchored axis-parallel hyper rectangles (Figure 2.6, middle). Note that the following relationship holds between the l_∞ star discrepancy $\mathcal{D}_*(P_n)$ and the l_∞ extreme discrepancy $\mathcal{D}(P_n)$ [KN74]:

$$\mathcal{D}_*(P_n) \leq \mathcal{D}(P_n) \leq 2^s \mathcal{D}_*(P_n). \tag{2.6}$$

Finally, it is worth noting the existence of the l_∞ *isotropic discrepancy* that computes the discrepancy using any convex shape (Figure 2.6, right). The following relationship holds between the l_∞ isotropic discrepancy $\mathcal{D}_o(P_n)$ and the l_∞ extreme discrepancy [KN74]:

$$\mathcal{D}(P_n) \leq \mathcal{D}_o(P_n) \leq (4s\sqrt{s}+1)\mathcal{D}(P_n)^{\frac{1}{s}}. \tag{2.7}$$

As the extreme and isotropic discrepancies are extremely difficult to compute and are both related to the star discrepancy, we will not discuss them in details here.

Measuring the discrepancy of a point set characterizes the uniformity of this single set. However, a careful reader will have noted that, aside from pathological samplers, the discrepancy of a set decreases as the size of the point set increases. We can use this property to characterizes a sampler with the "speed" at which its discrepancy decreases. For example, for a pure random sampler in 2-D, the l_∞ star discrepancy decreases at a speed of $\frac{1}{\sqrt{n}}$, when for the regular grid, it decreases in $1-(1-\frac{1}{n})^2$ [Lem09; Nie92]. The following lower bound on the l_∞ star discrepancy has been shown for $s = 2$ [Sch72]:

$$\mathcal{D}_*(P_n) \geq C\frac{log(n)^{s-1}}{n}, \tag{2.8}$$

where C is a constant dependant on s but independent of n [Nie92].

If you have a sampler with a discrepancy decreasing in $O\left(\frac{log(n)^{s-1}}{n}\right)$ as the number of samples n increases, this sampler is said to be a *low discrepancy sampler*. If this sampler is also sequential, we refer to it as a *sequence*. The l_∞ star discrepancy is defined using axis-parallel hyper rectangles anchored at the origin (Figure 2.6 on the preceding page, left). This means that when two samples have the same coordinate on either axis, a box of infinitely small size can contain several samples. This almost guarantees to break the low discrepancy property. Therefore, one of the first conditions (necessary but not sufficient) for a point set to be low discrepancy is that it never presents several samples aligned on either dimensions.

Computing the l_∞ star discrepancy can be done using explicit formulas for $s = 1$

$$\mathcal{D}_*(P_n) = \frac{1}{2n} + \max_{1\leq i\leq n}\left|x^{(i)} - \frac{2i-1}{2n}\right| \tag{2.9}$$

and for $s = 2$, as detailed in [Nie72] (note that similar formulas exist for the extreme discrepancy).

It has been observed that the set of hyper-rectangles to consider can be discretized (Fig. 2.7 on the next page). Following the notations in [Gne+11], we denote

$$\mathbf{X}_j(P_n) := \{x_j^{(i)}, i \in \{1, \cdots, n\}\},$$

the set containing all the j^{th} coordinates of a set P_n. We also denote

$$\mathbf{X}(P_n) := \mathbf{X}_1 \times \cdots \times \mathbf{X}_s,$$

the set of all the points that can be made from all the coordinates of all the samples of P_n. Now we can redefine the l_∞ star discrepancy as

$$\mathcal{D}_*(P_n) = \sup_{\mathbf{v}\in\mathbf{X}(P_n)} |v_1 \cdots v_s - \alpha(P_n, \mathbf{v})| \tag{2.10}$$

which can be computed exactly using an enumeration algorithm in $O(n^s)$ [Nie72] (Fig. 2.7 on the facing page). We stress that such an algorithm is only practicable for small values of n and/or s. The computation of the star discrepancy has been proven to be a NP-hard problem

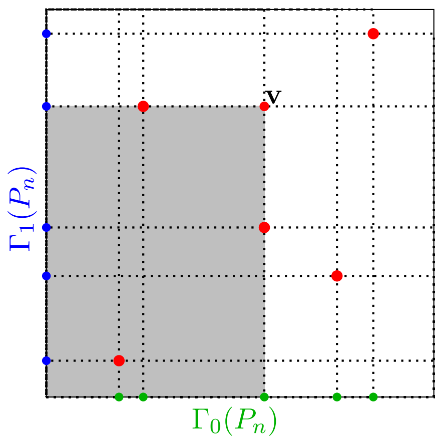


Fig. 2.7.: Computing the discrete l_∞ star discrepancy in 2-D.

in higher dimensions [Gne+08]. Note that the best known algorithm to compute the l_∞ star discrepancy is in $O(n^{1+s/2})$ [Dob+96] and that several algorithms exist to approximate it in s-D [Thi01; Gne+11; Sha10].

This discrepancy is not the only defined discrepancy measure, one can also compute l_p star discrepancies. We find that the l_2 star discrepancy can be computed exactly in $O(sn^2)$, which is useful as s increases.

2.1.2 l_2 discrepancy

The l_2 star discrepancy, denoted $\mathcal{T}_2$, is a variation of the l_∞ star discrepancy. Instead of using the l_∞ norm (sup), we use the l_2 norm :

$$\mathcal{T}_2(P_n) := \left(\int_{[0,1)^s} (|v_1 \cdots v_s - \alpha(P_n, \mathbf{v})|)^2 d\mathbf{v} \right)^{\frac{1}{2}}. \tag{2.11}$$

It should be used carefully as it is known to both put an emphasis on points close to 0 and to be irrelevant for sets that contains too few samples relatively to the number of dimensions [Mat98]. However, it can be computed very easily following Warnock's formula [War73],

$$(\mathcal{T}_2(P_n))^2 = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \prod_{k=1}^{s} \left(1 - \max\left(x_k^{(i)}, x_k^{(j)}\right)\right) - \frac{2^{-s+1}}{n} \sum_{i=1}^{n} \prod_{k=1}^{s} \left(1 - \left(x_k^{(i)}\right)^2\right) + 3^{-s}, \tag{2.12}$$

where $x_k^{(i)}$ is the k^{th} coordinate of the i^{th} sample in the set. This gives us an algorithm in $O(sn^2)$ to compute $\mathcal{T}_2$.

Several versions on the l_2 star discrepancy were defined [Hic98], each of them with a formula similar to Warnock's. Among them, the one that interests us here is the *generalized l_2 star discrepancy*, defined by Hickernell [Hic98] with the following formula

$$(\mathcal{D}_2(P_n))^2 = \left(\frac{4}{3}\right)^s - \frac{2}{n}\sum_{i=1}^{n}\prod_{j=1}^{s}\frac{3-\left(x_j^{(i)}\right)^2}{2} + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\prod_{j=1}^{s}\left[2-\max\left(x_j^{(i)}, x_j^{(i')}\right)\right] \qquad (2.13)$$

This value is of special interest since it is related to the integration error, as will be discussed in section 2.4.1 on page 28. The generalized l_2 star discrepancy can also be computed in $O(sn^2)$. This makes it much more efficient in higher dimensions than the l_∞ star discrepancy, that has a complexity in $O(n^{1+s/2})$ [Dob+96].

For all those reasons, when evaluating the uniformity of higher dimensional pointsets (in practice, for all sets with $s > 2$), we rely on the generalized l_2 star discrepancy.

Those discrepancies measures are of uttermost importance in Monte Carlo integration. If a sampler is not uniform, meaning if it does not verify

$$\lim_{n\to\infty} \mathcal{D}_*(P_n) = 0,$$

it is not guaranteed to converge to the correct integrand value when used with Monte Carlo estimators. However, even if it is uniformly distributed, having a structured sampling pattern may lead to correlation between neighbouring samples, which leads to aliasing in the final renderings. To characterize the amount of aliasing generated from using a particular sampling pattern in rendering, we rely on spectral and spatial analysis (Section 2.2, and Section 2.3 on page 25).

2.2 Spectral analysis

A sampler can also be characterized by how the samples are spatially organized, and more specifically, by which frequencies arise in this distribution. First we will see how to compute the frequency representation of a signal and then, we will see how we can characterize a point set from this representation. Note that in [BB86], interested readers can find a much more detailed description of all the tools presented here.

2.2.1 The Fourier transform

There are two mains representations for a signal. The most common is the spatial representation, which associate to a value $x \in \Omega$ a value $y \in f(x)$, with x being the time or a position in space, and y the function value. However, it appears that many real-valued signals can be represented by a combination of sinusoids with different phases, amplitudes and frequencies. Such representation of a signal is called its *frequency*, or *spectral representation*. The transformation from the spatial representation to the frequency representation of a signal is given by the *Fourier Transform*.

The representation of signals as a set of sinusoids is referred to as the *Fourier series*, defined in 1-D by

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cos(xn\omega) + \sum_{n=1}^{\infty} b_n \sin(xn\omega) \tag{2.14}$$

where the a_n and b_n, that control the amplitude of the sinusoids, are defined in the complex domain $\mathbb{C}$ and where $n\omega$ is the frequency of the sinusoids. Thus, the Fourier series can also be expressed as a sum of complex exponentials

$$f(x) = \sum_{n=0}^{\infty} c_n e^{ixn\omega}. \tag{2.15}$$

Those Fourier series assume that the represented signal is periodic. For non periodic signals, we will rather use the *Fourier transform*. This operation translates from the spatial to the frequency representation of f. This transform associates to a frequency ω its amplitude and phase in the Fourier serie decomposition of f. It is given by the following equation in 1-D,

$$\mathcal{F}_f(\omega) = \int_{\mathbb{R}} f(x) e^{-ix\omega} dx \tag{2.16}$$

for a function $f(x)$, defined over $\mathbb{R}$, with f presenting frequencies over the domain Ω. The real part of $\mathcal{F}_f(\omega)$ represents the amplitude of the sinusoid and its imaginary part the phase of the signal f at frequency ω. Note that

$$\mathcal{F}_f(0) = \int_{\mathbb{R}} f(x) dx. \tag{2.17}$$

Figure 2.8 on the following page illustrates the relationship between the Fourier transform and the Fourier serie of a signal. In the following, we will often refer to $\mathcal{F}_f(\omega)$ as the *spectrum* of a signal f.

Note that the *Inverse Fourier transform* can transform back the spectrum of f into the original signal f as

$$f(x) = \int_{\mathbb{C}} \mathcal{F}_f(\omega) e^{ix\omega} d\omega \tag{2.18}$$

The existence of this function allows to say that the frequency and spatial representations of a signal are dual.

For simplicity, we only gave formulas and examples for the 1-D Fourier transform. However, the Fourier transform is defined in s dimensions. A s-D signal presents a s-D spectrum, computed with similar equations. You just replace the scalar products between x and ω with s-D dot products in Equations (2.16) and 2.18.

However, we point out that for practical applications, we handle signals that are defined over a closed domain Ω. Therefore, to apply a Fourier transform to our point sets, we implicitly assume

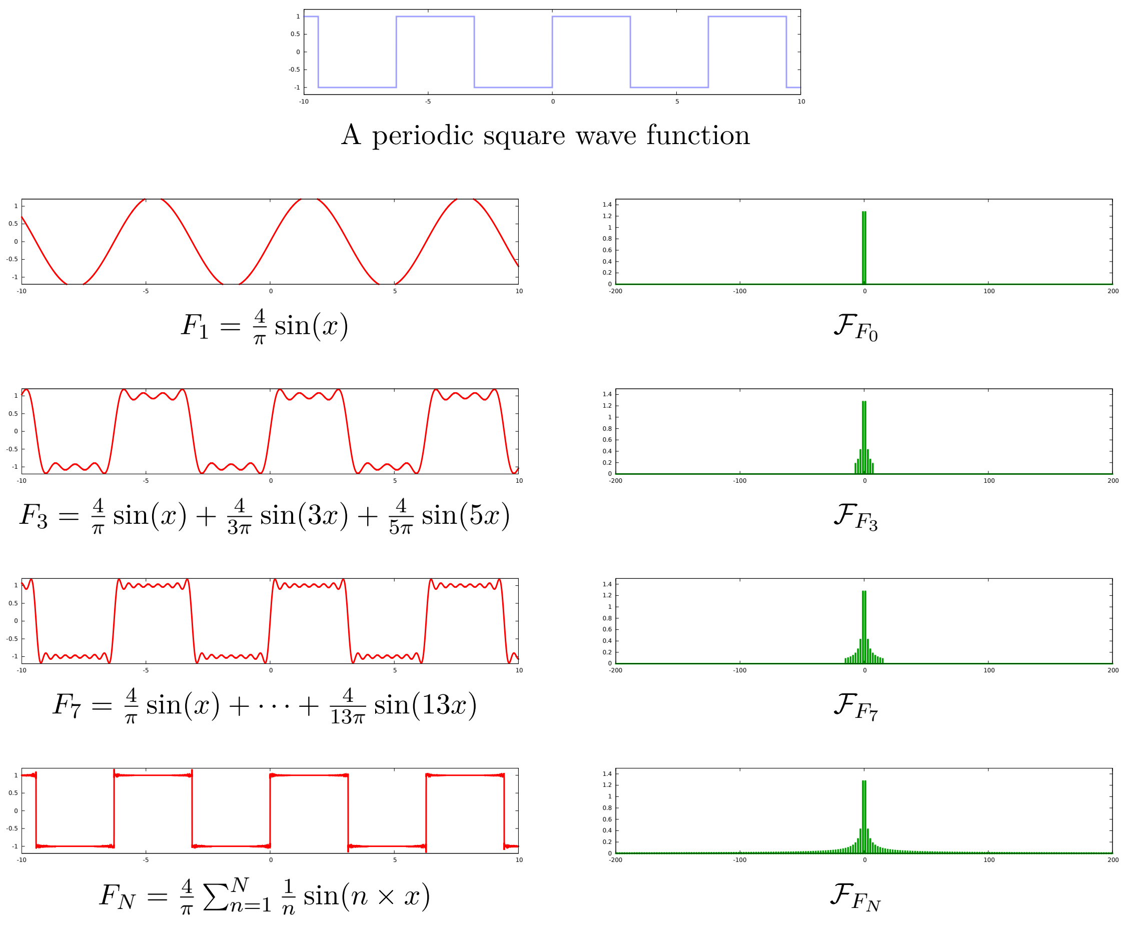


Fig. 2.8.: Fourier Serie decompositions F_N of the square wave function and their corresponding Fourier transforms.

that they repeat periodically (to avoid oscillation in the Fourier transform due to border effects). Furthermore, we handle only a discrete representation of our signals, therefore, in practice, we perform discrete and periodic Fourier transforms, which are equivalent to Fourier series.

2.2.2 The Power Spectrum

Here, we want to characterize our point sets by the amount of energy of each frequencies. Therefore, what interests us is the amplitude of each sinusoids in the decomposition of our s-D point set P_n. We denote by $\mathcal{P}_{P_n}$ the *Power spectrum* of a point set. $\mathcal{P}_{P_n}$ is formally defined as

$$\mathcal{P}_{P_n}(\omega) = |\mathcal{F}_{P_n}(\omega)|^2. \tag{2.19}$$

Using the norm of $\mathcal{F}_{P_n}(\omega)$ removes the imaginary part and preserves only the squared amplitude. This results in having all the values of $\mathcal{P}_{P_n}$ in $\mathbb{R}$, with the removal of the phase information that was contained in the imaginary part of $\mathcal{F}_{P_n}$.

A s-D point set, leads to a s-D power spectrum. So, for simplicity, we will often use its radial representation instead. This representation averages all the values of the Power spectrum between two s-D spheres, turning a s-D representation to a 1-D representation of our Power spectrum. This radial average is only truly valid for isotropic spectra. Still, for anisotropic spectra, it can still be used to identify peaks in the Power spectrum, a.k.a. frequencies that are so present that they are almost unaffected by averaging. All of this is illustrated Figure 2.9 on the next page.

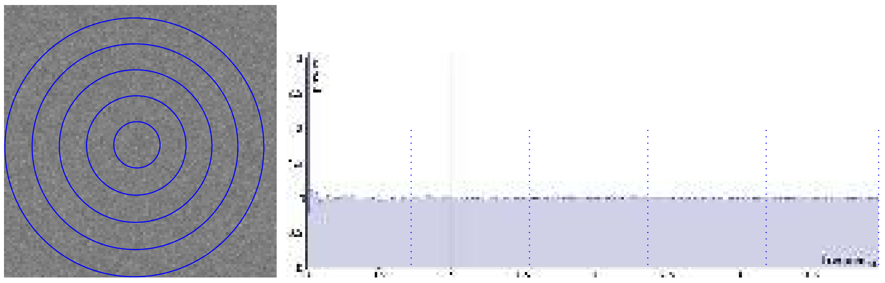
2.2.3 Differential Analysis

Differential analysis is a tool was presented in [WW11], and that can be thought of as a reformulation of the classical spectral analysis. It lets us compute the spectral properties of sets of uniform and non-uniform density by reformulating the spectral analysis as a difference between pairs of samples.

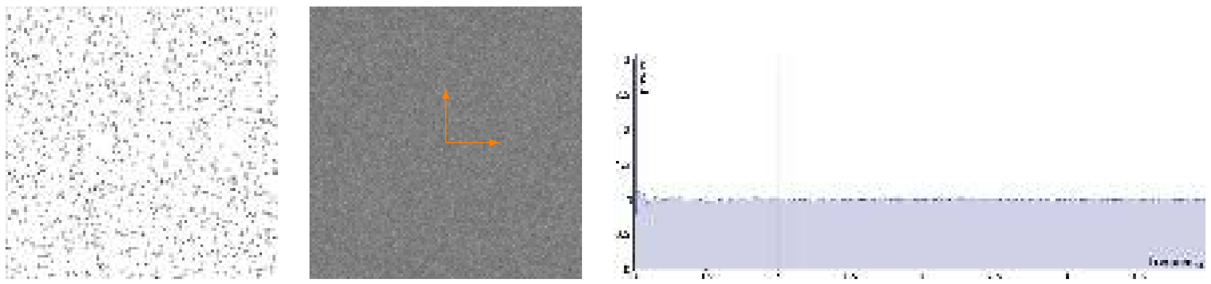
The Fourier analysis is originally expressed in terms of the absolute coordinates of samples. When the samples are distributed non-uniformly, or on a domain that is not a unit-hypercube, such analysis is difficult to carry. Differential analysis is instead expressed in terms of relative positions between two samples. By matching the distance function used to measure the distance between samples to the domain or the density of this distribution, it can produce a pertinent spectral analysis from non uniform sampling patterns or non Euclidean domains.

This differential formulation can be directly derived from the formulation of the Fourier transform for a set of samples. Formally, if we formulate the Discrete Fourier transform of a point set P_n as follows

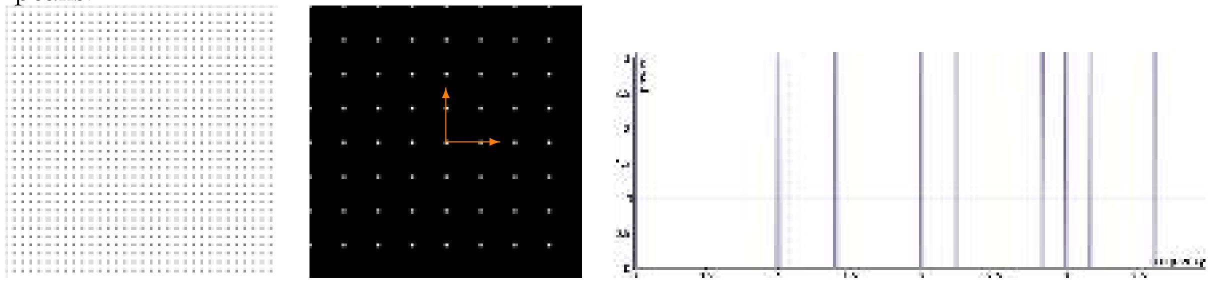
$$\mathcal{F}_\omega = \sum_{k=0}^{n-1} e^{-2\pi i(\omega \cdot \mathbf{x}^{(k)})} \tag{2.20}$$



The radial average is computed by averaging all the spectrum values between two s-D spheres, thus leading to a 1-D representation of the original s-D spectrum.



A purely random pointset (right), where every frequency appears. Its power spectrum (middle) is isotropic as all frequency appear equally, and its radial average (left) confirms the absence of peaks.



A pointset (right) with a finite number of frequency. Its power spectrum (middle) is clearly non isotropic (and was enhanced to make the peaks visible) but when you look at its radial average (left), the recurrent frquencies still appear clearly

Fig. 2.9.: Power spectrum and Radial average characterization.

where $\mathbf{x}^{(k)}$ is the k^{th} sample from P_n, ω is the frequency vector and $\cdot$ is the inner dot product. This leads to the following formulation for the Power Fourier Spectrum

$$\mathcal{P}_\omega = |\mathcal{F}_\omega|^2 = \mathcal{P}^r_\omega + \mathcal{P}^i_\omega \tag{2.21}$$

where $\mathcal{P}^r_\omega$ (resp. $\mathcal{P}^i_{\mathbf{f}}$) is the real part (resp. the imaginary part) of $\mathcal{P}_\omega$.

$$\mathcal{P}^r_\omega = \frac{1}{N}\left(\sum_{k=0}^{n-1} cos(2\pi i(\omega \cdot \mathbf{x}^{(k)}))\right)^2 \tag{2.22}$$

$$\mathcal{P}^i_\omega = \frac{1}{N}\left(\sum_{k=0}^{n-1} sin(2\pi i(\omega \cdot \mathbf{x}^{(k)}))\right)^2 . \tag{2.23}$$

Using the sum rules of trigonometry, this leads to

$$\mathcal{P}_\omega = \frac{1}{N}\sum_{j,k=0}^{n-1} cos(2\pi i(\omega \cdot (\mathbf{x}^{(k)} - \mathbf{x}^{(j)}))) \tag{2.24}$$

where $\mathcal{P}_\omega$ only depends on the differential between the pairs of samples $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(j)}$.

Therefore, using this type of analysis is just as discriminative as using traditional Fourier analysis. However, note that it applies efficiently only on domains where the distance function between pairs of samples can be properly defined for the entire domain (for example using geodesics for surfaces, of weighting the distance by the inverse density function for adaptive sampling).

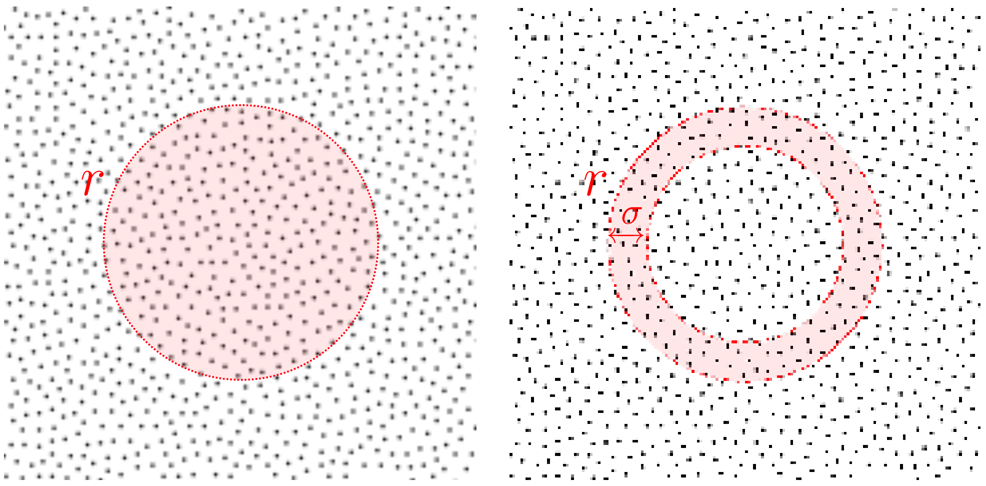
This type of analysis prooves that the spatial organization of samples can be directly related to the spectral properties of this distribution. Furthermore, the results of spatial analysis can be strictly equivalent to the results of spectral analysis, when using proper tools, as presented in the next section.

2.3 Spatial Analysis

This spatial analysis of point sets is closely related to the spectral analysis. The simplest measure in spatial analysis is the *minimal distance* d (usually l_2 Euclidean distance) between two samples in a point set P_n,

$$d(x, x') := \min_{x,x' \in P_n} ||x - x'||^2. \tag{2.25}$$

However, this measure, as simple as it is is not very useful. Instead, we will rely on the distribution of distances between pairs of points, called the *pair correlation function*.



Ripley's K function sums the average number of samples at a distance $d < r$ from a typical sample

The PCF sums the average number of samples at a distance $d \approx r$ from a typical sample

Fig. 2.10.: Illustration of Ripley's K function and of the PCF

2.3.1 Pair Correlation Function

In this whole section, we assume that our point set P_n is stationary (see Section 1.4 on page 8). Before presenting the *Pair Correlation Function* (PCF), we will start by presenting two other second order estimators, known as the K-function and the L-function. For the notations, we will follow the choices made by [Ill+08].

We will start by denoting $N^{(i)}(r)$, the function that estimates the number of samples in a radius r from a given sample $\mathbf{x}^{(i)}$ (Fig. 2.10). This function in itself does not convey much information. However, it will be the basis for all second order spectral analysis functions, such as *Ripley's K-function*, the *L-function* or the *Pair Correlation Function* (PCF).

Ripley's K-function represents the average number of points that are at a distance $d < r$ from a typical point. If we denote this value $K(r)$, we have for a point set P_n,

$$K(r) := \frac{1}{\lambda n} \sum_{i=0}^{n} N^{(i)}(r) \qquad (2.26)$$

where λ is the density of samples described in 1.4 on page 8. The normalization of $K(r)$ by λ, allows K to be independent from the global density and thus to be more sensitive to local fluctuations.

This K-function relates the number of samples to the area of a ball of radius r. To alleviate this and relate instead the number of samples to r itself, one can use the L-function. This variant of the K-function is defined as

$$L(r) := \sqrt[s]{\frac{K(r)}{b_s}} \qquad (2.27)$$

where s is the number of dimensions and b_s is the volume on the unit sphere in $\mathbb{R}^s$.

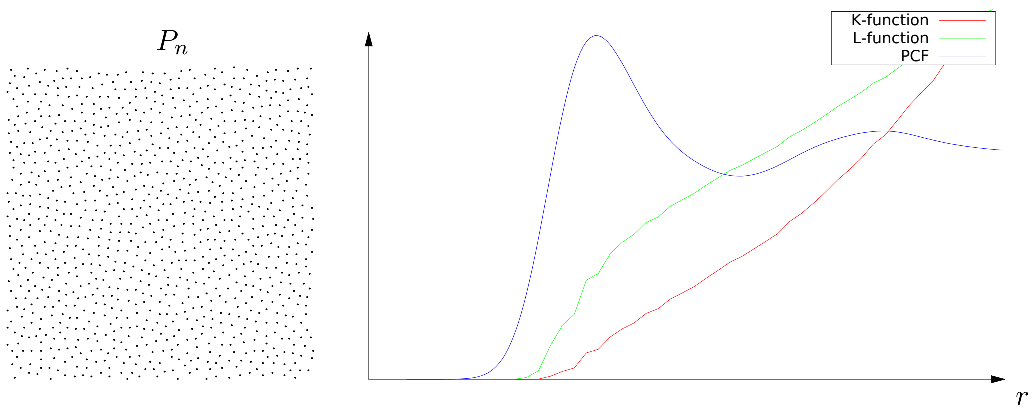


Fig. 2.11.: Ripley's K-function, the L-function and the PCF of the pointset P_n.

Both those functions can be seen as the cumulative distribution function (with some normalization) of the PCF. This PCF, noted $\varrho(r)$, is considered as the best second order characteristic. Intuitively, it expresses the number of samples that are at a distance r for a typical sample (Fig. 2.10 on the preceding page). For $r \geq 0$, it can be defined as

$$\varrho(r) := \frac{K(r)}{db_s r^{s-1}}. \tag{2.28}$$

Oztireli [OG12] devised a simplified estimator for this measure in the particular case of isotropic and stationary point processes. We therefore estimate the PCF of a pointset P_n in the unit domain $[0,1)^s$ using the following formula

$$\varrho(r) = \frac{1}{n^2 r^{s-1}} \sum_{i \neq j} k_\sigma(r - d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})), \tag{2.29}$$

where $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is a distance measure between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. The factor k_σ is used to smooth out the function. Oztireli relies on this smoothing to assume ergodicity for all sets. He uses the Gaussian function as a smoothing kernel, but one could use a box kernel or a triangle kernel instead. To compute a PCF, we use this estimator with 3 parameters, the minimal r, $r_{\min}$, the maximal r, $r_{\max}$ and the smoothing value σ. Those values are usually chosen empirically. Note that as the number of samples increase, the distances between samples will be very different for similar distributions. To alleviate this, we normalize the distances in our estimations using the maximal possible radius for n samples ([GM09], Eq (5)). The results of all those estimators are illustrated Figure 2.11.

The issue when computing the values for ϱ, K and L, is that we have as input a point set defined in a bounded domain. However, those estimators implicitly assume an infinite set. This creates a bias since for points close to the border of the domain, their nearest neighbours outside the domain will not be taken into account. To avoid this, one can periodically tile or mirror a point set. One can also use more advanced estimators that take those effects into account. However, with a sufficiently big set and with sufficiently small radiuses, those edge effects can often be ignored.

The PCF is also a very interesting estimator as has been proven to be the spatial equivalent of the Fourier transform. You can find the mathematical relationship between both those measures in [Kai+16]. We recall that the differential analysis (Section 2.2.3 on page 23) was also proven to be equivalent to the Fourier Transform. This means that in most cases, the spatial and spectral analysis are interchangeable, and the use of one or the other is partly dictated by which will make the computations easier. However, the spectral analysis has been thoroughly studied and has been proven to be directly correlated to the variance in Monte Carlo estimators. This studies have been partly done for the spatial analysis but it is still a work in progress. The relationships between those characterizations (spatial analysis, spectral analysis but also uniformity) and the error in Monte Carlo integration will be detailed in the next section.

2.4 Relationship to error in integration

All the previously detailed characterisation was not randomly chosen. Those particular measures are of interest to us as they all relate to the error in the resulting MC/QMC estimator. In this section we see how we can use this relationship to devise sampling strategies that reduce this error as much as possible. Note that similarly to the discrepancy, for unbiased samplers, the integration error reduces as the number of samples increases. Each sampling pattern therefore has a *convergence speed*, that characterizes how the variance of the MC estimator using this sampling pattern changes as the number n of samples increases. Our reference worst case is the convergence speed of a pure random sampler, which is $O(1/n)$ (Fig. 2.14 on page 32).

We remind here that we define the squared error in integration $\mathcal{I}_e$ as

$$\Delta = |\mathcal{I}_n - \mathcal{I}|, \tag{2.30}$$

where $\mathcal{I}$ is the real integration value and $\mathcal{I}_n$ is the approximation obtained from estimating it with a MC/QMC estimator, using n samples obtained from a given point process.

Thanks to the Koksma-Hlawka theorem [Hla61], the uniformity of a point set happens to bound the integration error of QMC estimators. We detail this theorem first, and then, we see how in the case of stochastic patterns, the error can be reduced thanks to variance analysis. We remind that in this section, we focus on unbiased samplers, and therefore the error in integration is strictly equivalent to the variance (see Section 1.1.2 on page 4).

2.4.1 Koksma-Hlawka

The first theorem that interests us is the Koksma-Hlawka theorem [Hla61]. This theorem was developed by Hlawka and proven by Koksma (for the 1-D case) [Kok42]. It bounds the integration error Δ using the discrepancy of a point set using the following inequality

$$\Delta \leq \mathcal{D}_*(P_n)V(f), \tag{2.31}$$

where $V(f)$ is assumed to be finite, and represents the variation of the function f in the sense of Hardy and Krause (definition below) [Lem09].

This boundary implies that for point sets that are low discrepancy, the integration error is supposedly lower than for other point sets. An extension of this theorem was developed for the generalized l_2 discrepancy [Lem09] as

$$\Delta \leq \mathcal{D}_2(P_n)V_2(f), \tag{2.32}$$

where $V_2(f)$ is finite and is the l_2 equivalent of $V(f)$.

As detailed in [Kel06], this theorem is however a little unsuited for computer graphics. The issue is that it only applies on the class of functions of bounded variation in the sense of Hardy and Krause. A 1-D function is said to be of bounded variation if its total variation is finite. More formally, we denote V_a^b the variation of a 1-D function f in an interval $[a, b]$ with

$$V_a^b(f) := \sup_{P \in \mathcal{P}} \sum_{i=0}^{n_P - 1} |f(x_{i+1}) - f(x_i)| \tag{2.33}$$

which takes the supremum over the set $\mathcal{P} = \{P = \{x_0, x_1, \dots x_{nP}\} |\ P$ is a partition of $[a, b]$ with $x_i \leq x_{i+1}$ for $0 \leq i \leq n_{P-1}\}$ of all partitions of $[a, b]$. Then, f is a Bounded Variation (BV) function if

$$V_a^b(f) < +\infty. \tag{2.34}$$

A BV function in the sense of Hardy and Krause is an extension of this definition to 2-D functions, which computes if the function is BV along the x and the y axis. In [Kel06], Keller argues that this class of functions is very unhandy as, for discontinuities that are not axis aligned (which are very common in graphics), V_a^b becomes infinite. Following this observation, [Kel06] derived several new inequalities to relate the discrepancy to the integration error. However, as they imply diving into deeper details of BV functions and lead to results similar to the Koksma-Hlawka theorem, we will not detail this here. Interested readers can refer to [Kel06].

We can already note that such theorem implies that a Low Discrepancy sampler, with a discrepancy of $O(\log(n)^{s-1}/n)$ when the number of samples n increases has a 2-D squared convergence speed of $O(\log(n)^2/n^2)$ (Fig. 2.14 on page 32).

Even though those boundaries are very useful for deterministic samplers, they are still of limited use, since they are just an upper bound instead of a direct relation. This direct relation can be found when using stochastic samplers and variance analysis.

2.4.2 Integration error analysis

For stochastic samplers, we don't rely much on the integration error, but instead we rely on the MSE (as detailed Section 1.1.2):

$$MSE = \text{Bias}^2 + \text{Variance}. \tag{2.35}$$

Since we are only interested in unbiased stochastic samplers, this formulation states that the observed integration error is solely related to the variance of the approximation.

In the case where the error is solely due to the variance, we can rely on two very strong results. The first one [Pil+15], directly relates the variance to the Fourier spectrum of both the sampling pattern and the function to sample. The second one [Özt16] relates the PCF of a sampling pattern to the integration error.

Spectral variance analysis

Many papers have conducted analysis of the variance in Monte Carlo integration, in both the Euclidean and spherical domains [Dur11; SK13; Hes+10; Ram+12; Pil+15]. We will detail here the results obtained by [Pil+15]. Their result is an expression of the variance observed when using stationary point patterns, as a direct function of their Fourier spectrum. We denote I the function to integrate, and $\mathcal{I}_n$ the MC estimation of the integral of I with a point set P_n. The variance of $\mathcal{I}_n$, is given by

$$var(\mathcal{I}_n) = \frac{\mu(\Omega)^2}{n} \int_\Omega \langle \mathcal{P}_s(\omega) \rangle \mathcal{P}_I(\omega) d\omega \tag{2.36}$$

where $\mu(\Omega)$ is the Lesbegue measure for the unit toroidal domain $[0,1)^s$ and $\mathcal{P}_s, \mathcal{P}_I$ are the Power Spectra of respectively the sampling pattern and the integrand I. This formula can be further simplified for isotropic point patterns. Expressed in polar coordinates $(\mathbf{n}, \rho)$, we have

$$var(\mathcal{I}_n) = \frac{\mu(\Omega)^2 \mu(S^{s-1})}{n} \int_0^\infty \rho^{s-1} \hat{\mathcal{P}}_s(\rho) \hat{\mathcal{P}}_I(\rho) d\rho \tag{2.37}$$

where S^{s-1} is the surface area of the s-D sphere, and $\hat{\mathcal{P}}_s, \hat{\mathcal{P}}_I$ are the radial average of the power spectra of respectively the point set and the integrand function. Note that as we are using the radial power spectrum, we assume an isotropic spectrum.

It has been shown that for natural images, $\hat{\mathcal{P}}_I$ will approximate a $\frac{1}{\omega}$ curve (see Chapter 7). This means that as ρ increases in Equation 2.37, the value of $\hat{\mathcal{P}}_I$ will tend to 0. This means that to reduce the variance, the value of $\hat{\mathcal{P}}_s$ must be as small as possible when ρ is small (Fig. 2.12 on the facing page).

Our goal is therefore, for integration purposes, to use preferably sampling patterns that present as little energy in the low frequencies as possible. In 1983, Yellot [Yel83] observed that the

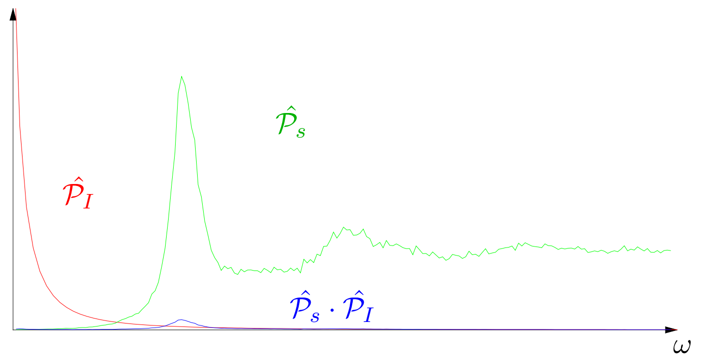
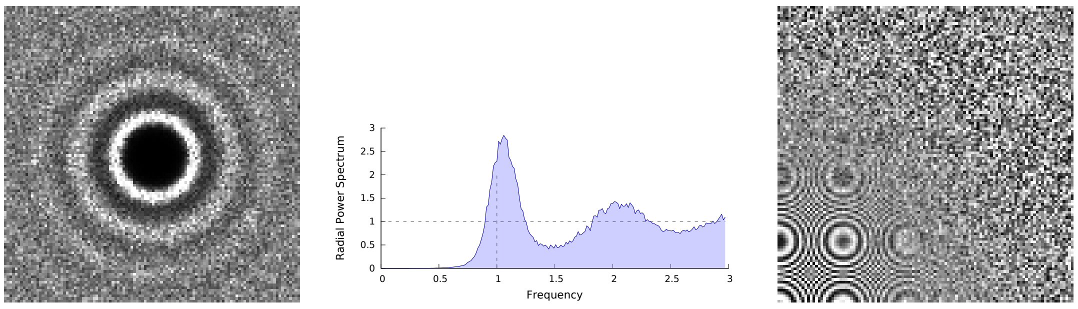


Fig. 2.12.: Product of the radial spectra of the sampling pattern $\hat{\mathcal{P}}_s$ and of the integrand $\hat{\mathcal{P}}_I$.



Fourier Spectrum

Radial Average

Reconstruction of the function $\sin(x^2 + y^2)$

Fig. 2.13.: Blue Noise fourier spectrum and radial average.

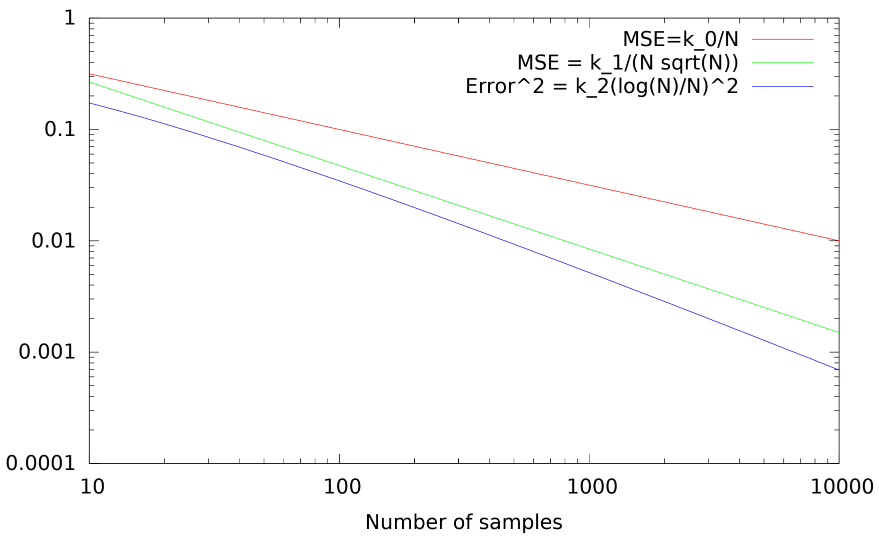
photoreceptors in the retinas of monkey where distributed following a very uniform but yet irregular distribution. It has been theorized by Ulichney [Uli88] that this distribution, called *Blue Noise*, leads to the the best spectrum for integration. This ideal spectrum presents no energy in the lowest frequencies, followed by a peak and a flat region (Fig. 2.13). Currently, samplers that generate samples with such a spectrum are referred to as Blue Noise samplers.

Following this analysis, the theoretical MSE of Monte Carlo estimators using a stratified sampler decreases in $O(1/(n\sqrt{n}))$ as n increases. A similar convergence speed was also empirically devised for the Blue Noise samplers. This is better than the speed of the pure random sampler but asymptotically not as good as the convergence speed of Low Discrepancy Samplers (Fig. 2.14 on the following page).

Spatial variance analysis

In [Özt16], the authors devised a formula relating the PCF of a stationary sampling pattern to its resulting variance in MC integration. We will denote ϱ the PCF of a sampling pattern. From Campbell's theorem, they expressed the expected value of the MC integration of a function f defined over a domain Ω using the statistics of the sampler used. If we denote $\mathcal{I}_n$ the MC estimation of the integration of f over Ω, this leads them, for stationary processes, to

$$E(\mathcal{I}_n) = \int_{\Omega} f(\mathbf{x})\lambda d\mathbf{x} \tag{2.38}$$



The convergence speeds of the main categories of samplers, when the number n of samples increases. In 2-D, the pure random sampler, which is our worst case reference, converges in $O(1/n)$. The samplers with as little energy as possible in their low frequencies converge in $O(1/(n\sqrt{n}))$, when the low discrepancy samplers converge in $O(\log(n)^2/n^2)$.

Fig. 2.14.: Comparison of convergence speeds.

where λ is the intensity function (constant for stationary processes). They also derive $E(\mathcal{I}_n)^2$ as

$$E(\mathcal{I}_n)^2 = \int_\Omega f^2(\mathbf{x})\lambda d\mathbf{x} + E(\sum_{i\neq j} f_i f_j) \tag{2.39}$$

$$= \int_\Omega f^2(\mathbf{x})\lambda d\mathbf{x} + \int_{\Omega\times\Omega} I(\mathbf{x})I(\mathbf{y})\lambda^2 \varrho(\mathbf{x}-\mathbf{y})d\mathbf{x}d\mathbf{y}. \tag{2.40}$$

Finally, from

$$var(\mathcal{I}_n) = E(\mathcal{I}_n{}^2) - (E(\mathcal{I}_n))^2 \tag{2.41}$$

they derive the formula relating the variance to the PCF,

$$var(\hat{\mathcal{I}_n}) = \frac{1}{\lambda}\int f^2(\mathbf{x})d\mathbf{x} + \int a_f(\mathbf{h})\varrho(\mathbf{h})d\mathbf{h} - \left(\int f(\mathbf{x})d\mathbf{x}\right)^2 \tag{2.42}$$

where $\mathbf{h} = \mathbf{x} - \mathbf{y}$, and a_f represents the auto-correlation of f, which tends to 0 as $\mathbf{h}$ increases due to the finite support of f.

If we look at each term of this function, we see that the first term depends on $\frac{1}{\lambda}$, meaning that the more samples the higher the intensity and the lower the error. The third term does not involve the point pattern, so we only care for the second term. What can be seen here is that when $\mathbf{h}$ is small (when a_f is not 0), the value $\varrho(\mathbf{h})$ should be as small as possible. This underlines the spectral analysis from [Pil+15] that stated that as the Power spectrum of the function tends to 0 as the frequency increases, the Power spectrum of the point set should be as small as possible in the low frequencies. It also underlines the correlation between the PCF of a set and its radial spectrum.

2.5 Conclusion

In this section, we described several measures used to characterize a point set. Among them are the measures describing the uniformity of a set. The main one is the l_∞ star discrepancy. Thanks to the Koksma-Hlawka theorem, we know that a point set with a lower discrepancy limits the maximal possible error. As this value can be difficult to compute in s-D, one can rely on the generalized l_2 star discrepancy to ease the computations while preserving the Koksma-Hlawka inequality.

Then, we have all the measures that rely on computing statistics over the spatial distribution of samples. Those measures have been shown to be roughly equivalent to the spectral properties of the point set, while being in any cases much easier to compute, even in s-D. Furthermore, they are also directly related to the integration error, which makes them our main criteria when evaluating sampling patterns.

In the following chapter, we use all those characteristics to classify existing sampling techniques and sampling patterns, and to derive their convergence speeds.

3 State of the art

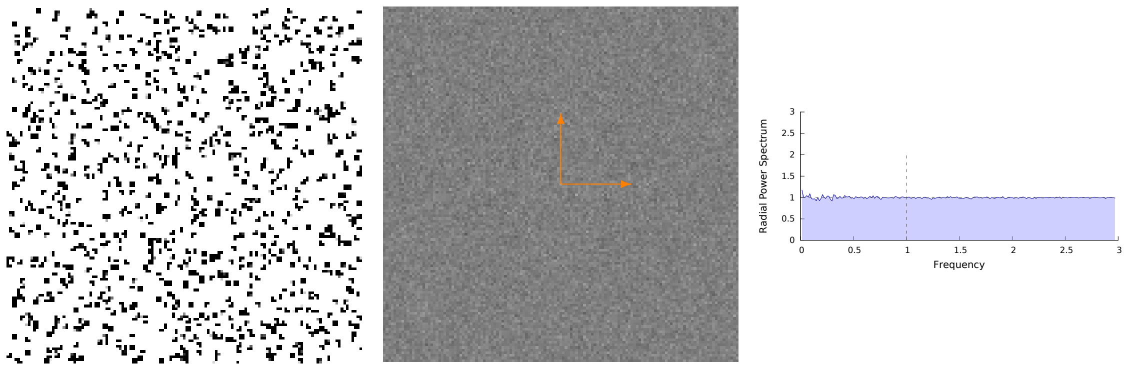
In this chapter, we present existing samplers, evaluating them based on the characteristics detailed in the previous section. We classify those samplers into 3 categories.

First, in Section 3.1 on the next page, we present some historical samplers. Such samplers usually aim at improving over the pure random sampler w.r.t. its uniformity. They are usually very simple and do not provide strong guarantees over the spatial organization of samples. However, such samplers are very efficient, and extend easily to higher dimensions. This means that even though, they are not optimal for Monte Carlo Rendering (where a higher correlation between samples is required to ensure a uniform and unstructured coverage of the integration domain), they are still implemented in most current rendering systems [Pha+16].

Section 3.2 on page 40 presents the samplers that generate Poisson Disk or Blue Noise spectrum. Both those types of samplers increase the amount of correlation between samples to try generating stochastic sets of samples with roughly the same distance between each sample and all of their neighbours. Note that the Poisson disk Spectrum differs slightly from the Blue Noise spectrum, as it presents a non-zero energy in the lowest frequencies, which is harmful to the variance of MC estimators (Section 2.4 on page 28). Still, in a vast part of the literature both those spectra are considered the same which is why we regrouped them. Contrary to historical samplers, Blue Noise and Poisson Disk samplers are limited to low dimensions (usually, they only generate 2-D samples).

Finally, Section 3.3 on page 57 details the samplers that are low discrepancy. Such samplers are usually defined from implicit analytical functions, based on bit reversal. This makes them very efficient, even in high dimensions, and allows to add consistent new samples iteratively within an existing point set. However, they are rarely stochastic and their analytical definition generates highly structured sampling patterns. Note that in most of this section, the samplers will be defined over the unit domain $[0, 1)^s$ (unless specified otherwise).

Figure 3.37 on page 71 presents a summary of the main samplers from each category, evaluated on the closeness of its spectrum to Blue Noise and on the quality of its discrepancy. Figure 3.38 on page 72 presents the same samplers, evaluated on their computing efficiency (speed) and on the number of dimensions that can be sampled.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.1.: Characteristics of a Whitenoise sampler.

3.1 Historical samplers

In this section, we present some historical samplers. Most of those samplers aim at improving over the pure random sampler. Thus, they usually present a higher error in integration than optimal Blue Noise samplers, but they do not generate aliasing during rendering since they do not present any structures in the sampling patterns. They are also very easy to implement and efficient even in higher dimensions. This is why they keep being so popular in rendering engines [Pha+16].

Whitenoise sampler We start by presenting the most naive sampler, the *Whitenoise* sampler. This sampler generates purely random samples. Consequently, in the resulting point set, all frequencies are present with the same energy which leads to a flat Fourier Spectrum (Fig. 3.1). Note that since the Fourier spectrum is flat, there is a higher energy in low frequencies, which leads to a high variance in integration. Due to the simple expression of this sampler, the MSE of the pure random sampler is known to be in $O\left(\frac{1}{n}\right)$, with a discrepancy in $O\left(\frac{1}{\sqrt{n}}\right)$.

Stratified sampler One of the drawbacks of using a Whitenoise sampler is that, as it is fully random, nothing prevents having subsets of samples that are arbitrarily close to each other. This translates as portions of space that contains a lot of samples, and others that are almost empty (see Figure 3.2 on the facing page, left). As the energy in the low frequencies of the spectrum is due to those empty spaces, we want to avoid them as much as possible. One way to do that, is to use *Stratified* sampling. This sampler, in its simplest form, partitions the space using a s-D grid, and generates a random sample inside each cell of this grid (see Figure 3.2 on the next page, right). This limits the number of arbitrarily close samples to subsets of size 2^s and thus increases the uniformity. As can be seen in the radial spectrum (Fig. 3.3 on the facing page), there are much less energy in the low frequencies which leads to a lower variance in MC integration, as detailed in [Lem09] (Section 1.1.2 on page 4).

This sampler is one of the most used in rendering engines as its variance is reasonably low and its implementation is very easy. However, this sampler cannot generate any number of samples. In dimension s, this sampler can only generate a point set P_n when $n = k^s$. Therefore, as we want

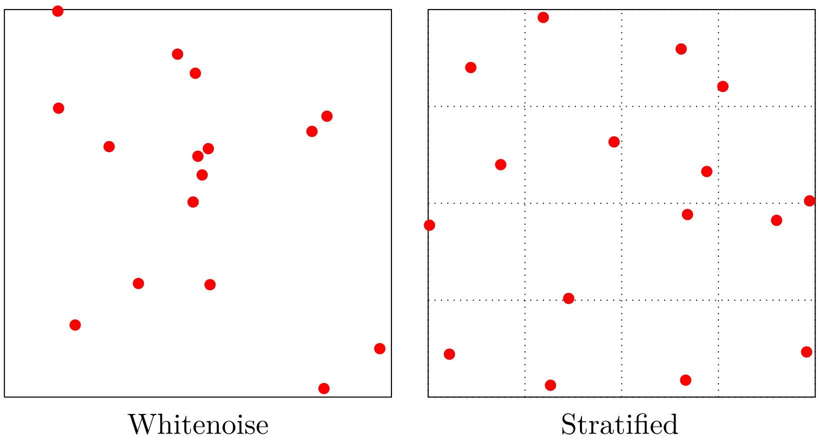


Fig. 3.2.: Comparison between a Whitenoise point set and a Stratified point set.

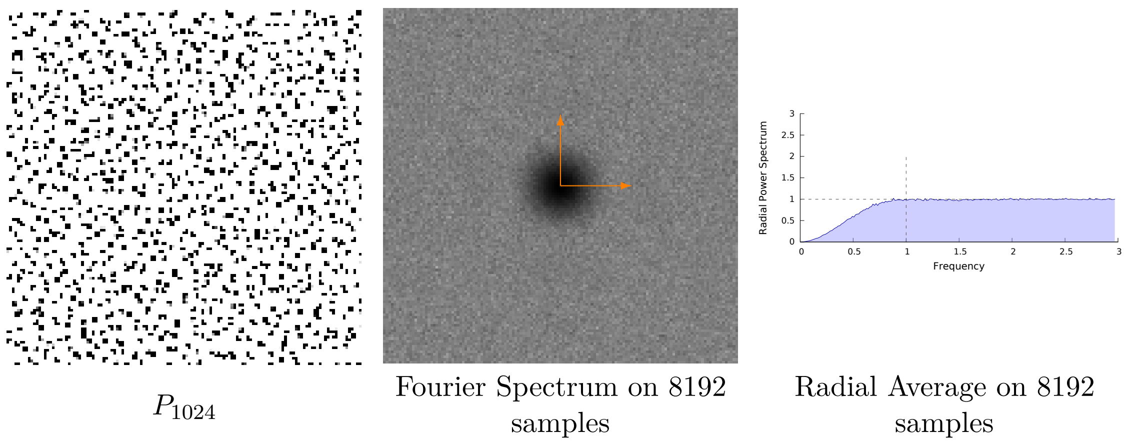


Fig. 3.3.: Characteristics of a Stratified sampler.

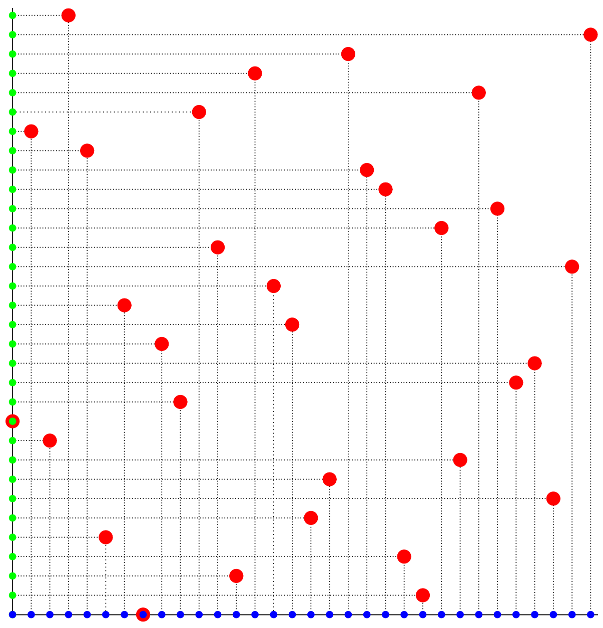


Fig. 3.4.: 2-D point set with stratified 1-D projections.

to integrate over more and more dimensions, we will have to generate more and more samples just to be able to properly sample each dimension. This phenomenon is called *the curse of dimensionality*, and is the main drawback of the Stratified sampler relatively to the Whitenoise sampler. This is usually alleviated in rendering by generating 2-D stratified sampling patterns that are then shuffled between dimensions [Pha+16].

This sampler has an improved uniformity but, several samples can still present the same x or y coordinate, leading to overlapping samples in the 1-D projections. As detailed in Section 2.1 on page 14, this overlap can increase drastically the discrepancy of a point set and thus should be avoided. To alleviate this, sampling strategies based on shuffling a canonical point set have been developed.

Shuffling canonical projections Most those methods aim at generating stochastic point sets while preserving the $(s-1)$-D projections [Shi+91; Chi+94; Ken13]. As explained in Section 2.1 on page 14, having two samples on the same line or on the same column is very harmful to the uniformity of this pattern. In the context of the samplers described here, preserving the $(s-1)$-D projections of a point set is used to ensure that those projections will be stratified. This supposedly decreases the error in the Monte Carlo estimators and therefore improves the quality of the rendered image. This property is illustrated Figure 3.4.

Those samplers usually start with a canonical set, either stratified or diagonalized (Fig. 3.5 on the next page). Then, they randomly swap the lines and columns in this arrangement. This preserves the 1-D projections, as no samples were on the same line or column originally and the coordinates are not changed, merely swapped.

The N-Rooks sampler is one of the main examples of such samplers [Shi+91]. But even though it preserves the stratification of the 1-D projections, it presents spectral characteristics that

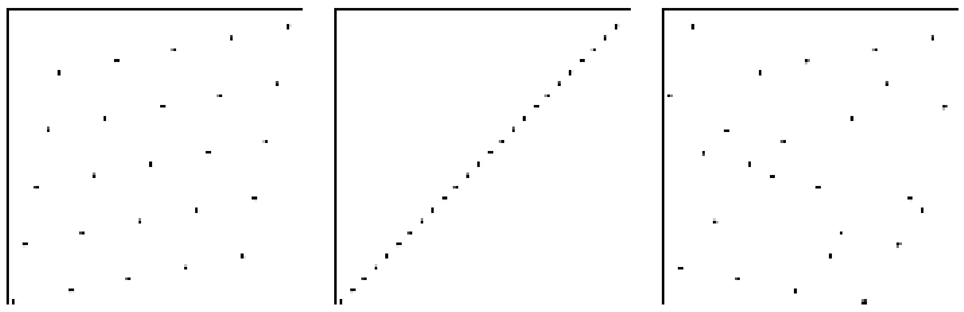


Fig. 3.5.: Shuffling canonical arrangements

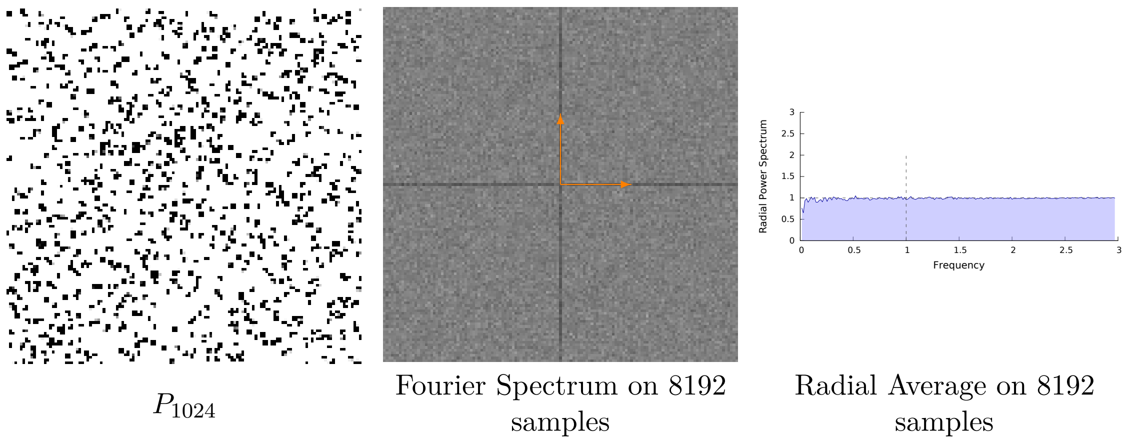
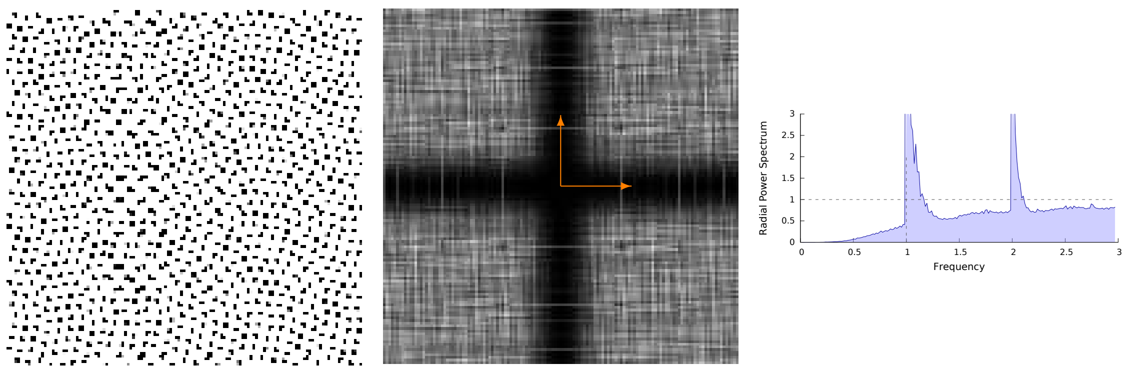


Fig. 3.6.: Characteristics of a N-Rooks sampler [Shi+91]

are almost identical to Whitenoise (Fig. 3.6). The only notable change is the presence of a "black cross" in the middle of the Fourier Spectrum (since there are no samples aligned on either dimension, there are no frequencies in the strictly vertical or strictly horizontal directions). The original goal of this sampling was to allow a stratified sampling for any number of samples, instead of simply for point set whose size is a power of s, but due to the poorer quality of its spectrum, this sampler is almost unused in rendering nowadays.

One can note that some authors [Ken13] alleviate this "Whitenoise effect" by correlating the swapping of coordinates. Their goal was to combine the very good properties of the stratified sampling with a higher uniformity, ensured by their original canonical arrangement. However, even if this gives good results for low number of samples, the correlation generates spectral peaks for a higher number of samples (Fig. 3.7 on the next page).

If we look at all the samplers presented in this section, we can note that are all very efficient and easy to implement, even in higher dimensions. They are therefore quite present in current rendering systems. However, as they rely or pure randomness (either in the sample generation or in the shuffling), they do not present Fourier spectra that are as good as the spectra from Blue Noise sampling patterns (Section 3.2 on the following page), or the excellent uniformity of Low discrepancy samplers (Section 3.3 on page 57).



P_{1024} | Fourier Spectrum on 8192 samples | Radial Average on 8192 samples

Fig. 3.7.: Characteristics of a Correlated Multi Jitter sampler [Ken13].

3.2 Poisson Disk and Blue Noise samplers

Using the analogy with the visible light spectrum, several colors of noise were devised (Fig. 3.8 on the facing page). The most famous one is the white noise, generated from pure randomness with a flat spectrum. One can also note the existence of red or pink noise, that present more energy in the lowest frequencies. Such noise are formed by clustered points and are unsuited for Monte Carlo integration, but are very useful in biology or physics to represent the distributions of for example plants or galaxies. Green noise presents higher mid frequencies, and can be used to modelize fallen leaves or the distributions of seeds around trees for example. Finally, Blue Noise puts the emphasis onto high frequencies. This type of noise is of major interest for Monte Carlo integration and is the kind of distribution that interest us here. A good survey of Blue Noise sampling techniques can be found in [Yan+15]. Many notations in this section will come from this survey.

Samplers that generate a Blue Noise spectrum can be roughly divided into four families. First, those that generate Poisson Disk distributions (Section 3.2.1), based on variants of the Dart Throwing algorithm. Contrary to a true Blue Noise distribution, the Fourier spectrum obtained from Poisson Disk is not zero for the lowest frequency [Pil+15]. However, those samplers are still considered Blue Noise in an important part of the literature. Then, we will present samplers that generate true Blue Noise distributions from gradient descent and energy minimization (Section 3.2.2 on page 44), and samplers that generate Blue Noise distributions from tiled based methods (Section 3.2.3 on page 49). Finally, we will see algorithms that generate Blue Noise from Dithering (Section 3.2.4 on page 52), and other less common methods (Section 3.2.5 on page 54).

3.2.1 Poisson Disk samplers

The most naive sampler generating Poisson Disk distribution is the *Dart Throwing* sampler [Coo86] (Fig. 3.9 on the facing page). The idea is to iteratively insert samples into an already existing set. A new sample is generated purely at random, and is kept only if it is at a sufficient

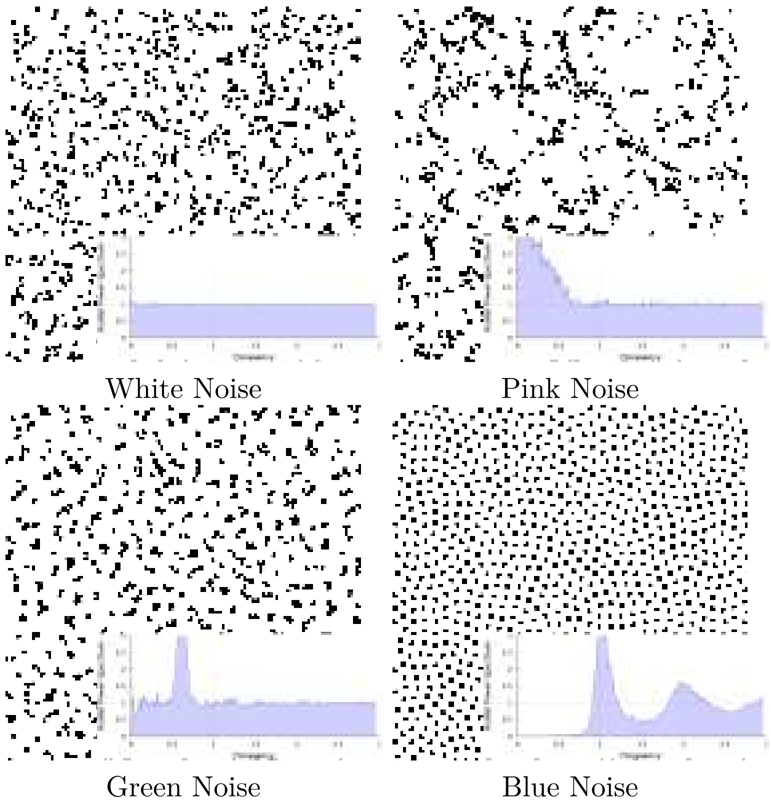


Fig. 3.8.: Pointset and Radial Spectrum from several colors of noise.

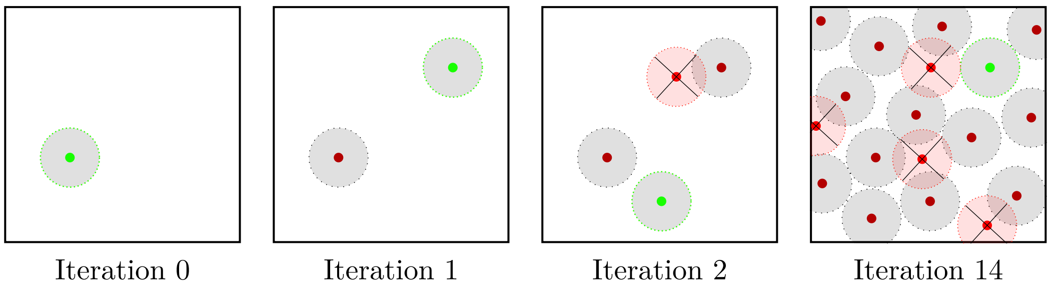


Fig. 3.9.: Dart Throwing algorithm.

distance from every other samples. This distance is called the disk radius, and is computed based on the maximal packing density γ_s of n s-D spheres with the following formula

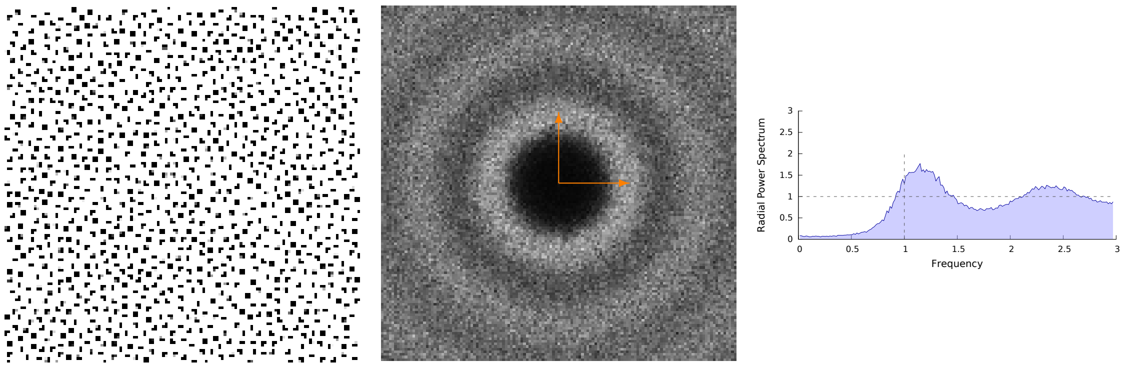
$$r_{\max} := \sqrt[s]{\frac{\gamma_s}{n}\frac{\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}, \tag{3.1}$$

where the Γ function is the extension of the factorial function to positive complex and real-valued arguments, defined as

$$\Gamma(x) := \int_0^\infty x^{z-1}e^{-x}dx \tag{3.2}$$

with $x \in \mathbb{C}$.

However, this algorithm has a major flaw: it is not guaranteed to end. If one wants to generate n samples, it is possible to reach a situation where $m < n$ samples where placed, but where there is no longer any position in space that is at a sufficient distance from all other samples.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

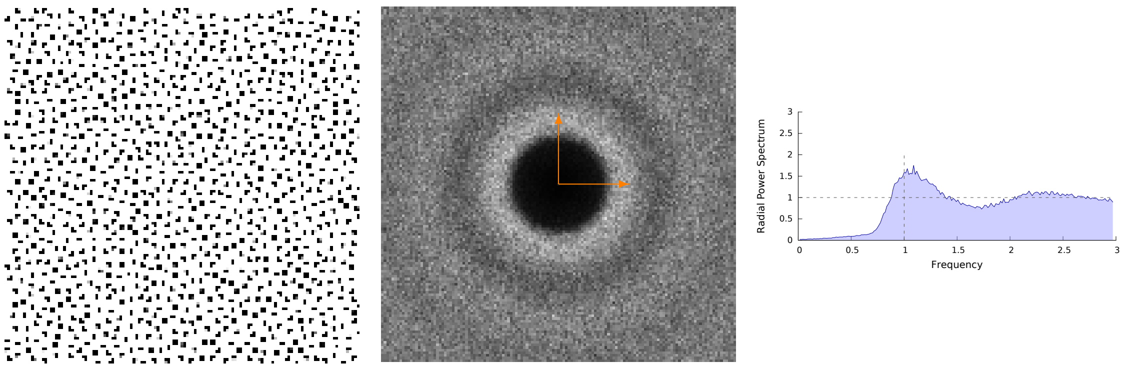
Fig. 3.10.: Characteristics of a Relaxation Dart Throwing sampler [MF92].

To alleviate that, the *relaxation dart throwing* method [MF92] was developed (Fig. 3.10). This algorithm initiates the radius $r_{\max}$ with a large value, and if there is too many unsuccessful trials, this radius is decreased. This process goes on until all n samples have been placed. Note that this relaxed dart throwing can be considered as a sequential sampler, as it iteratively adds samples within a pre-existing set, while being consistent regarding the distance between pairs of samples.

Note that those two methods, generate different types of Poisson disk sampling. The dart throwing sampler belongs to the family of *accurate samplers*, which generates point sets such that all the distances d between sample pairs are $d > r_{max}$. In opposition, the relaxed dart throwing algorithm is an *approximate sampler*. It generates point sets where the distance between the majority of sample pairs is $d > r_{max}$ but where there might be pairs of samples with $d < r_{max}$. Another side effect of the relaxed Dart Throwing is that the resulting sampling pattern is not necessarily *maximal*. This means that samples could still be added to the set without changing the value of $r_{\max}$.

For both those algorithms, one can note that they are highly inefficient. You may generate lots of samples before finding one that fits. Therefore, another simple approximate algorithm for Poisson disk sampling was devised, the *Best candidate algorithm* [Mit91]. This algorithm generates a new random samples by generating k new candidates, and then by selecting the sample with the largest minimum distance to all others. Note that this algorithm is guaranteed to finish and is much faster than dart throwing. However, it is sill has an $O(n^2)$ complexity.

Another noticeable approach, although quite exotic, is to generate Poisson Disk through molecular dynamics [LS90]. Such methods generates n random particles and moves them around with a collision check. The radius of the particles gradually increases until no particle can move anymore. The main asset of such methods is that particular dynamics offers well known and efficient algorithms, that can be parallelized on GPU. This makes such approach faster than the previous ones, but it may generate very regular structures in 2-D. Other similar methods will be devised for Blue Noise at the end of Section 3.2.2 on page 44.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.11.: Characteristics of Real Time Accurate Poisson Disk sampler [DH06].

Other algorithms were devised to generate Poisson disk sampling in $O(n)$ or $O(n\log(n))$, usually relying on efficient data structures. In [DH06], authors present a method that generates 2-D Poisson Disk sampling in $O(n\log(n))$ (Fig. 3.11). They accelerate the algorithm by only sampling subsets of the domain, which are known to contain at least one sample in a maximal distribution and by using a carefully chosen data structure. This new data structure is called *scalloped*, as the subsets to be sampled are defined as a difference between disk regions. Authors of [Jon06] also present an accelerated algorithm using the intersection of the exclusion disk and Voronoi diagrams to identify the subsets of space where new samples are allowed. As both those new structures are difficult to handle, other methods [Whi+07] use a simple quadtree structure to guide the sample placement. Some methods simply used the parallel properties of the GPU to accelerate the samples generation [Wei08; Bow+10; Xia+11; Ebe+11]. Note that [Ebe+11] also allows for sample generation on non convex domains. We also point that [Wei08] can generate higher dimension poisson disk sampling, although its practicability decreases as the dimension increases.

In his paper, Bridson [Bri07] accelerates the generation of samples in high dimension by partitioning the domain using a regular grid. This grid resolution is carefully chosen so that each cell contains at most one sample. Then, when generating a new sample, we use the best candidate algorithm but only test the distance against the samples in the neighbouring cells instead of testing the distances against all samples. This algorithm runs in $O(n)$ but generates approximate distributions. There exists similar algorithms that runs in $O(n\log(n))$ for accurate distributions [GM09]. However, since all those methods rely on a grid subdivision of the domain, they all face the curse of dimensionality. This means that as the number of dimension increases, the number of grid cells increases exponentially. To alleviate that, [Ebe+12] uses a flat implicit representation of a quadtree that is also ported on the GPU.

Note that until now, most of the aforementioned Blue Noise methods aimed at generating 2-D (or low-D) Poisson Disk sampling patterns. This limitation comes from the sampling structures used that were defined in 2-D, or simply to the computational cost of the algorithm that prevents its use in higher dimensions.

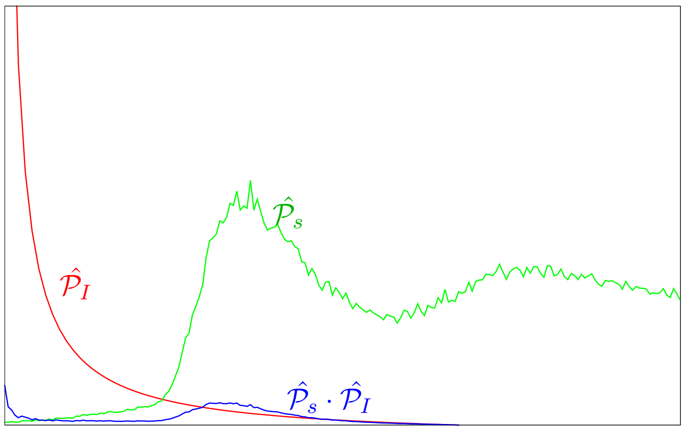


Fig. 3.12.: Product of the radial spectrum of a Poisson sampling pattern $\hat{\mathcal{P}}_s$ and of an integrand $\hat{\mathcal{P}}_I$.

For all those methods, we still have a non-zero energy in the low frequencies (Figures 3.10 on page 42, 3.11 on the previous page), in [Tor+06], authors found that, for a point set P_n with sufficiently large n, the minimal value of the Power Spectrum near the DC was $1/20$. It will therefore have a very bad behaviour regarding variance reduction, due to the relationship between the spectrum and the variance [Pil+15] (see Section 2.4.2 on page 30). Figure 3.12 illustrates the product of the radial spectrum of a Poisson Disk distribution [DH06] with the radial spectrum of a typical scene. One can note that since the radial spectrum of the point set is not null in the lower frequencies, the product has a higher value near 0. Since the variance is directly linked to the integrand of this product [Pil+15], this small peak at 0 automatically leads to a higher variance. Poisson Disk are therefore mainly used for stippling or object placement but not for Monte Carlo integration, which is our topic of interest. Instead, we prefer to use Blue Noise samplers.

There are several families of methods to generate Blue Noise samplers. The main ones are the algorithms based on energy minimization (Section 3.2.2), and the tiled based systems (Section 3.2.3 on page 49). But Blue Noise can also be generated from dithering matrices (Section 3.2.4 on page 52).

3.2.2 Optimization based Blue Noise

The samplers discussed in this section present the ideal spectrum for variance reduction as they present no energy in the lowest frequencies (Fig. 3.8 on page 41). We refer to this energy-less interval as the *zero region* of a spectrum. In this section, we present methods that generate Blue Noise from energy minimization. Note that most Blue Noise samplers were initially devised to account for adaptive sampling. We remind the reader that in this document, we only focus on uniform sampling, but we will still compare the performances of Blue Noise samplers regarding their adaptivity. As it was one of their original purpose, this whole section would make little sense if we did not. Nonetheless, our main focus will stay on their variance reduction property.

The simplest algorithm to generate Blue Noise from iterative optimization is known as Lloyd's relaxation [Llo82; Du+99], also referred to as Centroid Voronoi Tessellation (CVT) in Computer

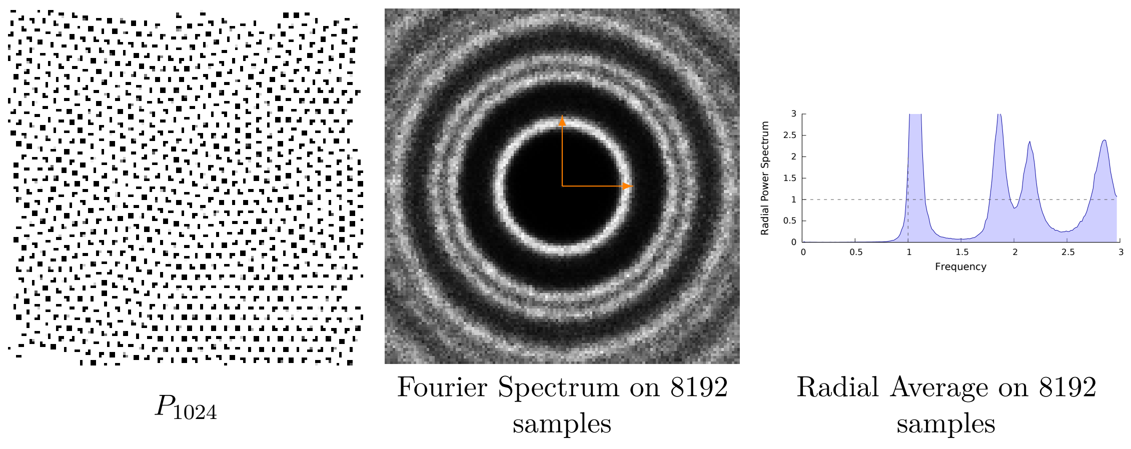


Fig. 3.13.: Characteristic of a Lloyd's relaxation [Du+99].

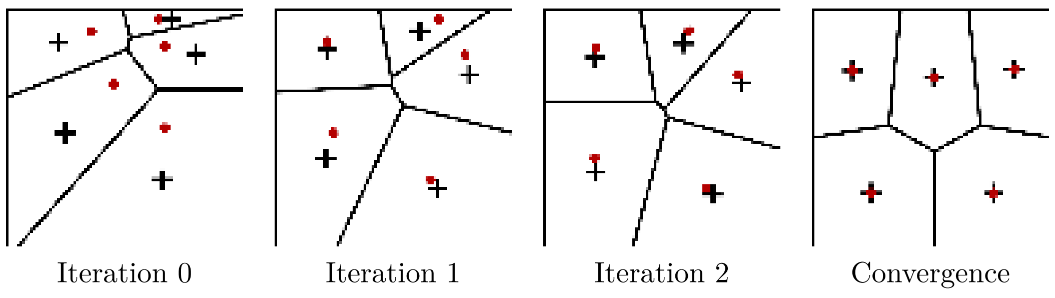


Fig. 3.14.: Lloyd's algorithm.

Graphics (Fig. 3.13). This algorithm generates the Voronoi diagram of a set of samples and displace each sample to the center of its Voronoi cell. The process is then repeated, until the system reaches stability (Fig. 3.14). Using such methods, a point set P_n is optimized by minimizing the following energy

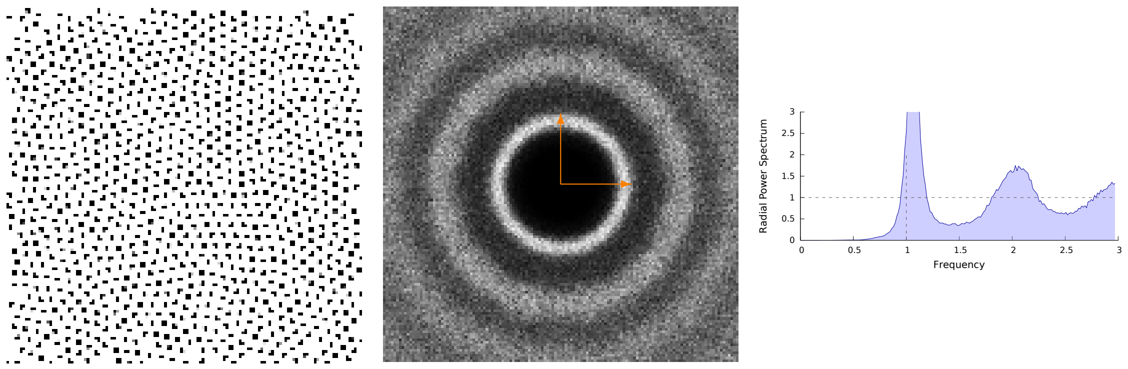
$$E_{CVT} := \sum_{i=1}^{n} \int_{V_i} ||\mathbf{x} - \mathbf{x}_i||^2 d\mathbf{x} \tag{3.3}$$

where $\{V_i\}_{i=1}^{n}$ is the set of cells of the Voronoi diagram with

$$V_i := \{\mathbf{x} \in \Omega, \forall j ||\mathbf{x} - \mathbf{x}_i|| \leq ||\mathbf{x} - \mathbf{x}_j||\}. \tag{3.4}$$

A key to the efficiency of such algorithms is the computation and update of the Voronoi diagram. In this document, we will assume that this computation is done and we will not go into further details regarding how it is done. An interested reader can refer to [All+05; Val+08; Yan+09; Ron+11; Yan+14b] for more information regarding Voronoi diagrams. In 2-D, point sets computed using Lloyd's algorithm converge to an hexagonal grid. To alleviate this, the optimization must be stopped after a certain number of iteration, and before reaching convergence. Unfortunately, there is no objective rule as to how to choose this number of iterations [Bal+09]. Furthermore, the computation of the Voronoi diagram limits this algorithm to low dimensions.

Early algorithms aimed at generating Blue Noise samples for stippling applications. Therefore, their goal was not the Fourier spectrum of the resulting set but its visual quality and adaptivity.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.15.: Characteristics of the Capacity Constraint Voronoi sampler [Bal+09]

In [Sec02], the authors proposed an adaptation of the Lloyd's relaxation, weighting the Voronoi diagram by the density function they were trying to stipple. Still for stippling purposes, Balzer et al. [Bal+09] (Fig. 3.15) replaced the Voronoi diagram by a *Capacity Constraint Voronoi Tesselation* (CCVT) (Fig. 3.15). They associate to each sample of a pointset P_n, a capacity c_i computed from the area of its Voronoi cell and of the underlying density function ρ,

$$c_i := \int_{V_i} \rho(\mathbf{x}) d\mathbf{x}. \tag{3.5}$$

Then, at each iteration, they update this capacity constraint diagram and move the samples to the centroids of their cells, such that $\forall i \in [1, n], c_i = c*$ with

$$c* := \frac{\int_\Omega \rho(\mathbf{x}) d\mathbf{x}}{n}. \tag{3.6}$$

Their method was then improved upon by [Li+10b] that proposed an accelerated algorithm to construct capacity constrained Voronoi tessellations, allowing this computation to be parallelized on GPU. Another paper by Xu et al. [Xu+11] aimed at speeding up this Blue Noise generation. In this paper, the authors no longer use a CCVT but instead the Capacity Constraint Delaunay Triangulation (CCDT), as the Delaunay triangulation is the dual structure from the Voronoi diagram. This method is faster than [Bal+09] and extends the definitions to mesh surfaces. Finally, [Che+12] combined CCVT and CVT for sample surfaces. Their energy function is

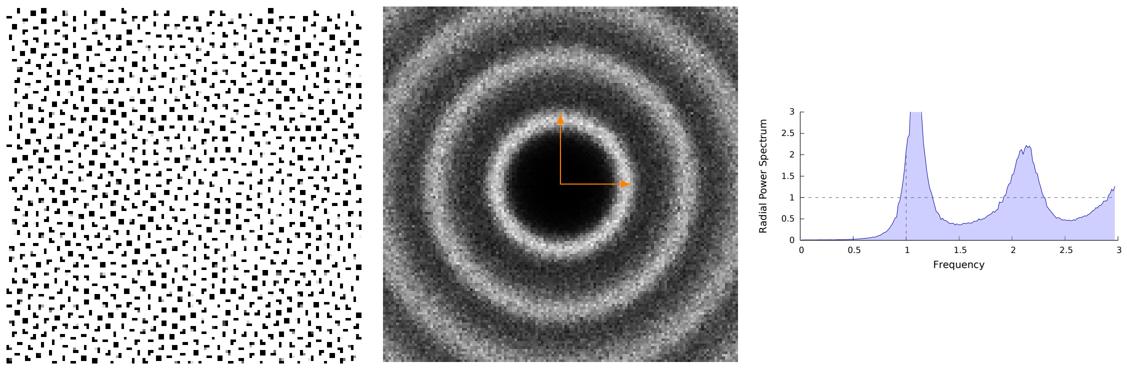
$$E_{CapCVT} := E_{CVT} + \lambda E_{CapVT} \tag{3.7}$$

where E_{CapVT} is defined as

$$E_{CapVT} := \sum_{i=1} n \left(\int_{V_i} \rho(\mathbf{x}) d\mathbf{x} \right). \tag{3.8}$$

They also achieve a significant performance improvement from an efficient optimization framework.

Other methods were devised based on [Eld+97]. This method places samples one by one, placing each new sample in the region with the less samples already present. They define the



P_{1024} — Fourier Spectrum on 8192 samples — Radial Average on 8192 samples

Fig. 3.16.: Characteristics of the FPO sampler [Sch+11]

position of this sample as the point of space that is further from any other samples. Proof exists that this sample necessarily lies on the Voronoi diagram of the current set. Therefore, at each iteration, the algorithm tests all points on the edges of the Voronoi diagram of the set, adds a point on the one that is farthest from all samples, and updates the Voronoi with this new sample. This idea was extended in [Sch+11], with their Farthest Point Optimization (FPO) method. In this paper, the whole set of samples is generated at random and iteratively optimized. Each iteration selects a sample, and pushes it as far away as possible from all other samples. However, the resulting pattern still presents some regularity that we would like to avoid (Fig. 3.16). In [Kan+11] the authors presented a similar method using a Delaunay triangulation updated iteratively. Using this Delaunay triangulation, the FPO algorithm was parallelized on the GPU by [CG12]. Finally, it was extended to surfaces and adaptive sampling by [Yan+14a].

A very notable method for Blue Noise generation was developed using Optimal Transport (BNOT) [DG+12] (Fig. 3.17 on the following page). They use a Power diagram instead of a Voronoi diagram. A Power diagram is an extension of the Voronoi diagram where each cell is defined as

$$V_i^w := \{\mathbf{x} \in \Omega, ||\mathbf{x} - \mathbf{x}_i||^2 - w_i \leq ||\mathbf{x} - \mathbf{x}_j||^2 - w_j \forall j\} \tag{3.9}$$

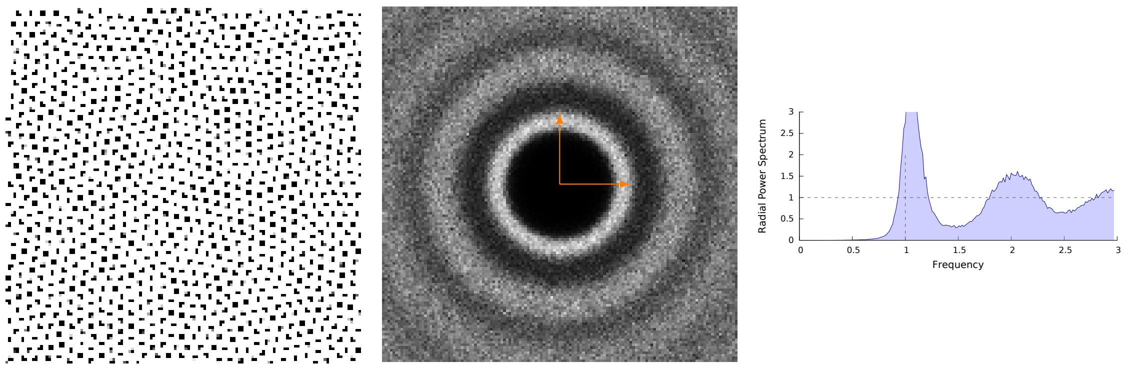
with $W = \{w_i\}_{i=1}^n$ the set of weights defined for each sample. They then minimize the following function

$$E_{BNOT} = \epsilon(W) + \sum_{i=1} n\lambda_i(c_i - c) \tag{3.10}$$

where $c_i = \int_{V_i^w} \rho(\mathbf{x}) d\mathbf{x})$ and $\epsilon(W)$ is defined as

$$\epsilon(W) = \sum_{i=1} n \int_{V_i^w} \rho(\mathbf{x}) ||\mathbf{x} - \mathbf{x}_i||^2. \tag{3.11}$$

There also exist methods that generate Blue Noise with particle based approaches [Fat11; Jia+15]. In [Fat11] (Fig. 3.19 on page 49), the authors motivation is to have a better computational efficiency than optimization based methods. They attach a density function to each sample and measure the difference between the sum of densities from the samples and the ex-



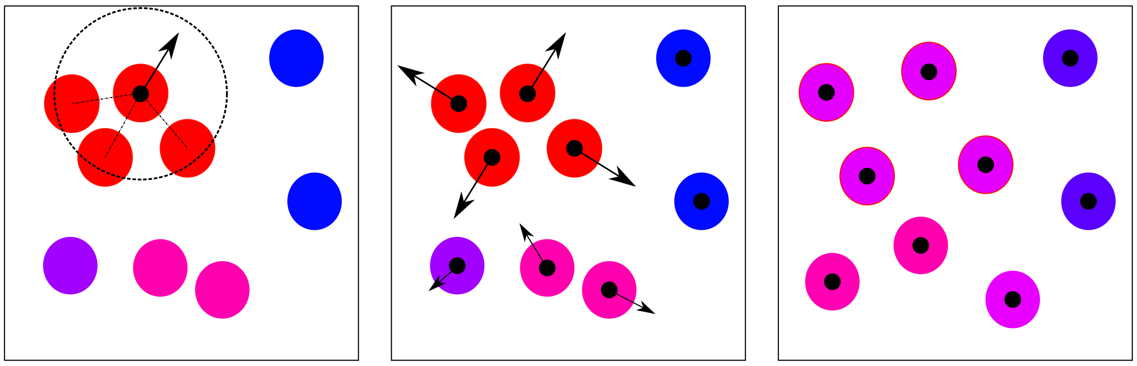
P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.17.: Characteristics of the BNOT sampler [DG+12].

pected density. This gives an energy function, but instead of minimizing this energy, they use it to define a statistical model, from which they draw points. This kind of algorithm is slow, but by generating samples one by one, the authors manage to generate their samples in linear time. It is worth noting that this algorithm was initially thought for stippling, and cannot be used for Monte Carlo integration. The issue is that, as the authors draw samples from statistical models, there is a minor shifting of the samples, imperceivable for stippling, but that generates a lot a bias when used with Monte Carlo estimators.

Other particle based approach are the ones of [Jia+15]. Their general algorithm is illustrated Figure 3.18 on the facing page. The authors rely on Smoothed Particle Hydrodynamics (SPH), a method from fluid simulation that minimizes the internal pressure variance. This method uses a spring mass system if the difference between the density of particle and the rest density is positive, the particles are pushed away, and if the difference is negative, the point are brought together. The adaptation of [Jia+15] is to have clamped this difference to 0, so that the points are never brought together. Each sample is associated to a density kernel. At each iteration a sample is added and the spring mass system is updated. Their method allows to generate higher dimensions Blue Noise at faster rates than other optimization based methods. More recently, [Ahm+16a] proposed a new particle based Blue Noise method, aiming at optimizing the coverage radius while preserving computational efficiency. However, to the best of our knowledge, such methods usually focus on surface sampling for remeshing and the resulting sets where never thoroughly tested for Monte Carlo integration.

All aforementioned methods tend to generate Blue Noise in "low-D". However, in a rendering context, we need to solve several recursive 2-D integrands (see Section 1.1.1 on page 2). In this context, the method in [Rei+15] is notable as its aim is to generate s-D sampling patterns, with the added constraint that they present the same spectrum in every projections. Authors suggest two methods. The first one, is derived from the dart throwing algorithm, which generates Poisson Disk sampling patterns in every projections. To do so, they add new constraints on the Dart Throwing algorithm, one for each projection. As in the original Dart Throwing algorithm, a new sample is added if it is a proper distance from all other samples in the domain, but it is also tested over all samples within each projections of the s-D set. Only a sample passing all those tests is added to the final set. The second method relies roughly on the same idea of

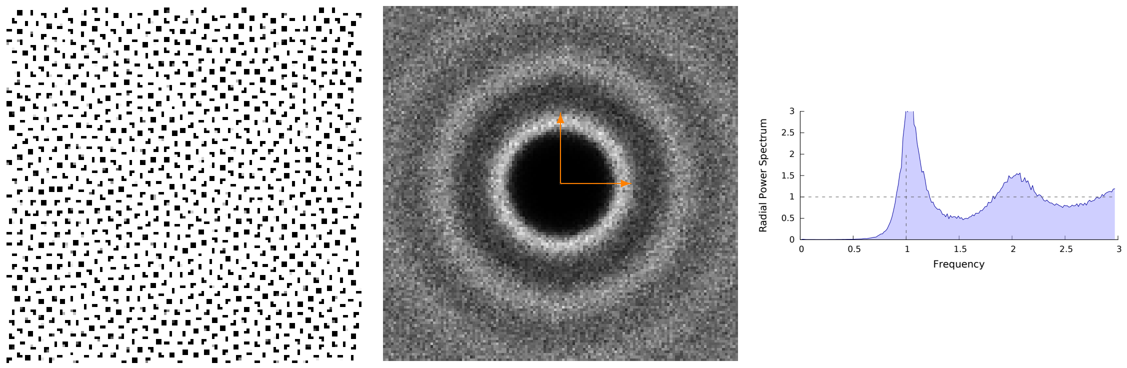


Identify the neighbours of a sample and compute forces through a spring mass system

Repeat for each sample

Apply forces to move the samples

Fig. 3.18.: Single iteration of a naive basic particle based Blue Noise sampling.



P_{1024}

Fourier Spectrum on 8192 samples

Radial Average on 8192 samples

Fig. 3.19.: Characteristics of the particle based sampler from [Fat11].

adding projection-wise constraints, but is derived from the Lloyd's relaxation algorithm, and generates Blue Noise sampling patterns in the projections. Unfortunately, despite the s-D claim of the paper, both those algorithms are so costly that they hardly generate samples in more than 4 or 5 dimensions. They are also almost unable to generate a high number of samples. Most their examples are limited to a few thousands samples.

All the methods in this section rely on iterative optimisations, usually through the computation and update of Voronoi Diagrams or through particle simulation. Those optimizations generate high quality Blue Noise but it has a computational cost. To alleviate this cost, Tiled Based methods were devised, based on precomputed tiling patterns, that simply needs to be combined together at run-time.

3.2.3 Tiled Based algorithms

In this section, we focus on methods generating Blue Noise from precomputed tiles filled with samples. Such methods are very efficient as they are based on simple recursive paving rules and tiles combinations. The main difficulty is avoiding artefacts due to the repetitive/periodic

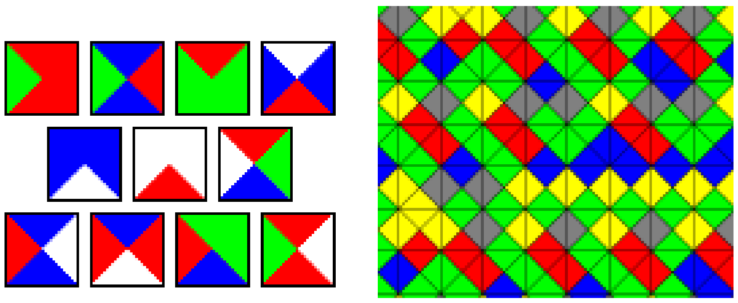


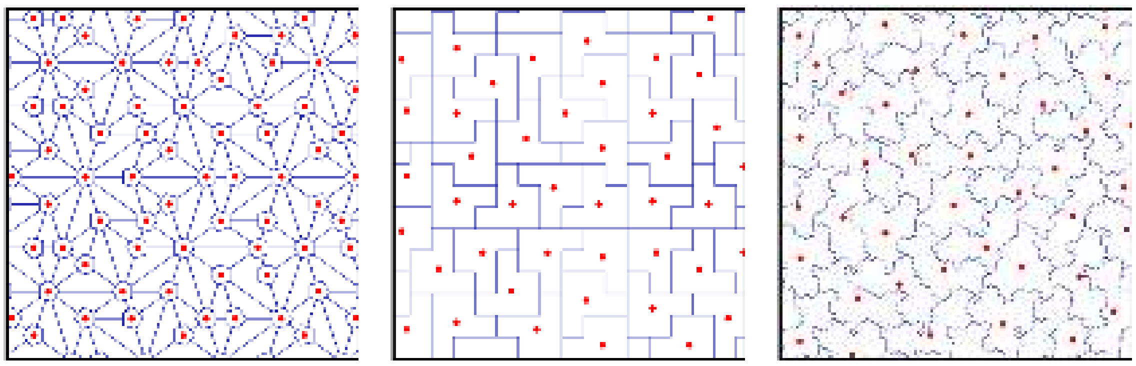
Fig. 3.20.: Wang Tiles paving.

structure of the paving, and of the repetition of the tiles. Those repetitions would lead to aliasing when used in Monte Carlo rendering. Furthermore, we need to maintain consistency at the boundaries between tiles. A whole family of Tiled-based methods are based on Wang tiles [Hil+01; Coh+03; LD05] (Fig. 3.20). Wang tiles can be used for an aperiodic paving of space (meaning without periodic structure) with square tiles (note that those tilings are not mathematically aperiodic, but as the tiling is randomly constructed, there is a probability 1 that it will not be periodic). To handle the boundary issues, each edge of each square tile is associated with a color. The final tiling can then only be composed of tiles with similar facing edge color, under the constraint that no tile can be rotated.

To use this tiling for Blue Noise, one fills the tiles with precomputed sets of Blue Noise samples. Those sets can be generated by a constrained Lloyd's relaxation, where this relaxation is additionally constrained on the edges to maintain a coherency between edges of the same color. Amongst such methods, the most notable is [Kop+06]. This algorithm, contrary to [Hil+01; Coh+03; LD05], supports adaptive sampling and random direct access of samples. Furthermore, the precomputation of their tiles takes only a few minutes and weight only a few mega bytes of storage. In their experiments, authors claim to generate a set of 8 tiles with 2048 samples each in around 20 minutes, stored in a 6 megabyte file.

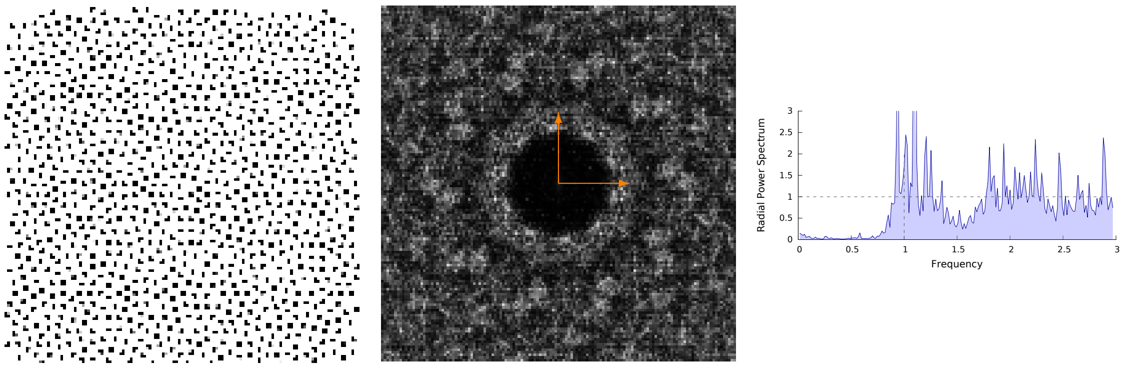
The drawback of this method is that it uses several samples per tiles. This implies that to generate an arbitrary number of samples, one needs to rely on a ranking algorithm. This ranking [Ost07] is not an off-line process and does not allow a fine control for adaptivity. Furthermore, storing the tiles or the recursive paving rules can be memory costly or they may be too little available tiles to avoid repetition in Monte Carlo rendering context.

Other methods use more complex tilings with a single sample per tile (Fig. 3.21 on the next page). This allows for a finer control for adaptivity and avoids the memory cost of tile storage. However, such methods are even more prone to aliasing than the previous ones. This is the case of Penrose tiling [Ost+04]. This method presents a recursive definition of a Penrose tiling where each tiles contains a sample. Furthermore, a set of precomputed correctional offsets (from a Lloyd's relaxation) is used to ensure Blue Noise properties for the final set. However, the obtained set presents several peaks in its Fourier spectrum (Fig. 3.22 on the facing page). To alleviate that, one can use polyominoes to tile the domain [Ost07]. The process of sampling with polyominoes is almost identical as sampling with penrose tiles, with a different correctional



Penrose Tiling [Ost+04] Polyominoes Tiling [Ost07] Irregular Trihexes tiling [Wac+14]

Fig. 3.21.: Sampling patterns from [Ost+04],[Ost07] and [Wac+14] and their underlying paving system.



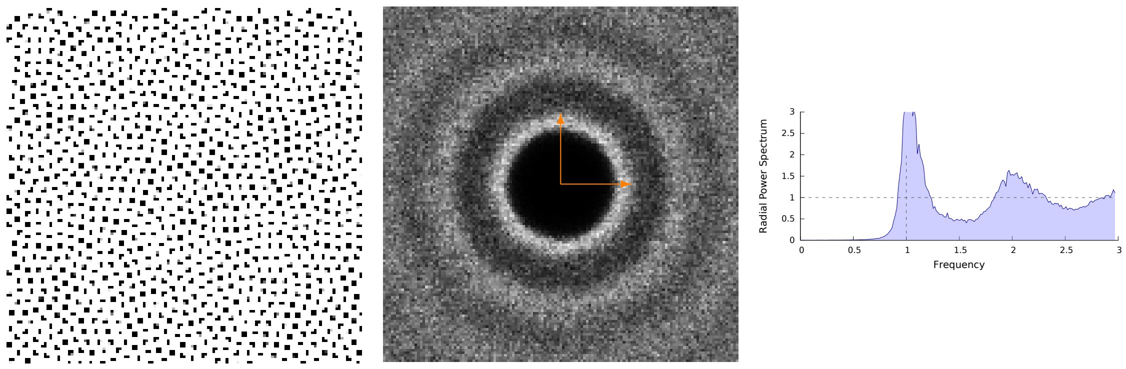
P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.22.: Characteristics of the Penrose tiling sampler [Ost+04].

offsets table. Even though sampling with polyominoes effectively reduces the amount of aliasing, it does not remove it entirely.

Through the usage of irregular trihexes, [Wac+14] achieved a tiling that is both recursively subdividable and without any obvious regularity. The resulting sampling set thus have a Fourier spectrum that is without peaks. Furthermore, their method also allows for user defined Fourier spectrum. The Fourier spectrum is controlled by the set of correctional offsets for the samples positions in the tiles. By changing such offsets, the final Fourier spectrum can be changed. However, due to its irregularity, their tiling patterns requires a table storing the subdivision rules. Combined with the table containing the offsets, this algorithm becomes very memory costly. All together, the tables given by the authors for the subdivision rules and the Blue Noise offsets weight about 2 gigabytes.

Finally, a notable method for Tiled based Blue Noise is the use of AA Patterns [Ahm+15] (Fig. 3.23 on the next page). As noted by [Wac+14], the tile storage can grow up to gigabytes of memory. Here, the authors aim at using tiles with an implicit definition, AA patterns. Those patterns come from error quantization and are used for ornamental purposes. By using those patterns as tiles and storing only the displacement vectors within each tile, AA patterns allow for tiled based Blue Noise generation without any memory costs. Their spectrum is also peak



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.23.: Characteristics of the AA Pattern sampler [Ahm+15].

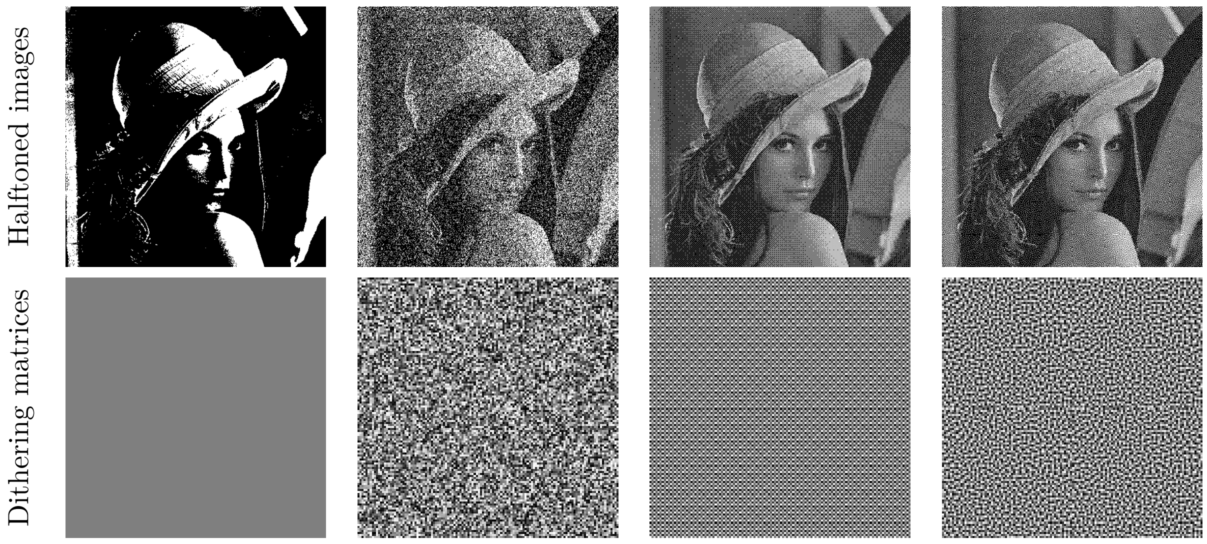
free due to the aperiodic tiling resulting from AA patterns. However, they no longer allow for adaptive sampling. Very recently, [Ahm+17] presented a method that generates Blue Noise from a regular grid. This scrambling allows for a very efficient handling of the recursivity and of neighbouring tiles.

Thanks to their recursive paving, all tiled-based methods are naturally suited for adaptive sampling. An important number of those methods where originally calibrated for stippling purposes. This means they aim at representing an image a set of dots, which can have applications, for example, for printing purposes. Another class of algorithms that can be used for the same purpose, the dithering methods. Such methods will be the topic of the following section.

3.2.4 Dithering based methods

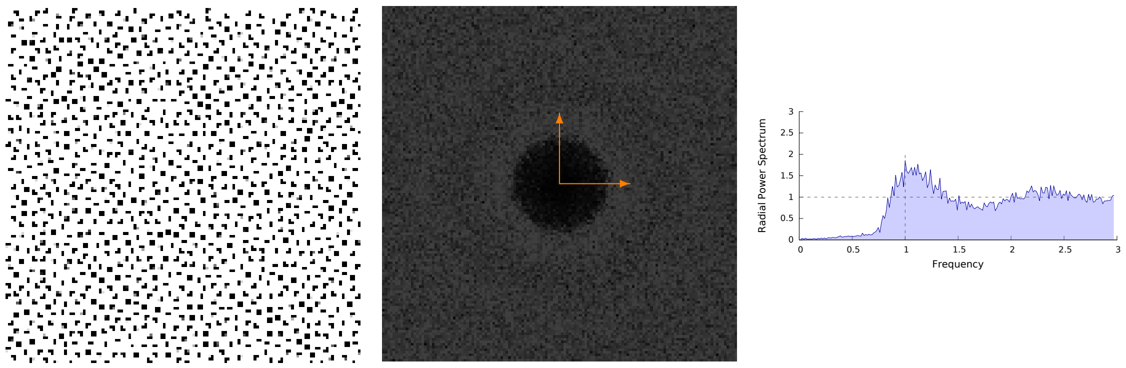
Originally, dithering was used when printing images. Printers only place ink dots on paper, therefore, a black and white image must be stippled (halftoned) to be printed. Dithering is used to prevent regularities in the halftoned image. There are several dithering algorithms [Uli87]. The easiest one is to simply threshold the original image, meaning that ink is placed for every point that is above a given threshold (Figure 3.24 on the facing page (a)). However, with such method, there is a lot of detail loss. To avoid that, this threshold can be made random, but this results in a lot of noise (Figure 3.24 on the next page (b)). Instead, one can use ordered dithering. A dithering matrix is precomputed and used as threshold (Figure 3.24 on the facing page (c)). Among such matrices, some are tuned for Blue Noise sampling patterns. This is such matrices that will interest us here (Figure 3.24 on the next page (d)).

Among the first methods to do Blue Noise dithering is [MP92]. They iteratively build dithering matrices from the difference between the Fourier transforms of the current matrix and the new matrix. Following this work, [Pur+94] tried to generate such matrices with an algorithm very similar to dart throwing. Still another method was proposed by [NBJ97] to generate such matrices from genetic algorithms. Here, the method that interests us more is the very recent method from [Cor+17] (Fig. 3.25 on the facing page). They pre-compute matrices using the method from [Pur+94] and then tile those matrices to generate a point set of arbitrary size.



(a) Constant (b) Random (c) Bayer (d) Blue Noise

Fig. 3.24.: Halftoned images from various dithering algorithms.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.25.: Characteristics of the Forced Random sampler [Cor+17].

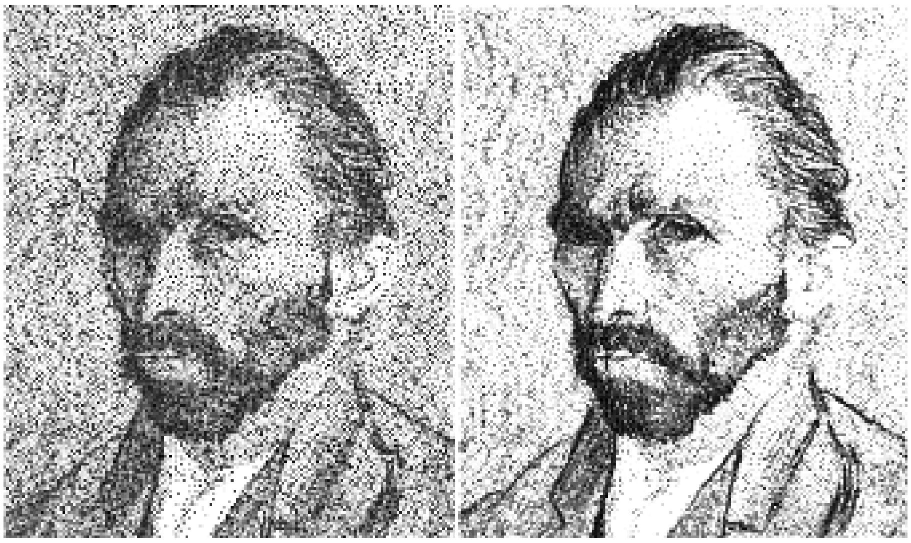


Fig. 3.26.: (left) Stippling with isotropic Blue Noise [Bal+09] and (right) with anisotropic Blue Noise [Li+10a]. Figure courtesy of [Li+10a]

.

With such dithering algorithms, we completed our tour of the main families of methods generating Blue Noise sampling patterns. However, this list is in no way exhaustive (far from it). Other notable algorithms exists that did not directly aimed at generating Blue Noise sampling distributions but that still may be considered as Blue Noise samplers. In the next section, we will detail some of those algorithms.

3.2.5 Other methods

In this section, we present methods that generate variants of the classical Blue Noise sampling patterns. Some try to generate anisotropic or Multi-class Blue Noise, when other aim at targeting a user defined Fourier spectrum, or improving over the original Blue Noise spctrum.

Anisotropic and Multi Class Blue Noise

Those methods are of little use for Monte Carlo integration. However, they are notable variants of the Blue Noise sampling patterns and are presented here for the sake of exhaustivity. The anisotropic Blue Noise presented in [Li+10a], is generated from an adaptation of the Dart throwing and the Lloyd's relaxation algorithm. The idea is to generate samples along a given vector field, mostly for stippling and importance sampling of surfaces (Fig. 3.26). For object placement, in many situations you may have to organize several families of objects. In [Wei10], the authors present a method for Multi-class Blue Noise. They also adapt the Dart throwing and Lloyd's relaxation to handle the attachment of an identifier to each sample, identifying the class of this sample.

Controlling the Fourier spectrum

A few methods were devised allowing to generate samples matching a user-given Fourier Spectrum, usually based on methods like Dart throwing or Gradient descent optimization. The versatility of such algorithms is very important as it could help creating tailored sampling patterns when rendering a given known scene.

The first paper to note is the one from [OG12]. It uses either Lloyd's relaxation or a Dart Throwing algorithm to generate a sampling pattern matching a particular Pair correlation function (Section 2.3 on page 25). For the Dart Throwing variant, the method changes the conditions of acceptance or rejection of the samples. When adding a new sample, the PCF (see Section 2.3 on page 25) of the new set is computed. The distance between this PCF and the targeted PCF is computed using a l_∞ or a l_p distance between the two PCFs. A sample is accepted if this distance between PCFs is below a certain threshold. Similarly, [OG12] adapted the Lloyd's algorithm by weighting the gradient in the optimization with the PCF. As it is based on simple Dart Throwing or simple gradient descent, this method cannot generate massive amounts of samples. Furthermore, this method finds it more difficult to generate very regular structures, as the PCF (when used with Euclidean distance) is an isotropic measure. But, as the PCF is also a s-D measure, this methods can generate pointsets in higher dimensions.

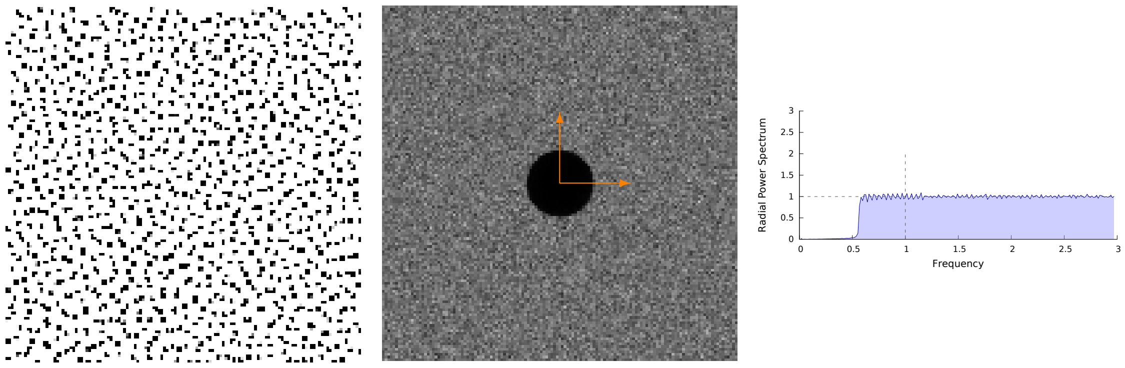
This idea of controlled Fourier spectrum was also developed in [Zho+12], where instead of using PCFs, they use differential analysis from [WW11] and a gradient descent analysis. Similarly to [OG12], as their fitting is based on a isotropic measure, they find it difficult to generate structured sampling patterns.

Finally, we can note the recent paper of [Rov+17], that uses the PCFs to generate sampling pattern with adaptive density and correlations. Authors provide a framework allowing to perform consistent analysis of such sets. However, even though such adaptive correlation can be very useful for stippling, object placement or reconstruction, it is still unknown how they behave on a context of Monte Carlo rendering.

We point out that all tiled based methods allow for a certain amount of spectrum control by changing the offset table.

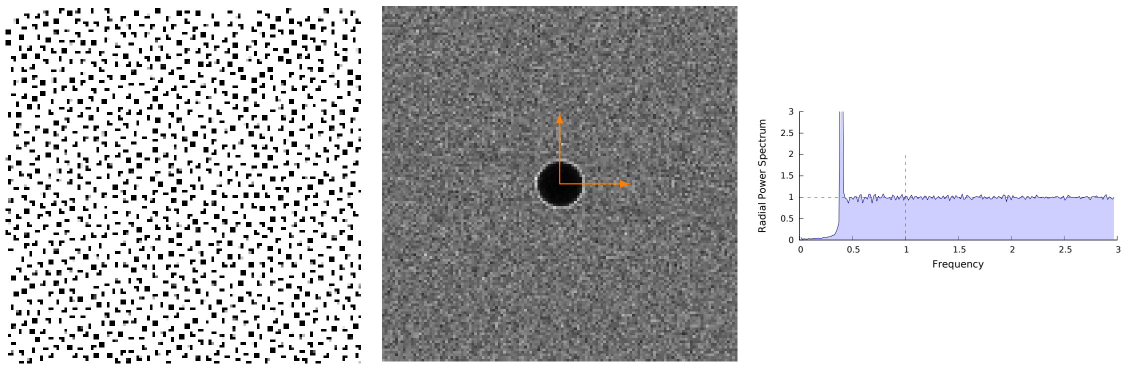
Variants on the Blue Noise spectrum

Other sampling patterns exist that aim at a particular Fourier spectrum, improving over the original Blue Noise spectrum. However, for the following methods, the aim is no longer to allow user control of the Fourier spectrum but to generate a point set with a Fourier spectrum that was chosen to have the largest possible zero region or to avoid the oscillation in the higher frequencies (or both).



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.27.: Characteristics of the Step Blue Noise sampler [Hec+13].



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.28.: Characteristics of the Single Peak Blue Noise sampler [Hec+13].

The first of such methods is the *Step blue noise* [Hec+13]. The authors aimed at obtaining a radial spectrum in the shape of a step function (Fig. 3.27). Here, the authors mostly aim at having a flat final spectrum. This is motivated by the idea that the high amount of energy in some frequencies and the accompanying oscillations are the part of the spectrum that generate most of the noise in the resulting image. In the same paper, they also propose the Single-Peak blue noise function, that instead of presenting a range of frequencies with higher energy, presents a single higher peak (Fig. 3.28). In both cases, the spectra are obtained from a gradient optimization.

A very recent paper [Kai+16] tried to increase the size of the zero region to its theoretical maximal size. To do so, they developed the Stair Blue Noise sampling pattern. They generate such a set from a target PCF, and use a least square optimization to optimize the PCF of a given set to match the Step Blue Noise target. The asset of this method is that it allows for a larger zero region than other methods, which reduces the variance in Monte Carlo integration [Pil+15].

We recall that in this document, we are focusing on unbiased samplers, therefore, in the context of this PhD, reducing this variance is equivalent to reducing the integration error. It is this property that motivates research for more efficient and higher dimensional Blue Noise sampling patterns. However, there are other ways to reduce this integration error. Another possibility is to rely on sampling patterns with a mathematically optimal uniformity. Such samplers are referred to as *Low discrepancy* samplers, and will be the topic of the next section.

3.3 Low discrepancy samplers

A sampler is usually said to be Low discrepancy if its l_∞ star discrepancy in s dimensions decreases in $O(\log(n)^{s-1}/n)$ when the number of samples n increases. Most of those sets are deterministic so it makes no sense to evaluate their variance properties. Instead, we simply evaluate their integration error, using the theorem of Koksma-Hlawka [Hla61]. This theorem states that the discrepancy of a sampler is an upper bound over its integration error, as detailed in Section 2.4.1 on page 28. This leads to a maximal error in $O(\log(n)^{s-1}/n)$, when using a QMC estimator with n samples. In this section, we present a list of existing samplers whose goal is to ensure the best possible uniformity.

The issue with most those methods is that the resulting sampling patterns present a very regular structure and are therefore very prone to aliasing. Furthermore, as they are deterministic, it becomes difficult to decorrelate the samples between pixels. A few methods exist to address both those issues. Those algorithms are referred to as *scrambling*.

Here, we first present existing lattices samplers, before presenting common digital nets and digital sequences. We will finish by detailing some of the scrambling methods.

3.3.1 Lattices

Lattices are usually avoided in Monte Carlo rendering as there is no sampling pattern that generates as much aliasing as regular grids. However, such structures present no energy at all outside the spectral peaks, therefore, for particular integrands with limited frequencies, they can be useful. A lattice point set is defined as

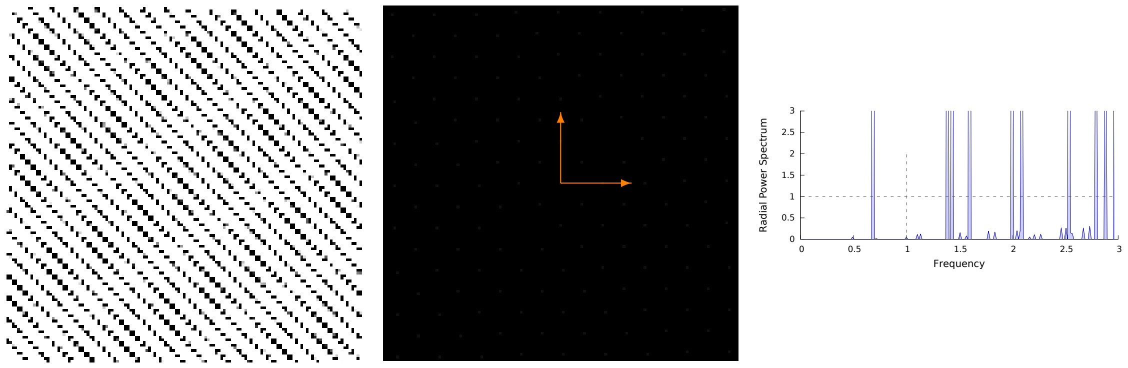
Definition 1 ([Lem09], Def 5.2) *For a given dimension s, a lattice point set P_n is defined by an integration lattice L_s of the form*

$$L_s := \{v_1\mathbf{w}_1 + ... + v_s\mathbf{w}_s, \mathbf{v} \in \mathbb{Z}^s\} \tag{3.12}$$

where the s vectors $\mathbf{w}_1, \cdots, \mathbf{w}_s$ in $\mathbb{R}^s$ - which form a basis - are linearly independent over the rational numbers and are such that $\mathbb{Z}^s \subset L_s$. The corresponding point set is obtained as

$$P_n := L_s \cap [0,1)^s. \tag{3.13}$$

In other words, the points in P_n are obtained by taking all integer linear combinations of the vectors that fall in $[0,1)^s$. The resulting number of points n in P_n can be shown to be equal to $1/|\det(W)|$, where W is the $s \times s$ matrix whose i^{th} row is $\mathbf{w}_i$.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.29.: Korobov point set with $a = 7$ [Kor59]. Zoom to see the peaks in the Fourier spectrum.

To reduce the number of possible bases, we can rely on the rank r of a lattice. A Rank-1 lattice is defined as

$$P_n := \left\{ \frac{i}{n}(z_1, \cdots, z_s) \mod 1, i = 0, \cdots, n-1 \right\} \tag{3.14}$$

where $z_i \in \mathbb{N}$. The vector $\mathbf{z}$ is referred to as the *generating vector*. An interesting property of such lattices is that they can be made fully projection regular simply by choosing the values for z_i to be prime with n.

One example of construction for the vector $\mathbf{z}$ is Korobov point sets [Kor59] (Fig. 3.29).
A Korobov point set P_n is created from a generator value a, and the following $\mathbf{z}$ function,

$$\mathbf{z} := (1, a, a^2 \mod n, \cdots, a^{s-1} \mod n) \tag{3.15}$$

$$P_n := \left\{ \left(\frac{i}{n}\mathbf{z} \right) \mod 1, i = 0, \cdots, n-1 \right\}. \tag{3.16}$$

When a and n are prime relatively to each other, Korobov point sets are fully projection regular.

Dammertz and Keller introduced those lattices for rendering purposes [DK08]. Their idea is to perform an exhaustive search over the Korobov generators to maximize the minimum distance between samples of the grid. This increases the size of the zero region in the Fourier Spectrum. As such Rank 1 lattices are very efficient to compute, such sampling pattern can prove very useful if we can ensure that there are no frequencies above this region.

For all previously presented grids, the computation of the samples requires a prior knowledge of the total number of samples to generate. To overcome this limitation, Hickernell introduces *extensible lattice sequences* [HH97]. The i^{th} sample of such sequence is generated from a vector $\mathbf{z}$ with the following function

$$\mathbf{v}_i := \phi_b(i-1)\mathbf{z} \mod 1, i \geq 1 \tag{3.17}$$

where ϕ_b is the radical inverse function in base b. In most cases, the $\mathbf{z}$ vector is found through exhaustive computer search [Hic+00; GL07].

However, even with an optimal $\mathbf{z}$ vector, we can point that in a typical rendered scene, we observe a very wide range of frequencies. Therefore, in most situations, the grid structure of the above mentioned methods will create massive aliasing artefacts. Instead, we will rather use other low discrepancy sampling patterns, the digital nets and sequences.

3.3.2 Digital Nets and Sequences

Before diving into the details of the algorithms generating digital nets and digital sequences, we will see more precisely what exactly is a digital net, and what is a digital sequence. The name digital net refers to all s-D point sets in the domain $[0,1)^s$, defined in a basis b, that are (t,k,s)-nets (see Definitions 2 and 3). Roughly speaking, such point sets contain b^k samples, and if you partition their domain into intervals that are fractions of b^q, you will have the same amount of samples inside each interval. When $q = 0$, there is a single interval containing all samples. The smaller the factor $t = k - q$, (with q the highest possible value inducing a valid partitioning) the better the uniformity of the net. More formally, Lemieux [Lem09] derives the definitions of (t,k,s)-nets from the concept of $(q_1, \cdots, q_s)$-equidistribution.

Definition 2 ([Lem09, Def 5.6]) *Let $q_1, \cdots, q_s$ be non-negative integers, and let $q_1 + \cdots + q_s$. A point set P_n with $n = b^k$ points is $(q_1, \cdots, q_s)$-equidistributed in base b if every cell (or elementary interval) of the form*

$$\mathcal{J}(\mathbf{r}) := \prod_{j=1}^{s} \left[\frac{r_j}{b^{q_j}}, \frac{r_j+1}{b^{q_j}}\right), \tag{3.18}$$

for $0 < r_j < b^{q_j}$, $j = 1, \cdots, s$, contains b^{k-q} points from P_n.

The intervals $\mathcal{J}(\mathbf{r})$ are called *dyadic intervals.*

Definition 3 ([Lem09, Def 5.7]) *A set P_n containing $n = b^k$ points is called a (t,k,s)-net in base b if it is $(q_1, \cdots, q_s)$-equidistributed in base b whenever $q \leq k - t$.*

Figure 3.30 on the following page illustrates a $(0,4,2)$-net in base 2. This set contains $2^4 = 16$ samples, that are equidistributed for all intervals which sizes are in $\frac{1}{2^{q_0}} \times \frac{1}{2^{q_1}}$ with $q_0 + q_1 \leq 4$. Figure 3.31 on page 61 illustrates a $(2,4,2)$-net in base 2. This set contains $2^4 = 16$ samples, that are equidistributed for all intervals which sizes are in $\frac{1}{2^{q_0}} \times \frac{1}{2^{q_1}}$ with $q_0 + q_1 \leq 4 - 2$. Note that, as stated in [Nie92], $(0,m,s)$-nets in base $b = 2$ can only exist for $s \leq 3$.

This leads us to the concept of digital sequence. A digital sequence refers to point sets of infinite size. This means that from a digital sequence, we can generate a point set P_n with n samples, and then, with the same algorithm, enrich this point set with m new samples to generate a new set P_{n+m}. Usually, such sequence is defined implicitly. A digital sequence is a (t,s)-sequence, which is an extension of a (t,k,s)-net.

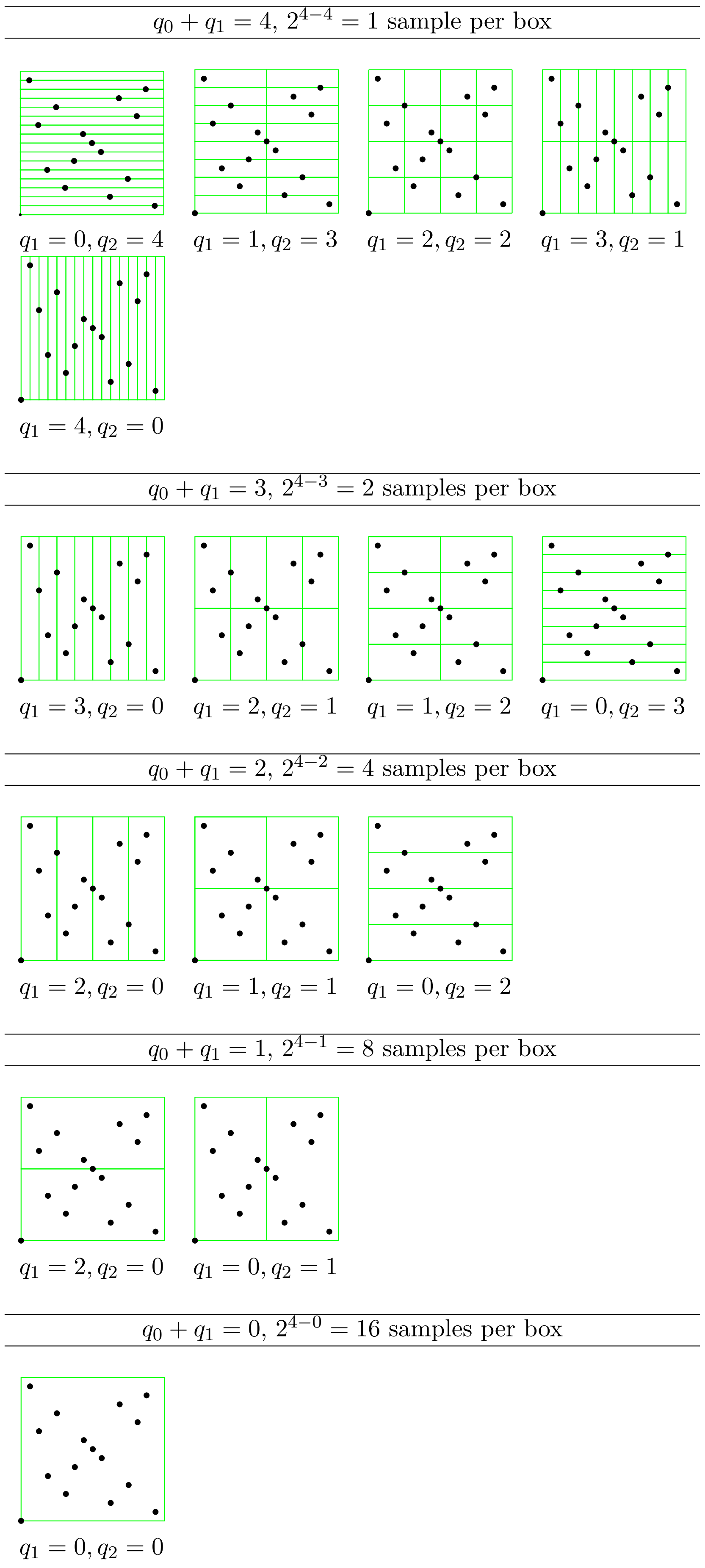


Fig. 3.30.: $(0, 4, 2)$-net in base 2.

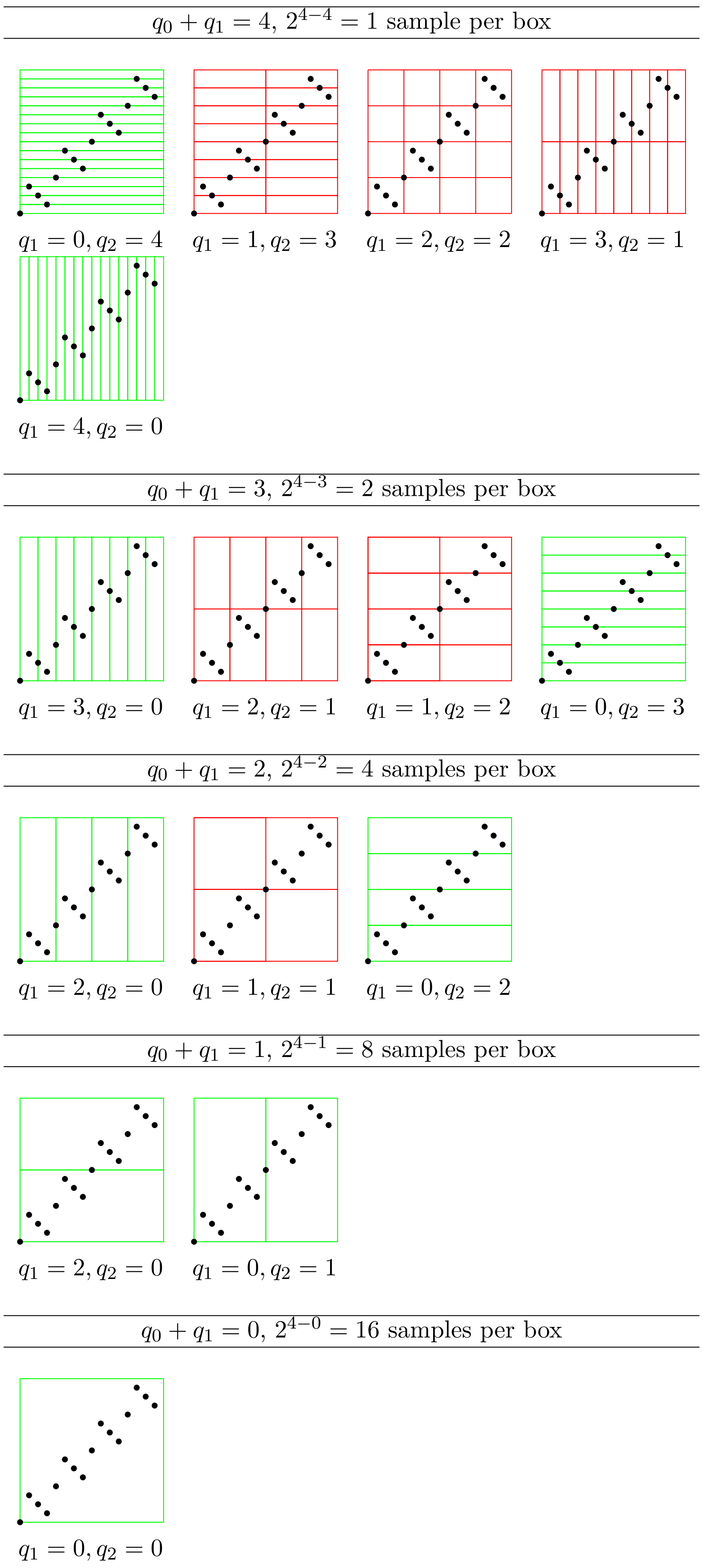


Fig. 3.31.: $(3, 4, 2)$-net in base 2.

Definition 4 ([Lem09, Def 5.7]) *A (t, s)-sequence is a sequence of points for which each b-ary segment of the form $\mathbf{v}_{(l+1)b^{k-1}}, \cdots, \mathbf{v}_{(l+1)b^{k}-1}$ with $k > t$ and $l > 0$ is a (t, k, s)-net in base b.*

Furthermore, if a sequence is a (t, s)-sequence, we have the following upper bound on its l_∞ star discrepancy ([Nie88, Eq 3.])

$$\mathcal{D}_* \leq c_s(\log(n))^s + O\left(\frac{\log(n)^{s-1}}{n}\right) \tag{3.19}$$

where c_s is a constant depending on the basis b, the number of dimensions s, and most important, on the t value. Formally, we have ([Lem09, Eq 5.6])

$$c_s := \frac{1}{s!} b^t \frac{b-1}{2\lfloor b/2 \rfloor} \left(\frac{\lfloor b/2 \rfloor}{\log(b)}\right)^s. \tag{3.20}$$

Equation (3.19) and (3.20) thus express how the t factor directly affect the low discrepancy property of the sequence.

For the (t, ks)-net definition, the t value is the smallest $t = q_0 + \cdots + q_{s-1}$ for which P_n is $q_0, \cdots, q_{s-1}$-equidistributed. Similarly, for a (t, s) sequence, the t value is the smallest such that the subsets of P_n are (t, k, s)-nets.

Now that the concepts of digital nets and sequences are defined, we will see the existing methods to generate such samplers. What definition 4 means, is that there exist an implicit link between the discrepancy and the net properties of a sampler. Even though the concept of low discrepancy is defined asymptotically and thus cannot be used to describe a single point set, we can however say that any set that is a (t, k, s)-net has a guaranteed uniformity, as good as its t value is low.

Many digital nets or sequences follow the same construction. The coordinate in each dimension d of each sample can be generated from an analytic function f_d, taking as parameter the index of the sample in the set. A sample $\mathbf{x}$, the i^{th} sample of a s-D set, can be represented as

$$\mathbf{x}^{(i)} := (f_1(i), ..., f_s(i)). \tag{3.21}$$

The function $f_d(i)$ calculates the coordinate in the d^{th} dimension of the i^{th} sample of the set. Thus, the values of the f_d functions are values in the domain of the sampling pattern which is usually $[0, 1)^s$. In most situations, the f_d function computes the coordinate of the i^{th} sample by performing bitwise modifications to the index i.

As was detailed Section 1.4 on page 8, the bit representation in base b of a value $x_d \in [0, 1)^s$ is as follows:

$$x_d = \sum_{i=1}^{\infty} a_i b^{-i} \tag{3.22}$$

also represented as

$$x_d = (.a_1 a_2 \cdots a_N)_b \tag{3.23}$$

where a_k is the k^{th} digit of the b-ary representation of x_d.

Following this, the function f_d can be noted as

$$f_d(n) := C_d \otimes (a_0, a_1, \cdots, a_N)^T \tag{3.24}$$

where C_d is a $N \times N$ matrix, and a_k is the k^{th} digit of the b-ary representation of the index n, with N the total number of digits needed to represent n in base b. Note that as the number of samples increases, the value of n increases as well, and therefore the value of N increases too as more digits will be needed to represent a higher n in base b.

The a_k values are defined in the $\mathbb{F}_2$ domain. This domain is a ring over $\mathbb{Z}/2\mathbb{Z}$ with operators for addition $\oplus$ and multiplication $\otimes$. The C_d matrices, defined in $\mathbb{F}_2^{N\times N}$, are called *generating matrices*. From here, what differentiates most methods generating digital nets and digital sequences are the C_d matrices used.

Note that if we want our resulting sampling pattern to be a $(0, k, s)$-net, the matrices C_d must follow some constraints:

Theorem 1 ([Nie92, Theorem 4.28]) *Let P_n be a point set of size $n = 2^k$ in base 2 and dimension s generated from the matrices $C_1, \cdots, C_s$. P_n is a $(0, k, s)$ net in base 2 if for all $\mathbf{d} = (d_1, \cdots, d_n)^T \in \mathbb{N}_0^s$ with $|\mathbf{d}| = m$ the following holds*

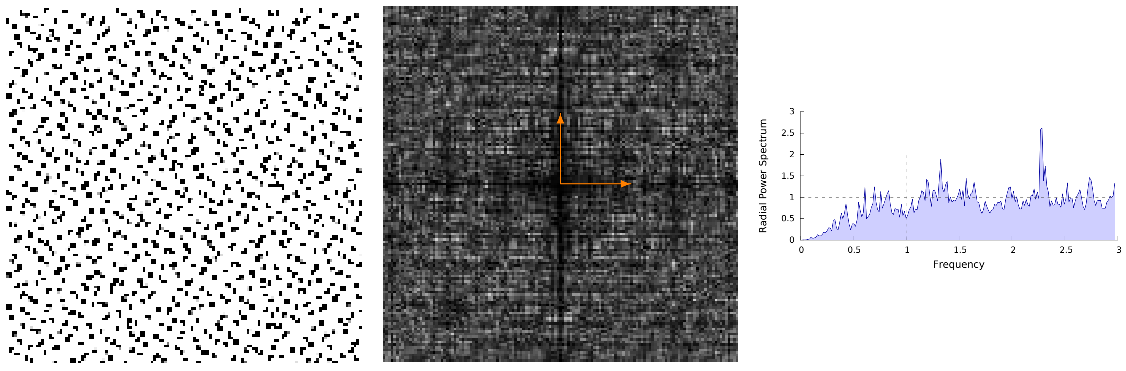
$$det(C^{\left(\sum_{j=1}^{s} d_j\right)}) \neq 0 \tag{3.25}$$

where

$$C^{\left(\sum_{j=1}^{s} d_j\right)} := \begin{pmatrix} c^1_{1,1} & \cdots & c^1_{1,m} \\ & \vdots & \\ c^1_{d_1,1} & \cdots & c^1_{d_1,m} \\ & \vdots & \\ c^s_{1,1} & \cdots & c^s_{1,m} \\ & \vdots & \\ c^s_{d_s,1} & \cdots & c^s_{d_s,m} \end{pmatrix} \in \mathbb{F}_2^{m\times m} \tag{3.26}$$

with $C_j := \left(c^j_{k,l}\right)_{k,l=1}^{m}$ for $j = 1, \cdots, s$. This means $C^{\left(\sum_{j=1}^{s} d_j\right)}$ is an $m \times m$ matrix consisting of the first d_j rows of the generator C_j for all $j = 1, \cdots, s$.

A careful reader will have noted that Theorem 1 only applies to digital nets. Further constraints apply for generating digital sequences. We would like to first point out that if a matrix C_d is not upper triangular, the resulting sampler is not a $(0, s)$-sequence. To be a sequence, a sampler has to be able to increase the size of the C_d matrices on the fly, without changing the result of



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.32.: Halton point set [Hal64].

the computation of the previous indices. Therefore, all values below the diagonal must be 0, to act as neutral elements in the computations and therefore preserve the position of the previous samples. When generating digital nets, we do not need to increase the size of the matrix on the fly as we already know the number of samples required. Therefore, we also do not need the matrix to be upper triangular. We will detail this further as we present the common existing digital sequences and digital nets.

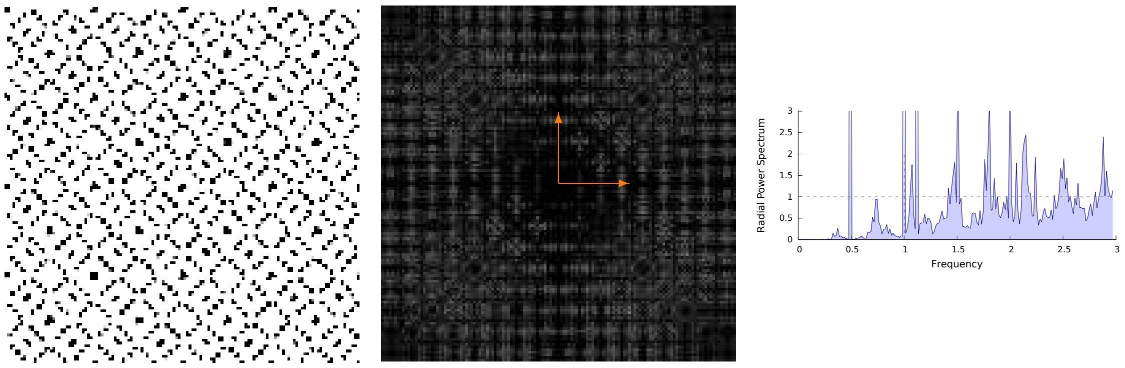
Digital Sequences

The first and simplest digital sequence is the Van der Corput sequence [Cor35]. This 1-D sequence generates the coordinates of the i^{th} sample by reversing its digit in base 2. This translates in Equation 3.24 on the previous page as

$$(a_0, a_1, \cdots, a_N)^T = \mathbf{I} \otimes (b_0, b_1, \cdots, b_N)^T \tag{3.27}$$

where $\mathbf{I}$ is the identity matrix, $x = (.a_0 a_1 \ldots a_N)_b$ and $i = (b_N \ldots b_1 b_0)_b$. The Van der corput sequence is often denoted ϕ.

This sequence was extended by Halton [Hal64] to a s-D sequence through the use of different bases in each dimension. When you perform a 1-D bit reversal in a basis b, it sums up as dividing your domain into b^n intervals and putting one sample inside each interval. In s-D, if the bases used are prime numbers on all dimensions, one is guaranteed that the intervals will not be aligned between dimensions and therefore the sequence will be low discrepancy. Furthermore, this use of several basis also helps reducing the regularities and therefore the peaks in the resulting Fourier spectrum (Fig. 3.32).



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.33.: Sobol point set with $t = 0$ [Sob67].

The Sobol Sequence

For sequences using a single basis, the most famous (t, s)-sequences is probably the Sobol sequence [Sob67] (Fig. 3.33). Defined in base 2, this sequence builds the matrices C_d from primitive polynomials. The choice of the primitive polynomials used will affect greatly the result of the sampling pattern (Fig. 3.34 on the following page).

The list of all the primitive polynomials usable for this sequence can be found in [Oei]. Each polynomial is referred to using its index in this list. We build each C_d matrix from the polynomials using the following algorithm, which can also be found in [JK08]. We remind that each dimension uses a different C_d matrix, generated from its corresponding primitive polynomial.

We start with a primitive polynomials of degree k in the field $\mathbb{F}_2$, denoted as

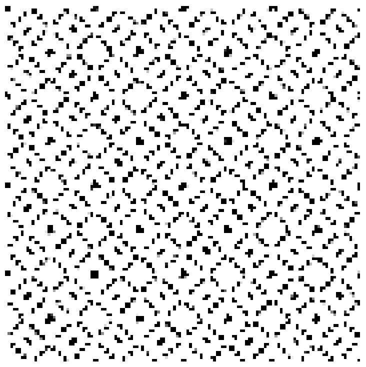
$$x^k \oplus a_1 x^{k-1} \oplus a_2 x^{k-2} \oplus \cdots \oplus a_{k-1} x + 1 \tag{3.28}$$

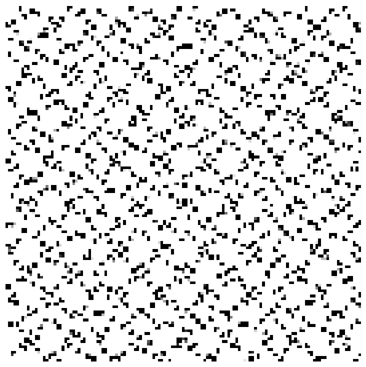
where the coefficients a_i are either 0 or 1. A set of numbers $m_1, m_2, \cdots$ is built from the following recurrence relation

$$m_j := 2a_1 m_{j-1} \oplus 2^2 a_2 m_{j-2} \oplus \cdots \oplus 2^{k-1} a_{k-1} m_{j-k+1} \oplus 2^k m_{j-k} \oplus m_{j-k} \tag{3.29}$$

You may have noticed from this equation that it requires an initial subset of m_j value to iterate upon. Those values can be chosen freely, as long as for each m_j with $1 \leq j \leq k$, m_j is odd and m_j is lesser than 2^j. Those conditions ensure that the binary C_d matrix formed for the m_j values will present 1 on its diagonal and will be upper triangular. Those numbers are used to create the set of *direction numbers* $\{v_1, v_2, \cdots\}$ with

$$v_j = \frac{m_j}{2^j} \tag{3.30}$$


(a) Polynomials 1 and $x + 1$, corresponding to indices 0 and 1.


(a) Polynomials $x^2 + x^1 + 1$ and $x^3 + x + 1$, corresponding to indices 3 and 4.

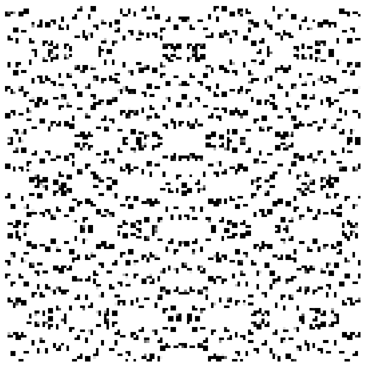

(a) Polynomials $x^3 + x^2 + 1$ and $x^4 + x^1 + 1$, corresponding to indices 5 and 6.

Fig. 3.34.: Sobol sets with polynomials of different degrees

Finally, the C_d matrix is created by concatenating the v_j values, expressed in binary, as its columns. Meaning, if we denote v_j^i the i^{th} bit of the number v_j, we have

$$C_d = \begin{pmatrix} v_1^0 & v_2^0 & v_3^0 & \cdots \\ v_1^1 & v_2^1 & v_3^1 & \cdots \\ v_1^2 & v_2^2 & v_3^2 & \cdots \\ \vdots & \vdots & \vdots & \iddots \end{pmatrix} = \begin{pmatrix} 1 & v_2^0 & v_3^0 & \cdots \\ 0 & 1 & v_3^1 & \cdots \\ 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \iddots \end{pmatrix}. \tag{3.31}$$

The simplification to an upper triangular matrix is guaranteed from the conditions on the initial m_j values.

The Sobol sequence has been proven to be a (t, s)-sequence. However, the higher the degree of the polynomials, the higher the t value and therefore the worse the quality of the sampling pattern. Formally, we have

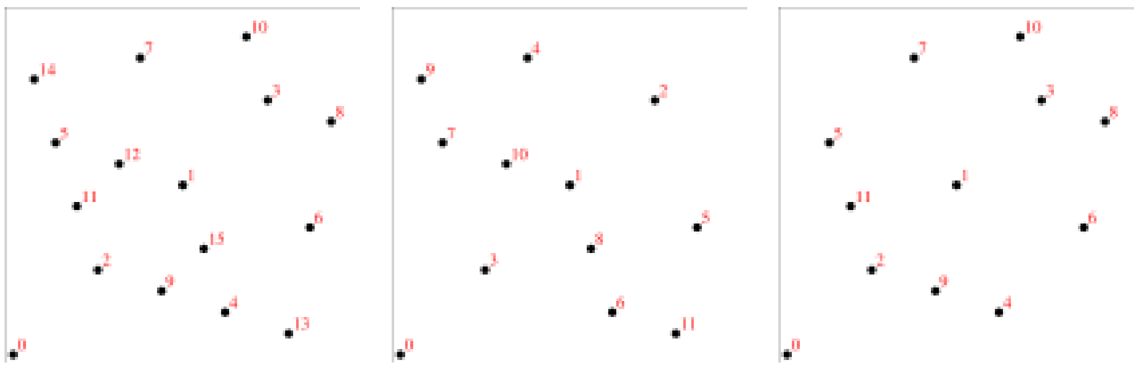
$$t := \sum_{d=1}^{s} k_d - 1, \tag{3.32}$$

where k_d is the degree of the polynomial used in the d^{th} dimension [Lem09]. It is interesting to know that there is a limited number of primitive polynomials of each degree, and that to be a (t, s)-sequence, the Sobol sequence can not use the same polynomial twice. Therefore, the higher the dimension of the point set, the worse its uniformity. This is illustrated in Figure 3.34.

This drawback is actually true for most of the low discrepancy sequences. One way to alleviate this issue is to permute the digits of the sample index before computing its coordinate. This does not change the position of the samples of the sequence but it changes their order in this sequence. This can be done using Gray Code. Each index i is replaced with its Gray Code equivalent, denoted $\mathcal{G}$, as

$$\mathcal{G}(i) := i \oplus \left[\frac{i}{2}\right] \tag{3.33}$$

where the $\oplus$ operator is the logical *xor* operator, and [.] represents the integral part function. Note that the function $\mathcal{G}$ takes as input an integer $i \in \mathbb{N}$, and outputs another integer $\mathcal{G}(i) \in \mathbb{N}$.



P_{16} The first 12 original samples The first 12 Gray code samples

Fig. 3.35.: Application of Gray Code on a Sobol sequence.

Applying Gray Code changes Equation 3.24 on page 63, that generates the d^{th} dimension of the i^{th} sample of the s-D sequence defined from a set of s generative matrices $\{C_d\}$, to

$$(c_N, \cdots, c_0)_2 = \mathcal{G}(i)(a_0, a_1, \cdots, a_N) = C_d \cdot (c_0, c_1, \cdots, c_N)^T. \tag{3.34}$$

The impact of Gray Code is illustrated Figure 3.35.

The Sobol sequence was later extended in several ways. In [Tez95], Tezuka obtained a sequence no longer restricted to primitive polynomials. In [Fau82], the Faure sequence was developed. This sequence aims at building $(0, s)$-sequences in base b, relying on Pascal Matrices raised to the power $j, j = 0, \cdots, s$. Note that this construction requires that $s \leq b$, which prevent a dynamic increase of the number of dimensions. Finally, Niederreiter [Nie88], extended this sequence to bases $b \neq 2$, which were further extended to Niederreiter-Xing sequences [NX95], where t increases linearly with s (instead of exponentially for other sequences), which is the optimal achievable rate.

All the sequences presented above allow for a direct generation of the i^{th} sample, without any prior knowledge of the total number of sample of the point set. This property adds constraints on the sample generation that limit the number of possible sampling patterns. If we choose to remove the sequentiality, we will generate digital nets, which are less constrained than digital sequences and therefore allow for new sampling patterns.

An interesting property of digital sequences is that any digital sequence defined in dimension s can be turned into a digital net in dimension $(s+1)$ [Nie92]. To do so, simply set the matrix C_{s+1} as

$$C_{s+1} := \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ & & & & \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix} \tag{3.35}$$

This can be intuitively defined as setting the $(s+1)^{th}$ coordinate of the i^{th} sample in a point set of size n as $\frac{i}{n}$.

The Hammersley point set [Ham60] is the combination of this property with the Van der Corput sequence (Fig. 3.36 on the facing page). This set is defined in 2-D, where the i^{th} sample $\mathbf{x}$ is defined as

$$\mathbf{x} := \left(\phi(i), \frac{i}{n}\right) \tag{3.36}$$

with ϕ being the Van der corput sequence and n being the size of the point set. Expressed in the framework of Equation 3.24 on page 63, we have

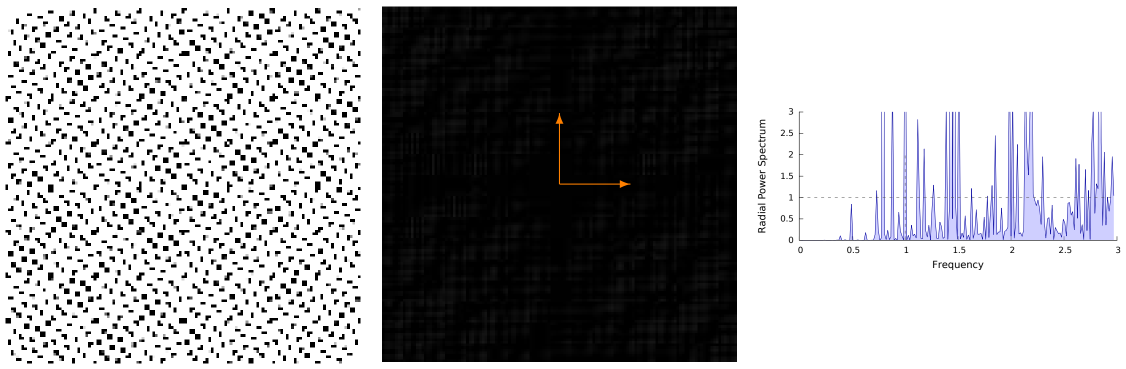
$$C_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{3.37}$$

$$C_2 = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ & & \iddots & & \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix} \tag{3.38}$$

Due to the high structure induced by the implicit definition of the sampling patterns, Digital nets are deterministic and regular sampling pattern. When used in rendering, this generates aliasing and correlation between pixels. To alleviate this, scrambling algorithms were devised and will be presented in Chapter 4.

3.3.3 Conclusion

In conclusion, among existing samples we can denote three major families. The stochastic historical samplers, the Blue Noise samplers, and the Low Discrepancy analytically defined



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 3.36.: Hammersley point set [Ham60].

samplers. Each of those families have its pros and cons, as illustrated on the graphs 3.37 and 3.38.

On the graph 3.37, we roughly compare the performances of the main samplers for each family, with respect to their spectrum quality, and their discrepancy. A spectrum is considered of good quality if it is close to the Blue Noise spectrum and does not present spectral peaks. For the discrepancy, the right most samplers are those that are Low Discrepancy. We can note that the best Low discrepancy spectrum is also the one with the worst Fourier spectrum, and that the discrepancy of the sampler with the best spectrum is only average.

The graph 3.38 regards mostly the practicality of various samplers. A sampler can present the best spectrum and Low discrepancy, if it can't be used in practice it is rather pointless. On this graph, we compare samplers with regard to their computational efficiency (the speed at which they can generate high amount of samples), and their dimensionality (a higher dimensionality means a sampler can generate samplers in higher dimensions). Both those characteristics are mandatory for rendering, as we solve recursive higher-dimensional integrands, and generate several hundreds of samples per pixels. It can be seen on this graph that canonical and Low Discrepancy samplers are very practical, when Blue Noise samplers are almost impossible to use in a rendering context, despite the quality of their spectrum.

Following this, we conclude that historical samplers have a spectrum of reasonable quality (in the sense that is does not generate aliasing) and are efficient for higher dimensions. Blue Noise samplers present an optimal spectrum and are therefore ideal for Monte Carlo Rendering as they have a very low variance and do not generate aliasing. However, such samplers are limited to low number of dimensions, and are not computationally efficient. Finally, the low discrepancy samplers present a very low error in integration, are very efficient even in high dimensions thanks to their analytical definition, and some also allow for iterative addition of samples. However, their structured sampling patterns induce strong aliasing when used in Monte Carlo rendering. Table 3.1 on the following page sums all the integration errors (or MSE) of various samplers, presented in this chapter.

Sampler	Worst Case MSE/Error^2	Discrepancy
Whitenoise	$O(1/n)$	$O(1/\sqrt(n))$
Stratified	$O(1/(n\sqrt{n}))$	
Poisson Disk	$O(1/n)$	
Blue Noise	$O(1/(n\sqrt{n}))$	
Low Discrepancy	$O((\log(n)/n)^2)$	$O(\log(n)/n)$

Tab. 3.1.: Worst case convergence speeds of the integration error when using warious sampling patterns, along with their discrepancy.

In this PhD, we aimed at creating hybrid samplers. Those samplers would have both a Blue Noise spectrum and be Low Discrepancy. This allows them to have the lowest error in integration, without any aliasing, while still being computationally efficient in higher dimensions. Chapter 5 on page 93 will now present a first Low Discrepancy Blue Noise sampler (LDBN) that was developed jointly by us and the Universities of Konstanz and Shengen. This sampler is a first step towards hybrid samplers but it is still limited to 2-D and does not present any sequential properties. Then, Chapter 4 on page 74 will present simplistic methods (inspired from energy minimization or genetic algorithms) to develop multi-dimensional low discrepancy sequences with controllable spectrum. Finally, Chapter 6 on page 107 will improve upon those naive ideas to present a new scrambling technique applied on the Sobol sequence that preserves its sequentiality while both giving a Blue Noise spectrum to its 2-D projections and maintaining their Low Discrepancy property. It also maintains some uniformity of the s-D set.

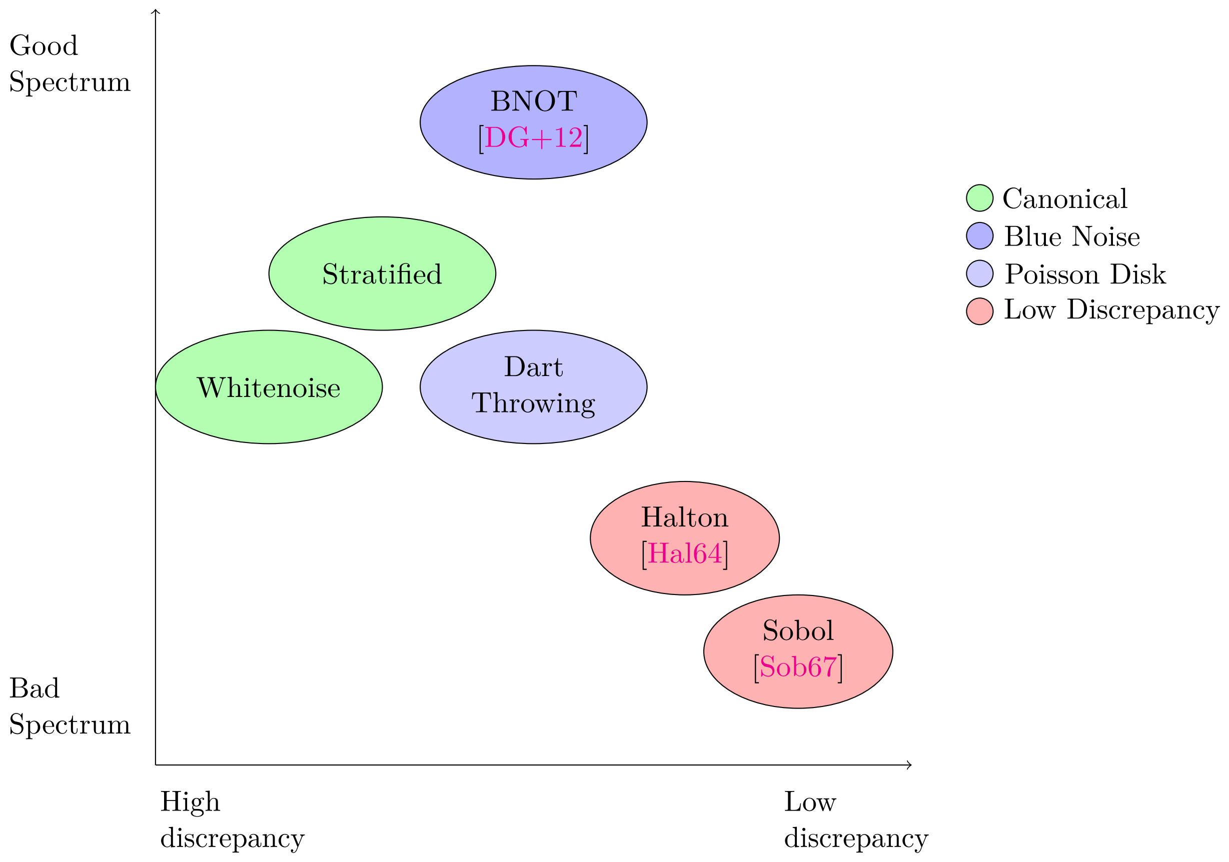


Fig. 3.37.: Overview of common state of the art samplers regarding spectral quality and discrepancy.

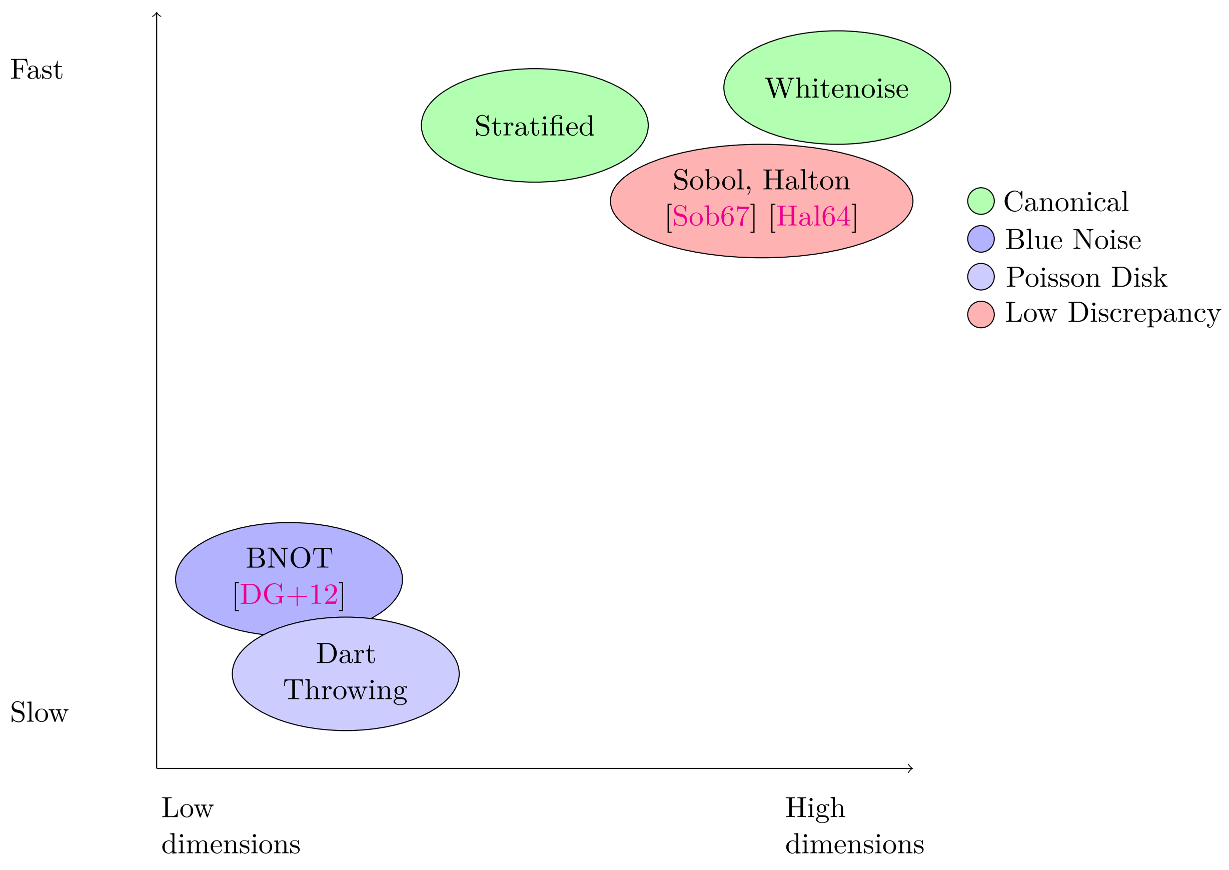


Fig. 3.38.: Overview of common state of the art samplers regarding their computational efficiency and dimensionality.

4 Optimizing a set

As shown in Section 2.1.1 on page 17, the discrepancy of a set creates an upper bound on the error of the Monte Carlo integration using this set. Therefore, the lower the discrepancy the lower the bound on the error. However, most low discrepancy samplers are highly structured, which generates aliasing when used for rendering an image. On the other hand, among stochastic samplers, the best distribution of samples for MSE reduction is the Blue Noise distribution (Section 2.4.2 on page 30). This distribution also does not present any spectral peaks and thus does not generate aliasing. We thus try to create sets that are both Low discrepancy and Blue Noise, in hope to have an optimal low in integration error along with an anti aliased spectrum.

In this Chapter, we present a few naive methods to improve the spectrum or the discrepancy of various samplers. The first category of methods, Section 4.1, aimed originally at randomizing Low Discrepancy analytic samplers. However, we will see that one of those methods, named *Owen's scrambling* [Owe95], also allows to improve the Fourier spectrum of point sets.

In a second time, Section 4.2 on page 79 presents our adaptation of Owen's scrambling that allows to generate samples sequentially. Finally, Section 4.3 on page 80 and Section 4.4 on page 84 presents our early attempts to either improve the discrepancy of a Blue Noise set or to scramble a Low Discrepancy set to give it a Blue Noise Fourier spectrum.

4.1 Randomizing Low discrepancy sets

In general, all the Low Discrepancy point sets and sequences presented Chapter 3 show very regular point placement. This leads to peak in their Fourier spectrum which in turn lead to aliasing. Furthermore, those point sets are deterministic, which is an issue in rendering as using the same point set in each pixel also leads to aliasing due to pixel correlation. In this section, we present some common randomizing algorithms to alleviate those correlations.

4.1.1 Shifting

The easiest method to randomize point sets are the digital shifts. Such methods combine each sample of the set with a single randomly chosen input. They thus do not break the sequentiality of an input sequence.

The most straightforward one is the Cranley-Patterson shift [CP76]. This applies a toric digital shift, chosen randomly, to a point set. More formally, with this method, a sample $\mathbf{x}$ is shifted to create a new sampler $\mathbf{x}'$ from a shift vector $\mathbf{v}$ with the following formula

$$\mathbf{x}' := \mathbf{x} + \mathbf{v} \mod 1, \tag{4.1}$$

where the $+$ operator performs the coordinate wise sum of the two vectors.

This scrambling efficiently decorrelates the pixels in rendering while preserving the Fourier spectrum of the set. Following [KN74, Lemma 1.1], such method preserves the uniformity of the given set. Note that if this scrambling is applied on a Digital net, it does not preserve the net properties of the input set.

If one needs to preserve the net properties of the set, one can use the Digital shift. This method takes a vector at random and performs a xor in base b between this vector and all samples. If we denote $\mathbf{x}''$ the Digital shifting of $\mathbf{x}$ with a vector $\mathbf{v}$, we have

$$\mathbf{x}'' := \mathbf{x} \oplus \mathbf{v}, \tag{4.2}$$

where the $\oplus$ operator performs the coordinate wise xor of the two vectors.

Shifting methods are extremely efficient when it comes to randomizing a point set. However, they pretty much do not do anything else. In the following subsections, we will see other randomizing methods that affect the Fourier spectrum of a point set. We will start by presenting a 1-D scrambling method, that we will reuse in Chapter 5 on page 93 to adapt the spectrum of a Low Discrepancy set. Then, we will present Owen's scrambling, which will be reused in Chapter 6 on page 107.

4.1.2 Chunk based 1-D scrambling

In [KN74], we can find the following theorem,

Theorem 2 ([KN74, Chapter 2, Theorem 4.1]) *Let $P_n := \{x_1, x_2, \cdots x_n\}$ and $P'_n := \{y_1, y_2, \cdots, y_n\}$ be two finite sequences in $[0, 1)$. Suppose $\epsilon_1, \epsilon_2, \cdots, \epsilon_n$ are nonnegative numbers such that $|x_k - y_k| \leq \epsilon_k$ for $1 \leq k \leq n$. Then, for any $\epsilon \geq 0$, we have*

$$|\mathcal{D}_*(P_n) - \mathcal{D}_*(P'_n)| \leq 2\epsilon + \frac{N(\epsilon)}{n} \tag{4.3}$$

where $N(\epsilon)$ denotes the number of $k, 1 \leq k \leq N$, such that $\epsilon_k > \epsilon$.

This theorem starts with two 1-D indexed sets of samples, such that for all i, the i^{th} sample of the first set can be paired with the i^{th} sample from the other set. The theorem then states that, if the difference between paired samples does not exceed a value ϵ, then the difference between the discrepancies of both input sets is in $O(1/n)$, and is bounded by $2\epsilon + n/n$.

In our case, we will use this theorem to evaluate how much scrambling a 1-D Low discrepancy sequence affects the discrepancy of the original sampling pattern. Instead of having two different sequences as input, we will take a 1-D Low Discrepancy sequence $\mathcal{S}$, and then its scrambled version. We then see that if we scramble it locally, the difference between the discrepancy of those sequences is bounded by the size of the scrambled subset. We will come back to this in Chapter 5 on page 93, where we will give formal definitions and formal proofs for this. We will also see how this 1-D theorem can be used in a 2-D context for spectral control of Low Discrepancy sequences.

There exists another method, that scrambles sets of higher dimensions, *Owen's scrambling* [Owe95], (already used for Rendering in [Vea97]). This method allows to preserve the discrepancy of a full s-D set by randomly scrambling each 1-D projections of this set. It can also impact the Fourier Spectrum of the initial set, and therefore served as a basis for most our experiments.

4.1.3 Owen's scrambling

For the particular case of digital nets, advanced scrambling method exists [Lem09]. The most important one for us is the *Owen's scrambling* [Owe95], also referred to as *fully random scrambling.*

This scrambling usually applies on sets that are (t, k, s)-nets, and guarantees to preserve the net properties of the set. It could be applied on other categories of sampling patterns but would not be as easy to implement and we would not have any guarantees on the resulting set.

It relies on swapping some coordinates of some samples, randomly chosen. This swapping is performed independently over each dimension. A s-D point set is permuted using s 1-D permutations, one for each dimension. Therefore, we can detail this scrambling in 1-D without any loss of generality.

Geometrically, a domain Ω is recursively subdivided by a factor 2. Then, a binary tree of boolean flags is used to determine if two subdivisions of space are to be swapped or not. Each flag of this tree corresponds to a pair of neighbouring subsets of the domain, and by filling this tree of boolean flag with random values, we will randomly swap portions of space. This is illustrated in 1-D Figure 4.1 on the next page and in 2-D Figure 4.2 on the facing page.

We now give a more formal definition of this scrambling. For a set of n 1-D samples P_n, if P_n is a $(t, k, 1)$-net, each sample can be expressed as an integer, defined in base 2 by $d = \log_2(n)$ bits. Thus, Owen's scrambling of this set requires a binary tree of depth d. This tree is filled with a set of $n-1$ boolean flags chosen randomly. Formally, a boolean tree is defined as a set of flags $\{\mathbf{B}_j\}$, where j is the binary vector representing the id of this flag at a given depth i. The length of the binary vector j allows to retrieve the level i of a given flag.

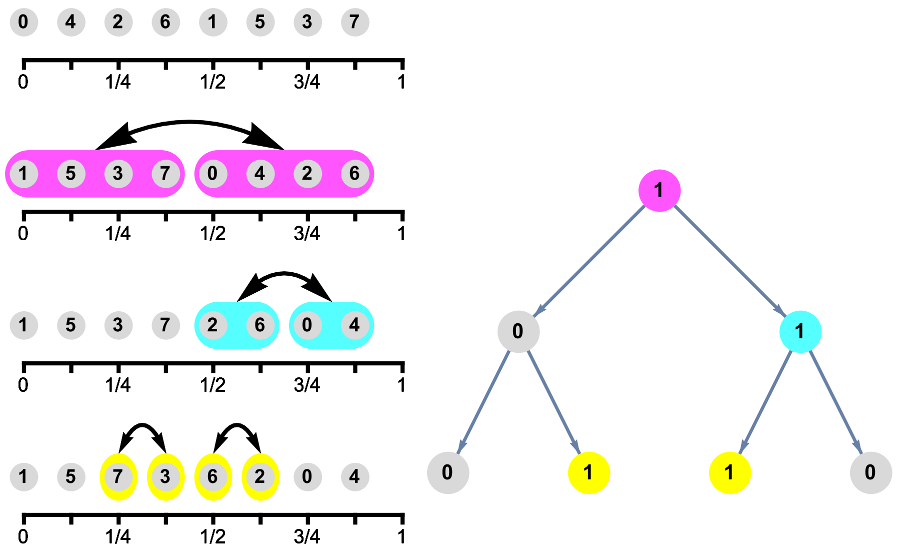
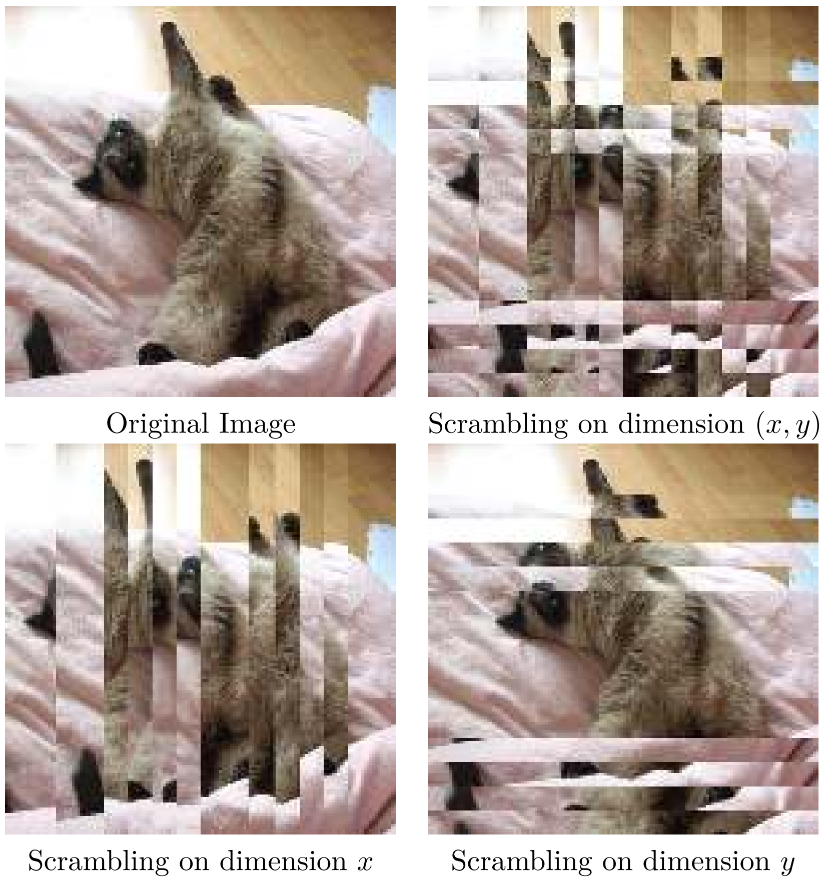
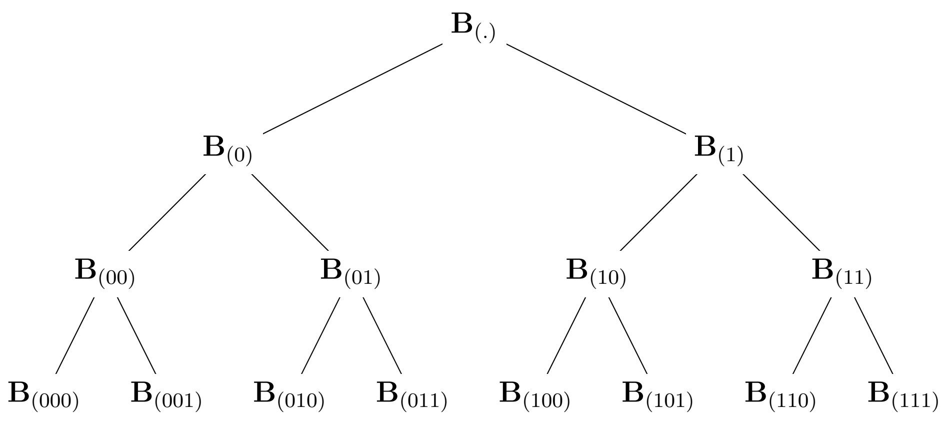


Fig. 4.1.: Owen's scrambling algorithm in 1-D [Owe95]



Original Image
Scrambling on dimension (x, y)
Scrambling on dimension x
Scrambling on dimension y

Fig. 4.2.: Geometric illustration of Owen's scrambling algorithm in 2-D [Owe95]



This boolean tree is then applied on each 1-D sample of the set. Let us consider a 1-D sample with coordinate $x \in [0,1)$ defined in base 2 with digits a_i (i.e. $x := \sum_{i=1}^{\infty} a_i 2^{-i}$). Owen's scrambling creates a new 1-D point $\pi(x)$ from a tree of depth d. The initial sample x is defined in base 2 as $x = (.a_0 a_1 \cdots a_N)$ Each digit a_k of this point is xored with the flags in the permutation tree as follows, leading to a new set of b_k digits:

$$\begin{aligned}
b_0 &:= \mathbf{B}_{(.)} \oplus a_0 \\
b_1 &:= \mathbf{B}_{(a_0)} \oplus a_1 \\
b_2 &:= \mathbf{B}_{(a_0 a_1)} \oplus a_2 \\
&\vdots \\
b_{d-1} &:= \mathbf{B}_{(a_0 a_1 \cdots a_{d-2})} \oplus a_{d-1}
\end{aligned}$$

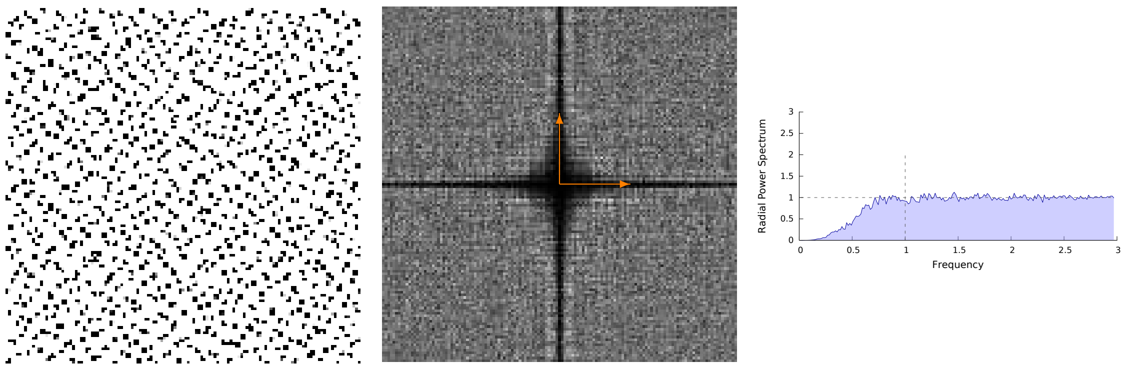
where the $\oplus$ operator performs the binary *xor* between two boolean values. This leads to a final scrambled point

$$\pi(x) := (.b_0 b_1 \cdots b_{d-1} a_d a_{d+1} \cdots a_N) \tag{4.4}$$

(note that if $N < d$, we have $\forall k, k > N, a_k = 0$).

This means that Owen's scrambling goes through all digits and swaps the digit a_i if its associated flag is set to 1 in the tree at depth i. Geometrically, swapping a digit a_i swaps all samples with the same $a_0 ... a_i$ binary prefix (thus all the samples that geometrically belong to the same portion of space). Owen's scrambling in 1-D therefore implies successive permutations between intervals of points of size 2^{-i+1}. Please refer to [Grü+12] for efficient implementations of this algorithm.

Note that this scrambling, when applied on a digital net, has the interesting property of both preserving the Low Discrepancy property of this net (meaning that it preserves its equidistribution), and of being able in some cases to remove the peaks in its Fourier Spectrum (Fig. 4.3 on the next page). This removal of the peaks happens with a probability one when the permutation trees, that control if two subintervals of space are swapped, are filled at random with a high enough depth d. This leads to a random exchange of samples coordinates, which breaks the original structure or the net.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 4.3.: Owen's scrambling [Owe95] applied on a Sobol sequence [Sob67]. The measurements were done on a single realisation of this scrambling.

4.2 Owen's scrambling, from a random set to a random sequence

In the previous section, we presented the original Owen's scrambling algorithm. This algorithm manages sets of samples. As the depth d of the tree is fixed, this algorithm implicitly assumes that we have only 2^d samples, and therefore, it is not sequential. If one tries to iteratively increase the size of this tree with randomly chosen flags, maintaining the consistency between samples would require re-evaluating this tree for all samples, which might move previously scrambled samples, thus breaking the sequentiality.

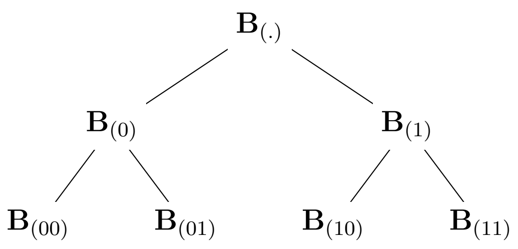
Our goal is to be able to iteratively increase this tree without having to re-evaluate this tree for previously existing samples, while still preserving the consistency of the whole sequence. To do so, we rely on the two following observations:

- Each boolean tree of depth d has 2^{d-1} different branches, identified as the binary vector of the id of their leaf node $(a_0, a_1, \ldots, a_{d-1})$. Each branch applies to the pair of samples whose $d-1$ first digits are $(a_0, a_1, \ldots, a_{d-2})$.

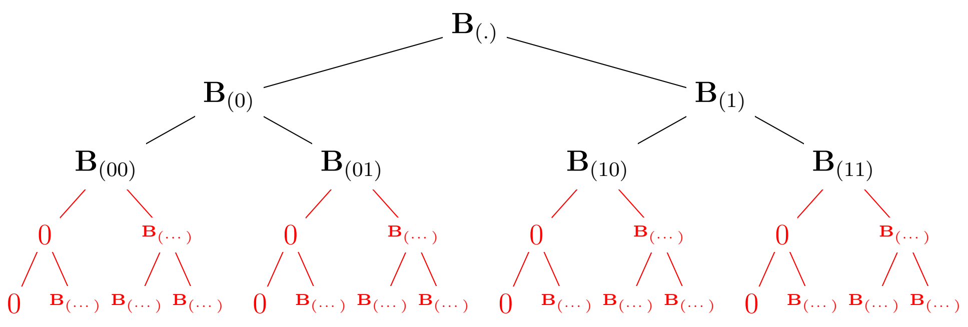
- If all the flags along the branch $(a_0, a_1, \ldots, a_{d-1})$ are 0 values, the samples with coordinate $(.a_0a_1 \ldots a_{d-2}0)_2$ and $(.a_0a_1 \ldots a_{d-2}1)_2$ will not be moved by scrambling.

Following those observations, it appears that we can iteratively increase the size of a permutation tree, creating a new tree of depth $d + m$, without affecting previously existing samples. To do so, we need to figure out which branches apply to those samples, and to fill the extensions of those branches with flags at 0.

For example, the following tree of depth 3:



can be extended, while preserving sequentiality, to the following tree of depth 5:



Depending on the size of m, only $m \cdot 2^d$ flags are required to be 0, leaving the remaining $2^{d+m} - m \cdot 2^d$ flags to be randomly chosen. Note the trade-off regarding the m value, if m is too small, the point set will not be properly anti-aliased, (Fig. 4.4 on the facing page) if m is too big, you may end up will evaluating a very deep tree for only a small set of samples.

With this contribution, one can use Owen's scrambling in a high dimensional sequential context. However, it still does not generate sets that are Blue Noise. A further extension of Owen's scrambling is presented in Section 4.4 on page 84 that allows generating small patterns with Blue Noise spectrum and Low discrepancy. Before that, we present in Section 4.3 another contribution that aims at improving the discrepancy of a stochastic pattern.

4.3 From an Anti-aliased to a Low Discrepancy set

In this section, we present a simple method that can be used to reduce the discrepancy of a pre-existing set. In its current state, our method takes as input a given set (for example from the BNOT sampler [DG+12]), and optimizes its generalized l_2 star discrepancy. Where the l_∞ star discrepancy keeps the maximal observed difference, the generalized l_2 star discrepancy averages all the squared differences observed from the area of anchored boxes and samples:

$$(\mathcal{D}_2(P_n))^2 := \left(\frac{4}{3}\right)^s - \frac{2}{n}\sum_{i=1}^{n}\prod_{j=1}^{s}\frac{3-\left(x_j^{(i)}\right)^2}{2} + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\prod_{j=1}^{s}\left[2-\max\left(x_j^{(i)}, x_j^{(i')}\right)\right]. \qquad (4.5)$$

The main idea, in this section, is to solve the following optimization problem:

Find point set $P^*(n)$ such that $\mathcal{D}_2(P_n^*)$ is minimal.

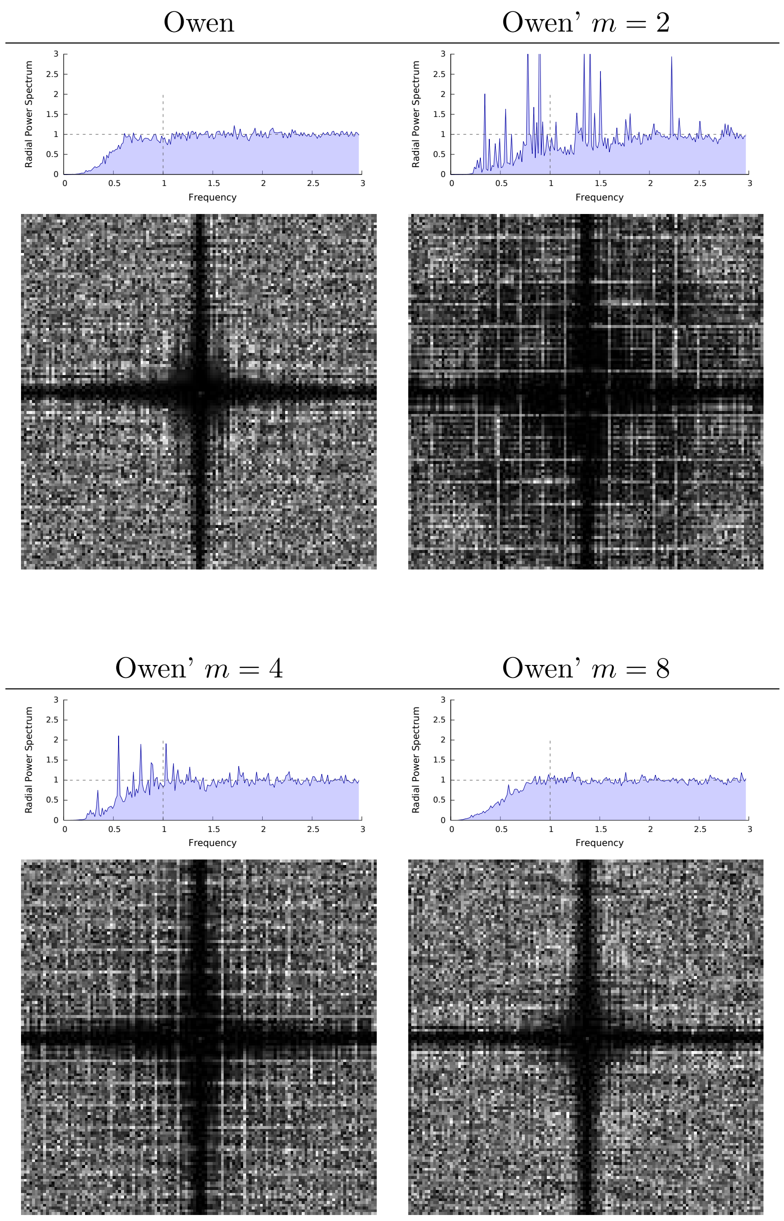


Fig. 4.4.: Spectral comparison between Owen's scrambling, and our adaptation; both applied on the same Sobol Sequence. We can see that for small m values, artefacts still appear but as m increases, there is no longer any visible difference between our set and the original Owen.

We follow a variational approach expressing the variation of the discrepancy as we slightly move a given point (*i.e.* its derivative with respect to sample positions). First, we observe that max terms in Eq. (4.5) cannot be differentiated. We first rewrite (4.5) using a smooth approximation of the max (sometimes referred as *softmax*):

$$\mathcal{M}_\alpha(u,v) := \frac{ue^{\alpha u} + ve^{\alpha v}}{e^{\alpha u} + e^{\alpha v}} \,. \tag{4.6}$$

This smooth approximation tends to the maximum value between u and v as α increases. A key property is that $\mathcal{M}$ can be differentiated with respect to u (similarly to v):

$$\frac{\partial \mathcal{M}_\alpha}{\partial u}(u,v) = \frac{e^{\alpha(u+v)}(\alpha(u-v)+1) + e^{2\alpha u}}{(e^{\alpha u} + e^{\alpha v})^2} \,. \tag{4.7}$$

Replacing the max by $\mathcal{M}_\alpha$ in (4.5) leads to a differentiable definition of the (square of the) generalized l_2 discrepancy. More precisely, for a given 2-D sample $\mathbf{x}^{(i)} := (x^i, y^i)$, we obtain the explicit gradients of discrepancy at this given point along its components:

$$\begin{aligned}\frac{\partial(\mathcal{D}_2(P_n))^2}{\partial x^i} =& -2 + y^i + \\ & 2\sum_{j=1}^{n}\left(2\left(-\frac{e^{\alpha x^i} + e^{\alpha x^i}\alpha x^i}{e^{\alpha x^i} + e^{\alpha x^j}} + \frac{\alpha e^{\alpha x^i}(e^{\alpha x^i}x^i e^{\alpha x^j}x^j)}{(e^{\alpha x^i} + e^{\alpha x^j})^2}\right)\left(2 - \frac{e^{\alpha y^i}y^i + e^{\alpha y^j}y^j}{e^{\alpha y^i} + e^{\alpha y^j}}\right)\right) ,\end{aligned} \tag{4.8}$$

$$\begin{aligned}\frac{\partial(\mathcal{D}_2(P_n))^2}{\partial y^i} =& -2 + x^i + \\ & 2\sum_{j=1}^{n}\left(2\left(-\frac{e^{\alpha y^i} + e^{\alpha y^i}\alpha y^i}{e^{\alpha y^i} + e^{\alpha y^j}} + \frac{\alpha e^{\alpha y^i}(e^{\alpha y^i}y^i e^{\alpha y^j}y^j)}{(e^{\alpha y^i} + e^{\alpha y^j})^2}\right)\left(2 - \frac{e^{\alpha x^i}x^i + e^{\alpha x^j}x^j}{e^{\alpha x^i} + e^{\alpha x^j}}\right)\right) .\end{aligned} \tag{4.9}$$

We can now combine all independent gradients into a single vector
$\nabla(\mathcal{D}_2(P_n))^2 := \left(\left(\frac{\partial(\mathcal{D}_2(P_n))^2}{\partial x^i}, \frac{\partial(\mathcal{D}_2(P_n))^2}{\partial y^i}\right)\right)_{i=0\ldots n-1}$ for all points i of P_n to define . For convex energy function, optimal sample positions P_n^* would have been obtained solving $\nabla(\mathcal{D}_2(P_n))^2 = 0$. In our context, Eq.(4.5) is not convex and has many local minima. We minimize the non linear energy using a simple Gradient Descent approach as described in Algorithm 1: we move each point of a given pattern to the opposite direction of its gradient by a factor (until the gradient norm is below a given threshold). Many improvements exist using for instance a line search to find a better λ at each step, or higher order schemes using Newton's or Quasi-Newton's approaches. We just describe here a proof of concept that already gives interesting results. Again, the fixed point of the algorithm may not be a global minimum but the improvement in terms of discrepancy is valuable while preserving the spectral properties of the starting point set.

To compute the gradient at a point $\mathbf{x}^{(i)}$ (line 4), we have experimented several strategies:

Algorithm 1: Simple gradient descent algorithm to minimize the generalized l_2 star discrepancy.

```
   Input: A 2-D point set $P_n$ to optimize and a threshold $\tau$.
   Output: A 2-D point set $P_n$ with a lower discrepancy.
1  repeat
2      $l_2norm \leftarrow 0$;
3      forall $\mathbf{x}^{(i)} \in P_n$ do
4          $\nabla \mathbf{x}^{(i)} \leftarrow \text{computeGradient}(\mathbf{x}^{(i)})$;
5          $\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i)} - \lambda \cdot \nabla \mathbf{x}^{(i)}$;
6          $l_2norm \leftarrow l_2norm + \|\nabla \mathbf{x}^{(i)}\|^2$;
7      end
8  until $l_2norm < \tau$;
9  return $P_n$;
```

Explicit smooth gradients: use explicit gradients using Eq. (4.8) and (4.9). As discussed earlier, we need the smoothing factor α to be large to have a robust approximation of the max operator. The main issue is that as the number of points increases, we need to also increase α to refine the approximation. For large point sets ($n > 500$), we observe numerical inconsistencies leading to incorrect results or non-convergence of Algorithm 1.

Auto-differentiation of the non-smooth generalized discrepancy: another solution consist in considering the non-smooth original definition of the Generalized discrepancy (hence with the max). In that case, the gradient is obtained using automatic differentiation of the function computing the energy. The auto-differentiation tool ([Dcp]) will *differentiate* the singularities around the "if" statements involved in the energy expressions for the max terms, leading to an expression of the *code-based* gradient that we can use in the Gradient Descent[1].

Results given below have been obtained using the second approach.

Figure 4.5 on the next page shows how changing the position of a single sample affects the overall discrepancy. The discrepancy of the point set on the left is computed for various displacements of the red sample, and the results are plotted on the right. We can see that the lowest discrepancy is achieved as the red sample moves towards the origin, as intuitively expected when seeing the point set.

Figure 4.6 on page 85 shows the Fourier spectrum of a Blue Noise set obtained from the BNOT algorithm [DG+12], and the Fourier spectrum of this same set after optimization. We can note that a black cross appears in the middle of the spectrum. This cross appears as, to reduce the discrepancy, the samples aligned on either axis were pushed away from each other by our optimization.

[1]Auto-differentiation tools use operators overloading to obtain the derivative $h'(y)$ at a point y of a code expression $h := f(g(k(x)))$. Basically, the tool uses an automatic chain rule to compute the derivative at y back-propagating the values through the source code graph (https://en.wikipedia.org/wiki/Automatic_differentiation)

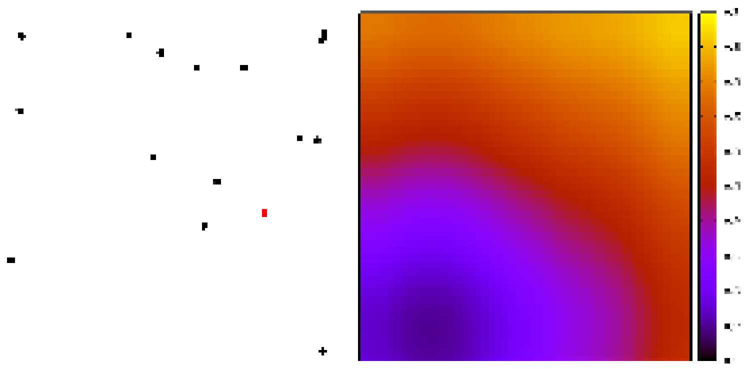


Fig. 4.5.: The measured discrepancy when changing the position of the red sample.

Figure 4.7 on page 86 presents the discrepancy of the original BNOT sampler, compared with its discrepancy after minimizing its generalized l_2 discrepancy. This Figure presents two graphs, showing the evolution of the l_∞ star discrepancy, and the evolution of the generalized l_2 star discrepancy.

For now, this method is still limited but we hope that, by combining it with Oztirelli's method to fit a point set to a given PCF [OG12], and by porting on the GPU, we could generate very quickly stochastic low discrepancy sets with a given Fourier spectrum, even for a high number of samples. We further could add an adaptivity criterion to our sets, using local discrepancies measures and PCF. But this is all for future work.

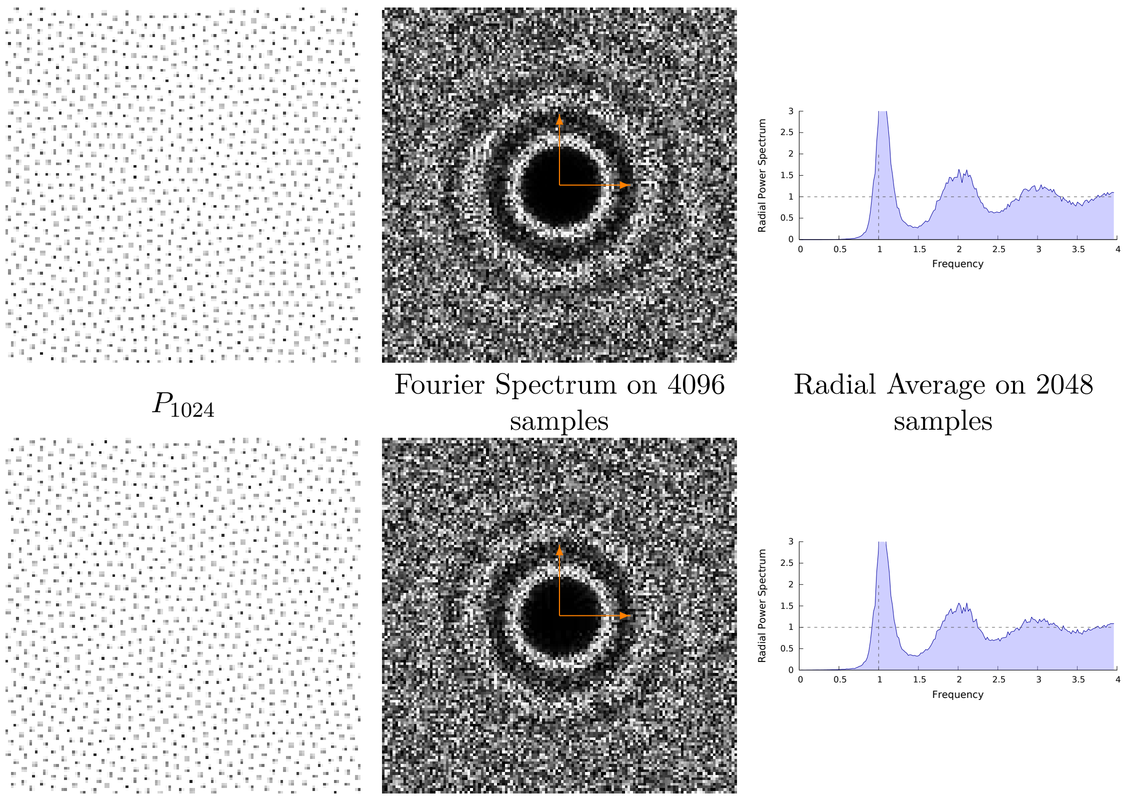
4.4 From a Low Discrepancy to an Anti-aliased set

The methods presented in this second section aim at scrambling/generating Low discrepancy sets to control their Fourier spectrum. We start by presenting an existing method, that optimizes generative matrices from exhaustive search (Section 4.4.1). We then present our new contributions, based on genetic algorithms, to generate Blue Noise sets from Owen's scrambling on the Sobol sequence (Section 4.4.2 on page 87).

4.4.1 Generative matrices

To generate Low discrepancy sets, a solution is to generate digital nets in base b. To do so, one needs s b-ary matrices (one for each dimensions), constrained by the Theorem 1 on page 63. Then, by computing the matrix vector product of the b vector of the index of samples and the appropriate matrix for each dimension, one generates a set of s-D samples that form a digital set.

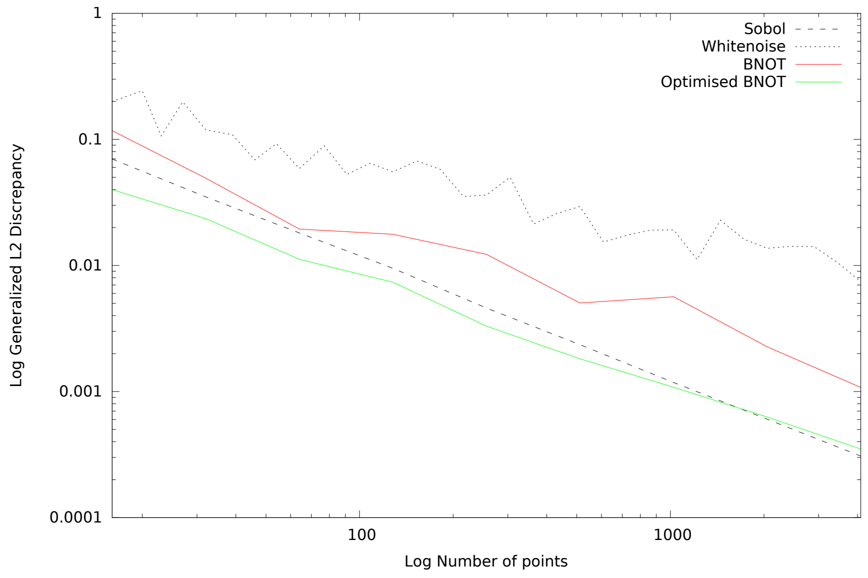
In [Grü+08], authors aim at generating 2-D digital sets while maximizing the minimal distance between samples. To do so, they start by exploiting a previously mentioned property of s-D digital sequences; they can be turned into $(s+1)$-D digital sets by setting one of their generating matrices as a mirror identity matrix. In their paper, the authors want to generate $(0, k, 2)$-nets



P_{1024} Fourier Spectrum on 4096 samples Radial Average on 2048 samples

P_{1024} Fourier Spectrum on 4096 samples Radial Average on 2048 samples

Fig. 4.6.: Top: The original BNOT set, Bottom: Results of our discrepancy optimization. The black cross in the spectrum is characteristic in sets with lower discrepancy.



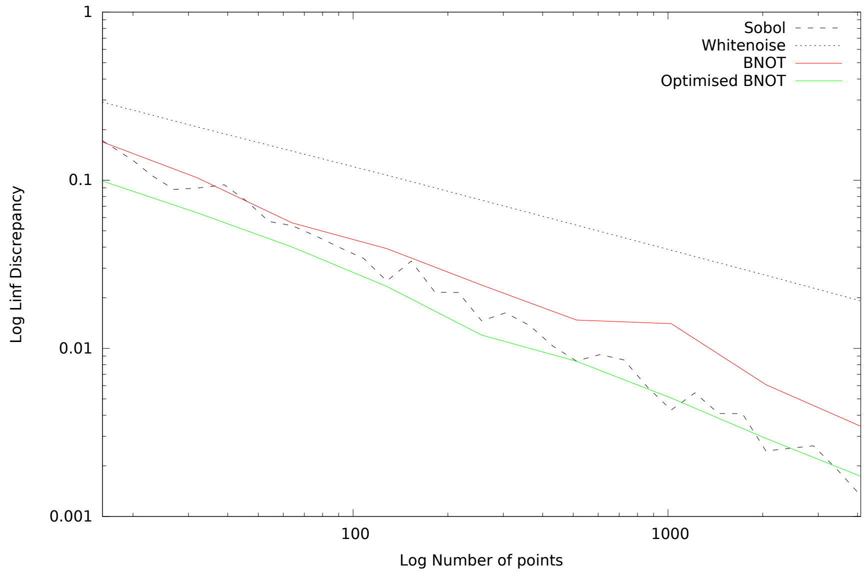
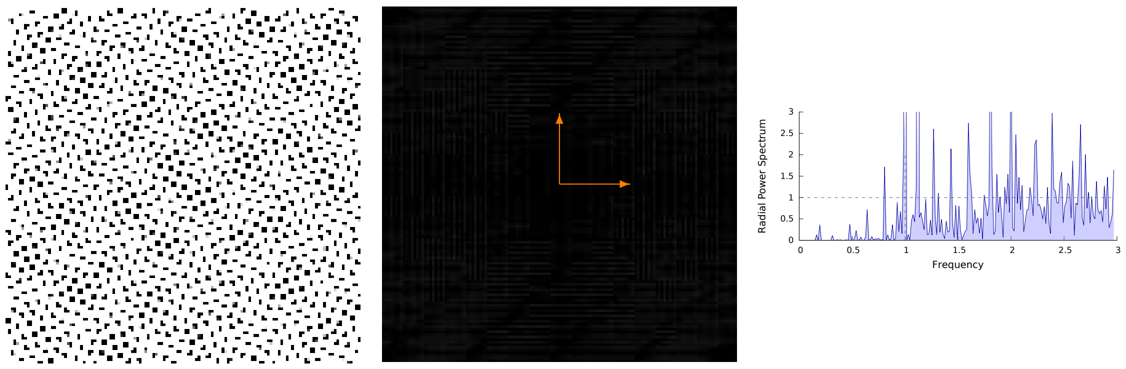


Fig. 4.7.: Evolution of the l_∞ and l_2 star discrepancy of the BNOT sampler, and of our minimization.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 4.8.: $(0, k, 2)$-nets with maximized minimum distance [Grü+08].

maximizing the minimal distance between pairs of samples of the set. They do this by setting the matrix C_1 as a mirror of the identity matrix:

$$C_1 := \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ & & \iddots & & \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix}. \tag{4.10}$$

Then, they simply perform an exhaustive search on all the possible generating matrices to find C_0, and keep the matrices that generated sets with the maximal minimal distance. This kind of exhaustive search limits them to 2-D sets of small size (up to 1024 samples), therefore, they suggest that those matrices can be increased by padding them with the identity matrix.

Even though they did manage to have a higher minimal distance between samples, we cannot say that their spectrum is Blue Noise or aliasing free (Fig. 4.8). Their issue is that, to make their exhaustive search feasible, they enforce the C_1 matrix, and, as that they generate digital nets, the matrix C_0 is highly constrained. Thus, they do not have a lot of freedom to generate their sets of samples.

We tried to alleviate some of those issues in Section 4.4.2 by using genetic algorithms and Owen's scrambling to perform our searches.

4.4.2 Genetic algorithms

Our goal is to optimize a Low Discrepancy point set. To do so, one way is to rely on a definition of this point set using a numerical structure, that can only take a finite number of values.

In the case of [Grü+08], their point sets were defined by a binary matrix. In our contribution, we chose to define our sets as a set of Owen's permutation trees, along with the initial Sobol set to scramble (each distinct initial set requires distinct permutation trees).

Note that for a s-D point set of size n, we have 2^{sn} possibilities for those permutation trees. We can thus generate optimal patterns for 2-D sets with a small number n of samples through an exhaustive search on all the possible Owen permutation trees. But, for merely $n = 16$ and $s = 2$, we need to test 2^{32} possibilities.

This exhaustive search is therefore impossible for higher dimensions or higher n values. Fortunately, the issue of finding the optimal solution in a space way too big to be entirely covered is well known in many domains, there are many algorithms that can optimize our search. We chose here to use genetic algorithms.

Our method initializes a population of permutation trees with random values. Each of those trees, once applied on the input point set leads to a scrambled set, that we will denote as an *individual*. Then, we sort those individuals using either the distance between their PCF and a target PCF, or simply by computing the minimal distance between samples (for Blue Noise). Once those individuals are sorted, we retrieve the best ones, that will form a set denoted the *elite* set. We then create new individuals by either crossing our elite individuals, or generating new random values. In our experiments, we crossed individuals by exchanging chunks of their permutation trees. We then keep iterating like this a finite number of times, or until we reached a good enough individual. This algorithm is given as Algorithm 2 on the next page.

Note that our method can become very expensive as the size of the population and the number iterations grows. This expensiveness of the method is partly alleviated as trees can be randomized. To randomized an optimized tree of depth d, simply keep the first $d - 1$ levels as they are and fill the d^{th} level with random values. The movement on samples will be minimal and will not affect much the quality of the final spectrum.

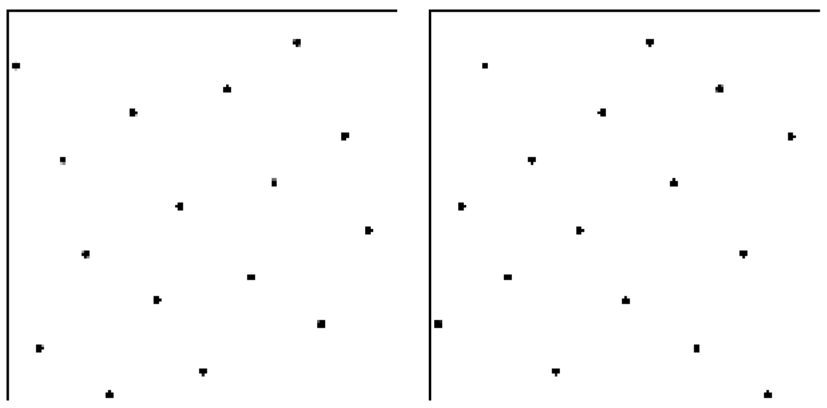
For small pointsets, we compare the results of this algorithm with the results from an exhaustive search in Figure 4.9 on page 90. Figure 4.10 shows how our optimization evolves with the number of iterations. Note that on both those figures, only small point sets are presented. As it is impossible for such small sets to be characterized using a PCF and a Fourier spectrum, we used the minimal distance between samples, denoted D. The higher this distance, the better the point set.

The results of our genetic search for sets with higher number of samples are given in Figure (Fig. 4.11 on page 90). This figure shows that for larger point sets, we no longer observe any improvement in the spectrum. This is due to the number of solutions that grows too much for a naive algorithm such as ours to find anything worthwhile.

Both methods, our genetic algorithms and the matrices from [Grü+08] have the same drawbacks. They both present results that are an improvement (though but not as good as we would like them to be), but they are so computationally expensive that they are impossible to use in

Algorithm 2: Our genetic algorithm to optimize Owen's permutation trees.

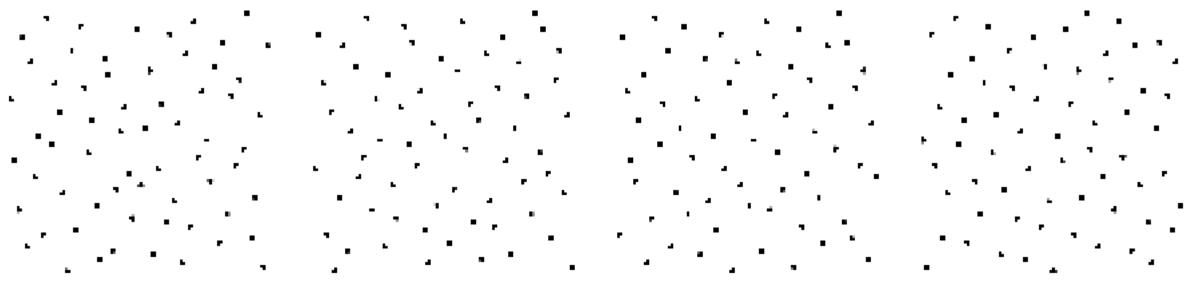
```
Input: A $s$-D point set to optimize
Output: A set of Owen's permutation trees optimizing the input point set
1  sizePopulation $\leftarrow$ 100;
2  nbIterations $\leftarrow$ 1000;
   // Initialize Population.
3  population $\leftarrow$ {};
4  repeat
5      individual.trees $\leftarrow$ $newRandomTrees()$;
6      individual.score $\leftarrow$ 0;
7      population $\leftarrow$ population $\cup$ individual;
8  until $i < sizePopulation$;
   // Iteratively sort and crosses individuals.
9  repeat
10     forall $individual \in population$ do
11         individual.score $\leftarrow$ $evaluateIndividual()$;
12     end
       // The higher the score the better the individual.
13     $sortPopulation()$;
       // Creating a set with all the best individuals.
14     elitePopulation $\leftarrow$ {};
15     repeat
16         elitePopulation $\leftarrow$ elitePopulation $\cup$ population[i];
17     until $i < \frac{sizePopulation}{10}$;
       // Using the elite individuals to create a new set of individuals.
18     forall $individual \in population$ do
           // We either mutate randomly or acquire genes from a member of
              elite individuals.
19         if $random() \mod 2 = 0$ then
20             crossIndividual.trees $\leftarrow$ $newRandomTrees()$;
21         else
22             crossIndividual $\leftarrow$ $getRandomEliteIndividual()$;
23         end
           // Replace part of the individual flags with the flags from
              crossIndividual
24         $cross$(individual,crossIndividual);
25     end
26 until $i < nbIterations$;
27 return population[0].trees;
```



Iteration 1000
$D \approx 0.225$

Exhaustive
$D \approx 0.225$

Fig. 4.9.: Comparison of the results our genetic algorithm with a point set of 16 samples and an initial population of 200 random individuals, with the results from the exhaustive search.



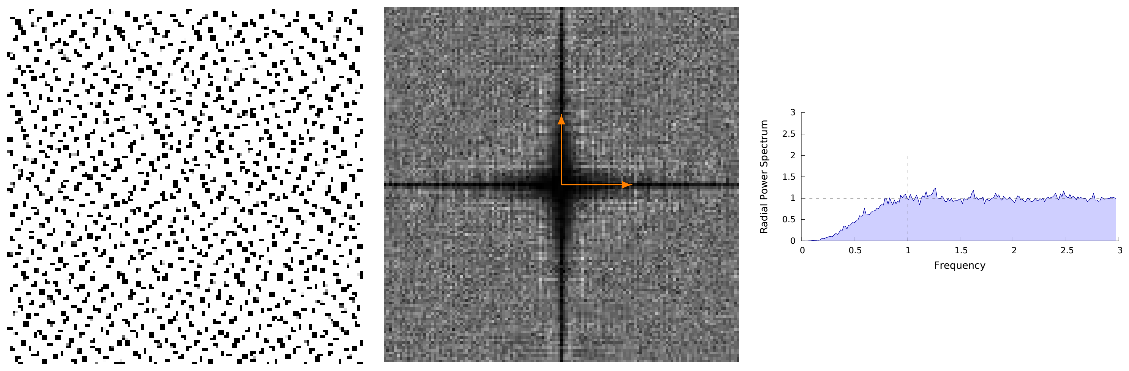
Iteration 1
$D \approx 0.0563$

Iteration 10
$D \approx 0.0662$

Iteration 100
$D \approx 0.0698$

Iteration 1000
$D \approx 0.0911$

Fig. 4.10.: Evolution of our genetic algorithm with a point set of 64 samples and an initial population of 200 random individuals.



P_{1024}

Fourier Spectrum on 8192 samples

Radial Average on 8192 samples

Fig. 4.11.: Characteristics of our genetic algorithm.

practice. In the following chapters, Chapter 5 and Chapter 6, we will present more advanced methods of the same category (trying to optimize a Low Discrepancy set to generate a Blue Noise Low Discrepancy set), that generate efficiently sets or sequences with a spectrum of higher quality.

4.5 Conclusion

All the methods presented in this section aim at solving the same issue, combining an optimal integration error (as the one of low discrepancy samplers) with the anti aliased properties of stochastic (and most of all Blue Noise) Fourier spectrum.

The main ideas exposed in Section 4.4 on page 84 follow the same simple idea; finding a numerical representations for low discrepancy sets or sequences and optimizing this value, either from optimization algorithms or simple brute force searching. Those methods give very good results but only for very limited number of samples and dimensions. We definitely believe that with more advanced optimization algorithms and the use of embarrassingly parallel components such as GPUs, we could get (in the case of Owen's scrambling) a very good optimization tree that could have practical applications. However, such a method is still to be developed.

Therefore, the following Chapters present instead two very specific methods to generate first, Blue Noise Low discrepancy 2-D sets (Chapter 5 on page 93) and then, a s-D sequential sampler with Low discrepancy Blue Noise 2-D projections (Chapter 6 on page 107).

5 Low Discrepancy Blue Noise point sets

In Chapter 4 on page 74, we saw how one can transform a Low Discrepancy set to improve its Fourier Spectrum. However, most of previous methods required exhaustive search or complex optimization schemes and are thus difficult to use in practice. Fastest algorithms, such as Owen's scrambling do improve upon the spectrum but are still far away from the ideal Blue Noise spectrum.

In this Chapter, we present a more advanced method that efficiently generates Low Discrepancy and Blue Noise 2-D point sets [Ahm+16b] (s-D sequences will be discussed in Chapter 6 on page 107). We achieved this by scrambling an input template Low Discrepancy point set to match its spectrum to the spectrum of a target stratified point set. We use Van der Corput sequence to create a stratified low discrepancy initial set and then, through coordinate wise rearrangement, we scramble it without affecting the discrepancy of the original set. In the following, Section 5.1 presents our input Low discrepancy construction. Then, Section 5.2 on page 98 presents our scrambling method.

5.1 Initial Stratified Low Discrepancy set

Our method starts with a template low discrepancy (LD) set, that will then be optimized to acquire a desired spectral profile (BNOT [DG+12], Step [Hec+13], etc). There are many LD constructions we could have chosen from as our input LD set (Sobol, Halton, Hammersley, Rank-1, etc.). The issue is that with such sets, it is difficult to identify the neighbourhoods of the points. As the spectral properties are sensitive to local neighbourhoods [WW11], we need to devise a new LD construction that addresses this issue, based on stratification. In the following, we start by formally defining a 2-D stratified point set. Then, we list the conditions to create a 2-D low discrepancy stratified point set. Finally, we present our choice as the template LD construction.

5.1.1 Stratification

In this section, we define formally a 2-D stratified point set. Suppose that we have a stratified point set P_n, with $n := k^2$ samples. We denote such a set as

$$P_n := \Big\{ (X + a_{X,Y}, Y + b_{X,Y}) : X, Y \in \{0 \ldots k-1\}; \\ a_{X,Y}, b_{X,Y} \in [0,1) \Big\}. \quad (5.1)$$

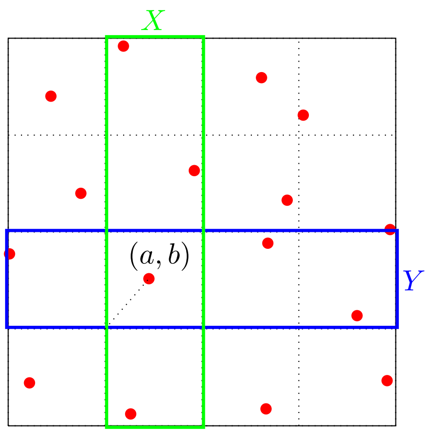


Fig. 5.1.: Illustration of Equation 5.1 on the preceding page, representing a stratified point set.

The pair (X, Y) selects a specific stratum, and an associated pair (a, b) specifies the location of the sample point within that stratum (Fig. 5.1). Note, that a and b are not necessarily functions of X and Y; $a_{X,Y}$ and $b_{X,Y}$ only indicate that each (a, b) is associated to a specific stratum (X, Y).

We point out that following this definition, a point set can only contain $k^2, k \in \mathbb{N}$ samples. Furthermore, such definition generates samples in the domain $[0, k)^2$. To sample the unit domain $[0, 1)^2$, one can simply scale the point set by $\frac{1}{k}$.

5.1.2 LD stratified point set

Stratified point sets are not naturally low discrepancy. Therefore, we need to develop a new construction that will let us access samples from the coordinates of their stratum while being a low discrepancy set. To do so, we generate stratified point sets P_n where, for every column X, we set the offsets values from a 1-D LD sequence $\{a_{X,0}, a_{X,1} \dots, a_{X,k-1}\}$, and similarly for every row Y, with the 1-D sequence $\{b_{0,Y}, b_{1,Y}, \dots, b_{k-1,Y}\}$. We can prove that this new construction is low discrepancy.

Intuitively, we can observe that the difference between the area of an integer set of strata and the fractions of samples of P_n that are located into those stratas is 0. Therefore, the discrepancy of the stratified set only depends on the discrepancy of the sequences used as offset in the rows and columns.

More formally, we denote by $\mathbf{a} := \{a_i\}$ a 1-D sequence of points in $[0, 1)$, $\mathbf{a}(k)$ denoting the set of the first k elements of the sequence. We consider a collection of k such sequences $\mathbf{a}^{(X)} := \left\{a_i^{(X)}\right\}$, indexed by an integer X; that is, $\mathbf{a}^{(X)}$ is the X^{th} sequence in the collection, while $a_i^{(X)}$ is the i^{th} entry in that sequence. We then consider a similar set up for $\mathbf{b}$, b, and Y ($\mathbf{b} := \{b_i\}$ and $\mathbf{b}(k)$) respectively, which gives us the following lemma:

Lemma 1 *If $n = k^2$ denotes the total number of points in a stratified set P_n, P_n being scaled to $[0,1)^s$, we have*

$$\mathcal{D}_*(P_n) \leq \frac{1}{k}\left(\max_{X,Y\in\{0\ldots k-1\}}\left(X\mathcal{D}_*(\mathbf{b}^{(X)}) + Y\mathcal{D}_*(\mathbf{a}^{(Y)})\right) + 1\right). \tag{5.2}$$

By proving Lemma 1, we will then prove that our construction generates low discrepancy sets, as long as a and b are Low discrepancy sequences.

Proof 1 *We assume that P_n is a 2-D point set defined over the domain $[0,1)^2$ (note that there is no loss of generality w.r.t. a point set defined over the domain $\mathbb{R}^2$). We remind that the l_∞ star discrepancy of P_n is defined as*

$$\mathcal{D}_*(P_n) := \sup_{\mathbf{v}\in[0,1)^2} |\lambda(\mathbf{v}) - \alpha(P_n, \mathbf{v})| \tag{5.3}$$

where $\lambda(\mathbf{v})$ is the Lebesgue measure of the interval $[0,\mathbf{v})^2$, and $\alpha(P_n,\mathbf{v})$ returns the fraction of points of P_n that belong to the interval $[0,\mathbf{v})^2$. We will denote $d(P_n, A)$ the 1-D discrepancy of P_n over a 1-D interval A.

Additivity of the Lebesgue measure (s-dimensional volume) and $\alpha(P_n,\mathbf{v})$ leads to the following property: Let A and B be 1-D disjoint subsets of $[0,1)$, then

$$d(P_n, A\cup B) \leq d(P_n, A) + d(P_n, B). \tag{5.4}$$

$\mathbf{a} := \{a_i\}$ *is a one-dimensional sequence of points in $[0,1)$. $\mathbf{a}(k)$ denotes the set of the first k elements of the sequence ($\mathbf{b} := \{b_i\}$ and $\mathbf{b}(k)$, respectively). We consider a collection of c such sequences $\mathbf{a}^{(X)} := \left\{a_i^{(X)}\right\}$, indexed by an integer X; that is, $\mathbf{a}^{(X)}$ is the Xth sequence in the collection, while $a_i^{(X)}$ is the i-th entry in that sequence. We consider a similar setup for $\mathbf{b}$, b, and Y, respectively.*

As P_n with $n = k^2$ is a 2-D scaled stratified point set, composed of m low discrepancy sequences it can be denoted as

$$P_n := \left\{\left(\frac{X + a_Y^{(X)}}{k}, \frac{Y + b_X^{(Y)}}{k}\right); X, Y \in \{0\ldots k-1\}\right\}. \tag{5.5}$$

In other words, the abscissa $\frac{X}{k}$ of a given point of the set is shifted by $\frac{a_Y^{(X)}}{k}$, whose numerator is the Yth element of the sequence $\mathbf{a}^{(X)}$. In the following, we simplify the notations of these sequences to $\mathbf{a}$ and $\mathbf{b}$ (and their points $\{a_i\}$ and $\{b_i\}$) in expressions whenever there is no ambiguity on X and Y values.

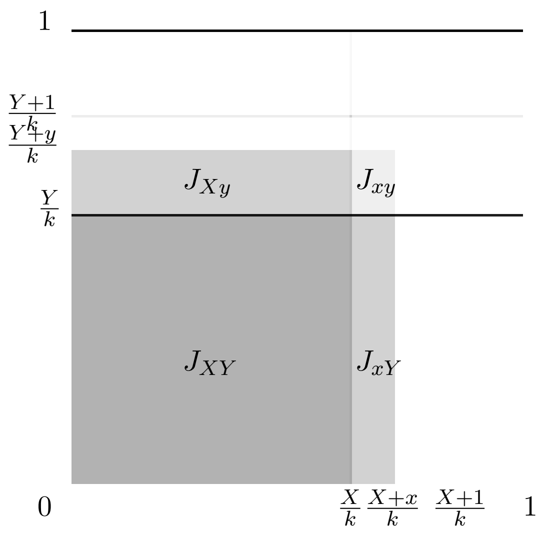


Fig. 5.2.: Notations for Lemma 1 on the previous page.

Let

$$J := [0, (X + x)/k] \times [0, (Y + y)/k] \; ; \quad x, y \in [0, 1) \, , \tag{5.6}$$

be an arbitrary interval in I^2. It can be partitioned into non-overlapping sub-intervals (see Figure B.1 on page 176):

$$J = J_{XY} + J_{Xx} + J_{xY} + J_{xy} \tag{5.7}$$

where

$$\begin{aligned}
J_{XY} &:= [0, X/k) \times [0, Y/k) \; , \\
J_{Xy} &:= [0, X/k) \times [Y/k, (Y + y)/k] \; , \\
J_{xY} &:= [X/n, (X + x)/k] \times [0, Y/k) \; , \\
J_{xy} &:= [X/n, (X + x)/k] \times [Y, (Y + y)/k] \; .
\end{aligned}$$

Thanks to the additive property of discrepancy, we have:

$$\begin{aligned}
d(P_n, J) \quad \leq \quad & d(P_n, J_{XY}) + d(P_n, J_{Xy}) + \\
& d(P_n, J_{xY}) + d(P_n, J_{xy}) \, .
\end{aligned} \tag{5.8}$$

Now

$$\begin{aligned}
d(P_n, J_{XY}) \quad &= \quad \left| \frac{\alpha(P_n, J_{XY})}{n} - \lambda_2(J_{XY}) \right| \\
&= \quad \left| \frac{X \cdot Y}{n} - \frac{X}{k} \cdot \frac{Y}{k} \right| = 0 \, .
\end{aligned} \tag{5.9}$$

For J_{Xy} we have

$$\left(\frac{X + a_Y}{k}, \frac{Y + b_X}{k} \right) \in J_{Xy} \implies b_X \leq y \, . \tag{5.10}$$

If we write J_y to denote the one-dimensional interval $[0,y]$, we observe that

$$\alpha(P_n, J_{Xy}) = \alpha\left(\mathbf{b}(X), J_y\right) \tag{5.11}$$

and

$$\lambda_2(J_{Xy}) = \frac{X}{k}\cdot\frac{y}{k} = \frac{X\lambda_1(J_y)}{n} \tag{5.12}$$

Therefore:

$$\begin{aligned} d(P_n, J_{Xy}) &= \left|\frac{\alpha(P_n, J_{Xy})}{n} - \lambda_2(J_{Xy})\right| \\ &= \left|\frac{\alpha(b(X), J_y)}{n} - \frac{X\cdot\lambda_1(J_y)}{n}\right| \\ &= \frac{X}{n}\left|\frac{A(J_y;\mathbf{b}(X))}{X} - \lambda_1(J_y)\right| \\ &= \frac{X}{n} d(\mathbf{b}(X), J_y) \le \frac{X}{n}\mathcal{D}_*\left(\mathbf{b}(X)\right)\ . \end{aligned} \tag{5.13}$$

Similarly

$$d(P_n, J_{xY}) \le \frac{Y}{n}\mathcal{D}_*(\mathbf{a}(Y))\,. \tag{5.14}$$

Finally, $\alpha(P_n, J_{xy})$ is either 0 or 1, and $\lambda_2(J_{xy})$ is at most $1/n$, therefore

$$d(P_n, J_{xy}) \le \frac{1}{n}\,. \tag{5.15}$$

Substituting Eqs. (5.9 on the facing page, 5.13, 5.14, and 5.15) into Eq. (5.8):

$$d(P_n, J) \le \frac{1}{n}\left(X\mathcal{D}_*(\mathbf{b}(X)) + Y\mathcal{D}_*(\mathbf{a}(Y)) + 1\right) \tag{5.16}$$

and Eq. (5.2) follows for the upper bound of the star discrepancy of the set P_n.

A regular grid uses a fixed value (typically 0 or $\frac{1}{2}$) for all the entries in $\mathbf{a}$ and $\mathbf{b}$, hence $\mathcal{D}_*(\mathbf{b}(X))$ and $\mathcal{D}_*(\mathbf{a}(Y))$ are $\mathcal{O}(1)$ and as such $\mathcal{D}_*(P_n)$ is $\mathcal{O}\left(\frac{\sqrt{n}}{n}\right)$.

If all of the $\mathbf{a}$'s and $\mathbf{b}$'s are low-discrepancy sequences, that is, $\mathcal{D}_*(\mathbf{b})$ is $\mathcal{O}(\log(X))$ and $\mathcal{D}_*(\mathbf{a})$ is $\mathcal{O}(\log(Y))$, both bounded by $\mathcal{O}(\log(\sqrt{n}))$, then the resulting point-set is a low-discrepancy point-set, with discrepancy in $\mathcal{O}\left(\frac{\log(n)}{n}\right)$. Note, however, that such a construction does not extend automatically to higher dimensions.

This lemma bounds the discrepancy of our construction and its corollary

Lemma 2 *If $\mathcal{D}_*(\mathbf{b}^{(X)})$ has a discrepancy in $O(\log(n)/n)$ and $\mathcal{D}_*(\mathbf{a}^{(Y)})$ also has a discrepancy in $O(\log(n)/n)$, then $\mathcal{D}_*(P_n)$ is in $O(\log(n)/n)$.*

allows us to call our construction a low discrepancy set. Note that our construction is only valid for 2-D point sets.

We can thus construct 2-D low discrepancy stratified point set by picking any 1-D low discrepancy sequence S, and use it to supply the horizontal offsets along the columns:

$$a_{X,Y} := S_Y \,,$$

and the vertical offsets along the rows:

$$b_{X,Y} := S_X \,.$$

We denote $\mathcal{P}^S$ a point set obtained using this construction.

5.1.3 Template point set

In practice, we chose the Van der Corput sequence ϕ (see Section 3.3.2 on page 64). The resulting template LD point set is then defined by:

$$\mathcal{P}^\phi := \Big\{ (X + \phi(Y), Y + \phi(X)) : \; X, Y \in \{0 \ldots k-1\} \Big\} \,. \tag{5.17}$$

This set, denoted $\mathcal{P}^\phi$ (Fig. 5.3 on the next page), will serve as our template LD set. Thanks to the stratification, our construction is inherently indexed in two dimensions, using a pair of indices. This allows to align our LD set with any stratified set, on a one-to-one basis. This will be a key element to control the spectral properties of the set.

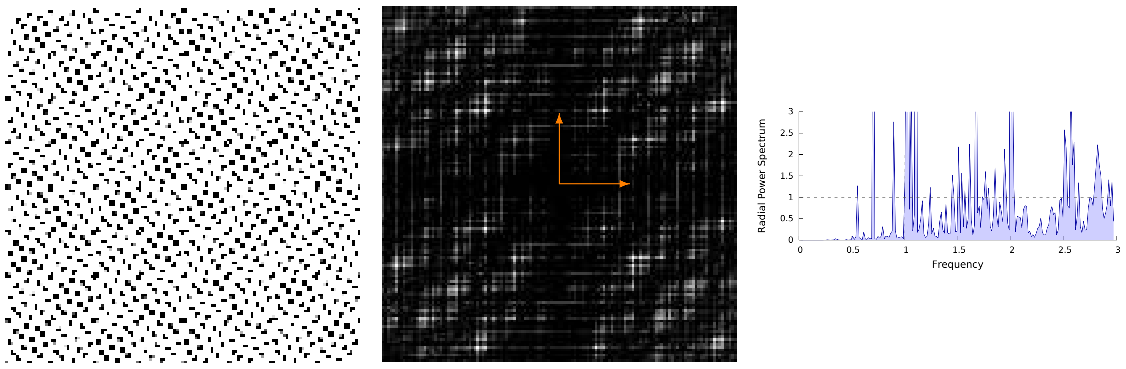
Following Lemma 2 on the preceding page, $\mathcal{P}^\phi$ is a low discrepancy point set with discrepancy $\mathcal{O}\left(\log(n)/n\right)$. It is worth noting that $\mathcal{P}^\phi$ coincides with the Hammersley construction when the generated number of points is a power of two; but not otherwise.

Once we have this initial set, we can optimize it by scrambling the samples to match a given spectral target. The critical part of this optimization is to do this without affecting the discrepancy of the complete set.

5.2 Scrambling

We can randomly scramble our template set by applying a discrepancy-preserving rearrangement of the sequences S used as offsets for each row and column. Here, the sequence S is the van der Corput sequence.

Unlike the global bit-wise scrambling techniques in common LD constructions, we randomly permute the entries of the constituent sequences (in rows and columns) locally within small



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 5.3.: Our template 2-D stratified low discrepancy point set.

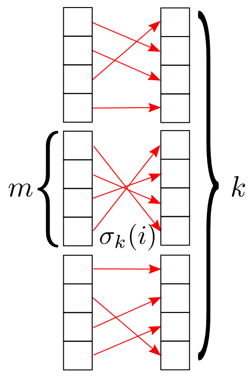


Fig. 5.4.: Random scrambling within columns, using chunks of size $m = 4$
.

chunks (Fig. 5.4). We choose a chunk size, m, and re-order every subsequent m entries of the input sequence S (we assume that m divides k):

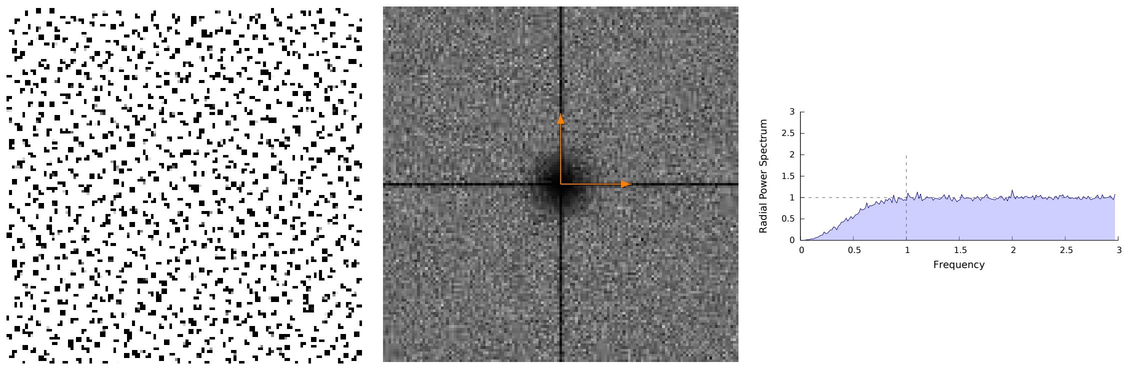
$$S'_{j\cdot m+i} = S_{j\cdot m+\sigma_j(i)}\,; \qquad j \in \{0\ldots k/m\}\,;\; i \in \{0\ldots m-1\}\,, \tag{5.18}$$

where each σ_j is a permutation over $\{0...m-1\}$. (Fig. 5.4)

We know that this permutation have little impact on the discrepancy. Any subset of S' differs from the corresponding subset of S in at most m points. According to [KN74, Theorem 4.1] (presented Section 4.1.2 on page 75), the difference in discrepancy of any subsets of length $n > m$ of the two sequences is then bounded by m/n. Since m is fixed, the difference

$$|\mathcal{D}_*(S') - \mathcal{D}_*(S)|$$

in discrepancy between the two sequences is $\mathcal{O}(1/n)$. Consequently, if S is an LD sequence, with $\mathcal{O}(\log(n)/n)$ discrepancy bound, then the shuffled sequence S' is also an LD sequence. The parameter m only alters the constant in the $\mathcal{O}(\cdot)$ notation, but we want to keep m small so that we stay close to the discrepancy of S.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 5.5.: A 2-D stratified low discrepancy point set obtained from random scrambling of $\mathcal{P}^\phi$.

Note that the statement of the mentioned theorem compares corresponding entries in two finite sequences. But since we are referring to subsets, these can be arranged in any order without altering their discrepancy. To give an example, the sets $(0, 0.25, 0.5, 0.75)$ and $(0, 0.5, 0.25, 0.75)$ have different discrepancies as sequences, but the same discrepancy as fixed sets.

To incorporate this 1-D chunk-wise permutation into $\mathcal{P}^\phi$, we apply two passes: first along the X (columns), then along the Y (rows). For each pass, we divide the X (resp. Y) into chuks of size m. Then, within each chunk, we reorder the offsets b (resp. a). Note that along the rows we reorder the vertical offsets and along the columns, we reorder the horizontal offsets. Algorithm 3 sums up this whole permutation.

At this point, the resulting point set (Fig. 5.5) closely resembles a scrambled LD set like Owen's [Owe95] (Fig. 4.3 on page 79). Our localized permutation, however, admits much more control over the placement of individual points than Owen's axis-wise scrambling, and it enables real optimization. Algorithm 3 sums up our random scrambling.

Algorithm 3: Stochastic permutation of $\mathcal{P}^S$.

```
1 for each column $X$ do
2     for each chunk $j$ do
3         Randomly permute the vector $\{a_{X,Y}\}_{Y=j \cdot m}^{j \cdot m+m-1}$;
4         Write the permuted vector back into the set;
5     end
6 end
7 Repeat with the rows by exchanging the roles of $a, X$ and $b, Y$, respectively;
```

5.2.1 Blue Noise scrambling

Instead of considering random permutations, we can construct the permutations such that the point set $\mathcal{P}^\phi$ is close to a given reference point set. Consider a stratified $t \times t$ point set (t=128 in our experiments), optimized to a specific blue noise profile (BNOT [DG+12], Step [Hec+13],

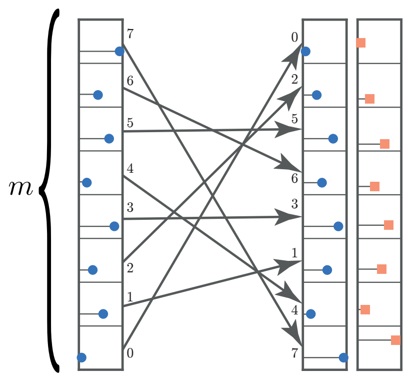


Fig. 5.6.: Optimizing the samples placement trough dimension-wise rearrangement, using $m = 8$
.

Poisson Disk, etc). We assume that t is a multiple of m, so that each row and column of the point set can be decomposed into chunks of size m.

The main idea is to start with a $t \times t$ chunk of the $\mathcal{P}^{\phi}$ template set, and to find the permutations for all 1-D chunks that would transform this point set so that it approximates the reference set. For each chunk of size m from the template LD set and the reference set, we rearrange the points of the template set so that the smallest offset a of the template aligns with the smallest offset a of the reference, the second smallest with the second smallest, and so on. This process thus minimizes the distance between the two sets. Figure 5.6 illustrates this for a single 1-D chunk.

The new sequence of indices, is then appropriately encoded and stored in the look up table. For a given point in a stratum (X_t, Y_t) in the $t \times t$ domain, the look-up table entry $LUT(X_t, Y_t)$ contains the offset $(X_m, Y_m) \in m \times m$, of the new sequence numbers within the $m \times m$ block. The offset of the sample in (X_t, Y_t) cannot however simply be replaced by the offset read from $LUT(X_t, Y_t)$. Doing this would replicate the $t \times t$ permuted point set, which would break the low discrepancy property since several points would project to the same 1-D point on either of the two axes. To overcome this issue, we define an indexing scheme related to the underlying 1-D LD sequence and the bit-reversal principle.

For a given abscissa X, we define $X_t := X \bmod t$, its position in a $t \times t$ domain, and $X_m := X \bmod m$, its position in a chunk of size m within the $t \times t$ domain (we assume similar definitions for Y_t and Y_m). The final offsets are given by:

$$a := \phi(Y - Y_m + L_Y) \tag{5.19}$$
$$b := \phi(X - X_m + L_X)\,, \tag{5.20}$$

where $(L_X, L_Y) := LUT(X_t, Y_t)$.

Eq. (5.19) and (5.20) perform both the initial offset construction, using the van der Corput sequence, and the optimized permutation within chunks of size m. More precisely, for a given X and Y, $\phi(X)$ is the vertical offset in the template LD point set (Eq. (5.17 on page 98)). The

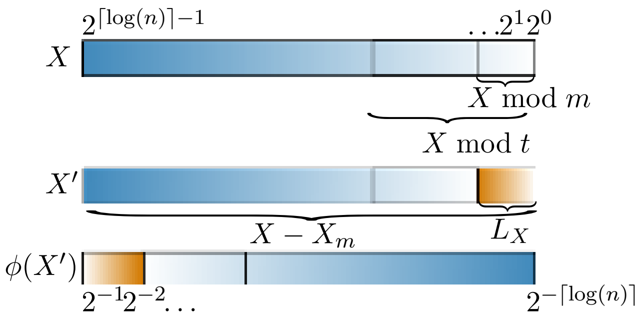


Fig. 5.7.: Bitwise representation of Eq. (5.19) and Eq. (5.20)

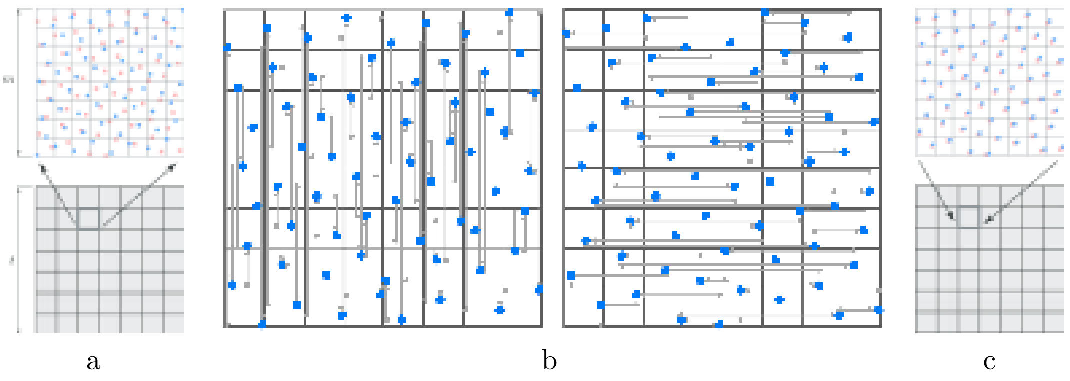
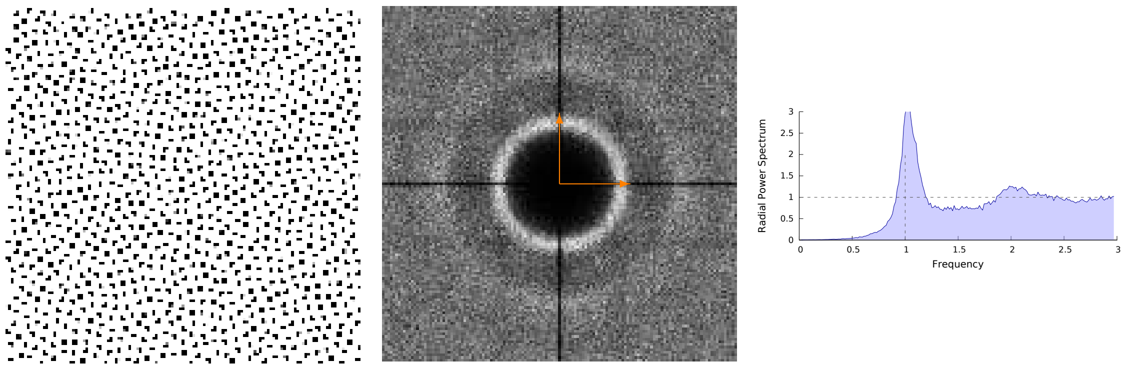


Fig. 5.8.: Reordering the template LD set (blue circles) to match a given reference (orange squares), without loosing its LD property.

optimized permutation is only performed within the size-m vertical and horizontal chunks the point (X, Y) belongs to. In a bitwise representation of X, the permutation only occurs in the least significant $\log_2(m)$ bits (Fig. 5.7).

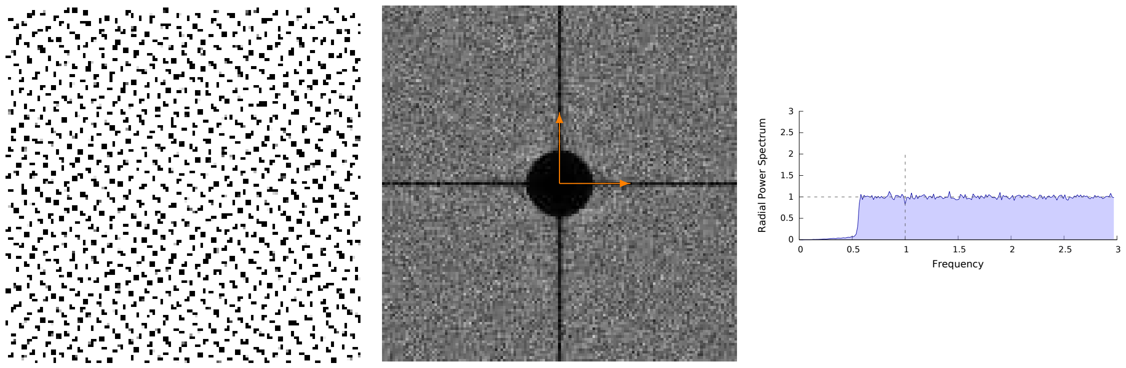
By performing those permutations independently along the x-axis, then along the y-axis, for each $m \times m$ block in the point-set, we can create a new pointset $P_{S'}$ that is still low discrepancy while presenting the spectral profile of the reference point set.Note that the two passes are completely independent, and can therefore be parallelized. Figure 5.8 illustrates this algorithm inside a single $m \times m$ block.

Note that the parameter m is a trade-off between the degradation of the discrepancy and the desired spectral profile. $m = 1$ reproduces $\mathcal{P}^S$; $m = t$ produces a merely Latinized copy of the reference set, with limited improvement of its discrepancy. From our experiments, $m = 16$ is a good compromise. The impact of m on the discrepancy was discussed earlier, following a theorem from [KN74] (see Section 4.1.2 on page 75. The size of the reference point set, t^2, is a trade-off between the memory footprint of the look-up table, and the quality of the spectral profile with respect to the reference set when $n > t^2$. Figure 5.11 illustrates how the spectrum evolves with different values for t and m.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 5.9.: Characteristics of the Low Discrepancy Blue Noise sampler with a BNOT [DG+12] target profile. $t = 128, m = 16$ [Ahm+16b].



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 5.10.: Characteristics of the Low Discrepancy Blue Noise sampler with a Step [Hec+13] target profile. $t = 128, m = 16$ [Ahm+16b].

Algorithm 4: Generating an optimized LD set from a reference stratified point set and our template LD point set.

```
1  Initialize the output set to $\mathcal{P}^S$ from Eq. (5.17);
2  for each column $X$ do
3      for each chunk $j$ do
4          Retrieve the vector $\mathbf{L_x} = \{a_{X,Y}\}_{Y=k\cdot m}^{j\cdot m+m-1}$ from the reference set;
5          Retrieve the corresponding vector, $\mathbf{Q_x}$, from the output set;
6          Rearrange $\mathbf{Q_x}$ so that the smallest entry aligns with the smallest entry in $\mathbf{L_x}$,
             the second smallest with the second smallest, and so on;
7          Write the permuted vector $\mathbf{Q_x}$ back to the output set;
8      end
9  end
10 Repeat with the rows by exchanging the roles of $a, X$ and $b, Y$, respectively;
```

Using previous notations, we define our final Low-Discrepancy Anti-Aliased sampler as follows:

$$\begin{aligned}\text{LDBN} := \Big\{&(X + \phi(Y - Y_m + L_Y),\\ &Y + \phi(X - X_m + L_X))\Big\}.\end{aligned} \tag{5.21}$$

Following Lemma 2 on page 97, we can affirm that the LDBN point set defined from van der Corput sequences in Eq.(5.21) has a discrepancy in $\mathcal{O}\left(\frac{\log(n)}{n}\right)$, and is thus a low discrepancy set.

We further empirically demonstrate in Chapter 7 on page 133 that when optimizing the look-up table with a reference point set with specific spectral properties (e.g, Step blue noise), LDBN preserves the low discrepancy of $\mathcal{P}^\phi$ while presenting a different spectral profile. The result of this sampler with different spectral profiles are presented in Figure 5.9 on the previous page, and in Figure 5.10 on the preceding page. Algorithm 4 sums up our optimized permutation.

5.3 Conclusion

On the practical side, LDBN is extremely computationally efficient, since it can be implemented using only bit manipulations and memory lookups. It is also considerably efficient in storage, since only integer permutations in $[0, m)$ have to be stored, rather than the actual real-valued offsets (a, b). If $m = 16$, then the storage space reduces to only one byte per entry.

However, despite being fast and generating Low Discrepancy sets with controllable spectrum, this method is still limited. It only operates on 2-D points, and only generates sets of samples. Following this, Chapter 6 on page 107 present a new construction to generate sequentially low discrepancy sets in higher dimensions.

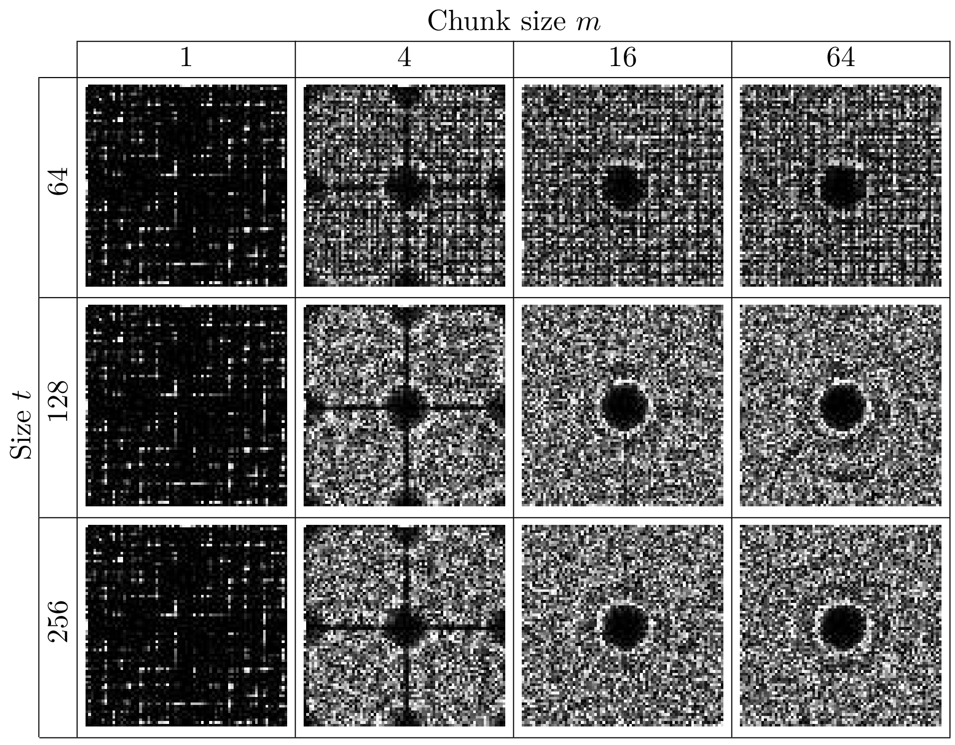


Fig. 5.11.: Fourier spectra from pointsets with 16000 samples obtained using our method with various values for the table size t and the chunk size m. As can be seen, the bigger the chunks m, the more Blue Noise the spectra, but if m increases too much, the Low Discrepancy property degrades. Also, as we increase t, we need less and less replication to generate our point sets which leads to cleaner spectra but at the cost of an increase in the memory requirements. In our tests, the parameters $m = 16$ and $t = 128$ are a good trade-off.

6 Multi-Dimensional Sequences with Low-Discrepancy Blue-Noise 2-D Projections

In this chapter, we detail a new 2-D scrambling that can be applied on projections of the Sobol sequence. This lets us generate s-dimensional sequences of samples with 2-D projections that are both low-discrepancy and present a Fourier spectrum approaching the Blue Noise spectrum.

Our tile-based method performs permutations on subsets of the original Sobol sequence. In the large space of all possible permutations, we choose those which better approach the target Blue Noise property, using pair-correlation statistics. We pre-calculate such optimized permutations for each possible Sobol sub-pattern, and store them in a Look-Up Table (LUT) efficiently accessible at runtime. We provide a complete and rigorous proof that such permutations preserve Sobol's net properties and thus the LDS properties of the point set in 2-D projections. Our muti-dimensional construction is computationally efficient, has relatively low memory footprint and supports adaptive sampling.

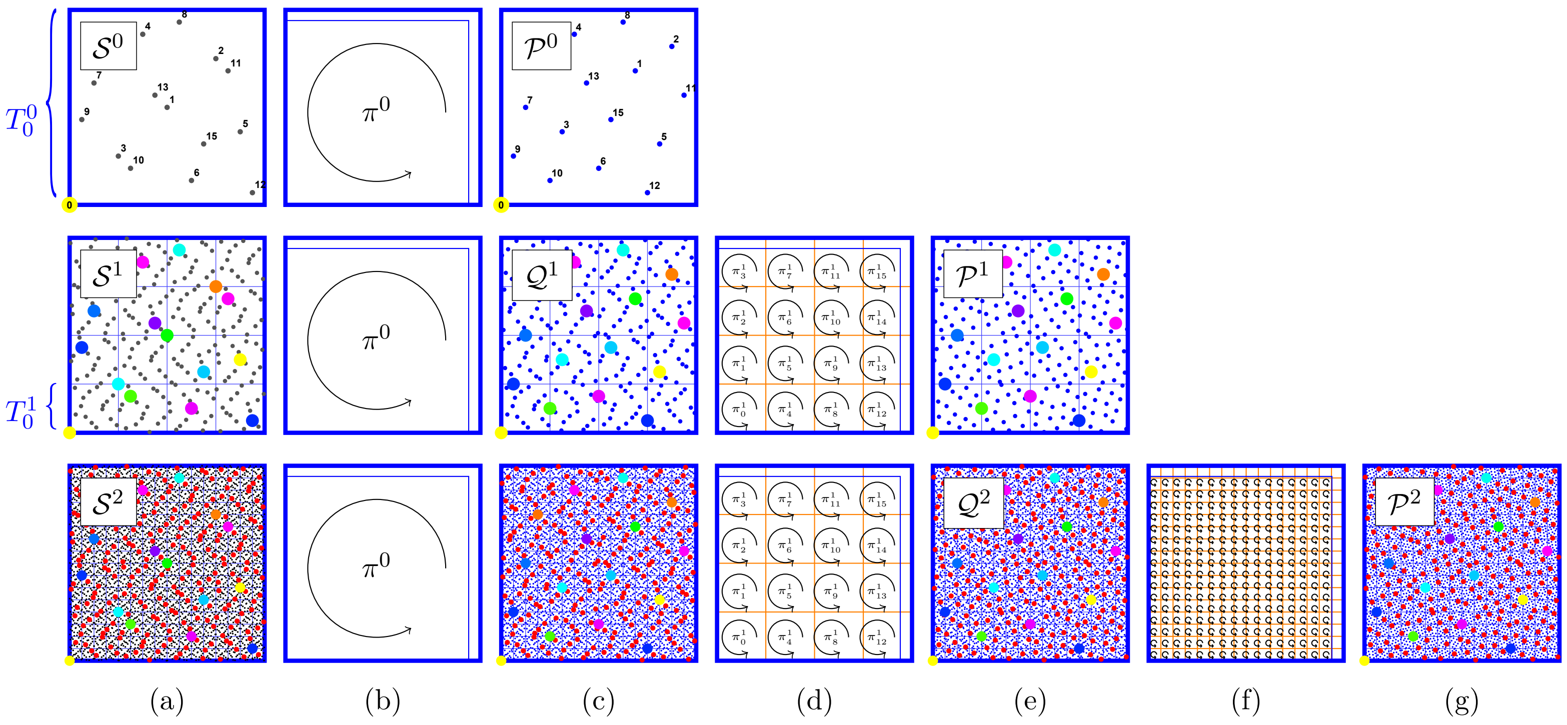
We start by presenting a general overview of our permutation (Section 6.1). Then we will dive into deeper details (Section 6.2 on page 117), wich will let us formally demonstrate its low discrepancy and sequentiality (Section 6.3 on page 122).

6.1 Overview

In this section, we provide a high-level, simplified and intuitive explanation of our scrambling. Rigorous definitions and technical details are provided in Section 6.2 on page 117. Section 6.1.1 presents our 2-D scrambling, which relies on a precomputed set of optimized local permutations. We then describe how we can build those permutations, along with the performances of the associated Look-Up Table in Section 6.1.2 on page 110 and 6.1.3 on page 114. Finally, we will see how this 2-D scrambling can help creating s-D sampling patterns Section 6.1.4 on page 114.

6.1.1 2-D Scrambling

Our sampler relies on independent scrambling of 2-D projections of the Sobol sequence. In this section we can therefore start by presenting how this scrambling applies on a single pair of dimensions without loss of generality.



Left column: original Sobol point sets (upper row: 16^1 points; middle row: 16^2 points; lower row: 16^3 points), generated using Sobol polynomials $x^2 + x^1 + 1$ and $x^4 + x^3 + 1$ for dimensions x and y. Note that all square partitions delimited by thin blue lines contain exactly 16 points each; this remains true for any power p of 16^p sampling points.
Upper row: We generate $\mathcal{P}^\lambda$ for $\lambda = 1$. The permutation π^0 makes first 16 points of the Sobol sequence more evenly distributed over the domain $[0, 1)^2$.
Middle row: We generate $\mathcal{P}^\lambda$ for $\lambda = 2$, iterating over $\mathcal{P}^1$. The permutation π^0 applied to 16^2 points of the Sobol sequence keeps the first 16 points well-distributed (large colored dots in the middle sub-figure, middle row), whereas the remaining 240 points (smaller blue dots) are unevenly distributed. Staged π_i^1 permutations make all first 256 points of the the sequence more evenly distributed (rightmost sub-image, middle row). Pivot points (the first point of each group of 16 points, according to Sobol's numbering) are shown as colored larger dots for the first 16^2 points and larger red dots for the first 16^3 points of the sequence. Note that after application of π^0 on 16^2 points, the pivots of permuted tiles of 16 points match the distribution of the first 16 points after application of π^0.
Lower row: We generate $\mathcal{P}^\lambda$ for $\lambda = 3$, iterating over $\mathcal{P}^2$. Similarly, 3 levels of permutations are needed in order to obtain even distribution of first 16^3 points of the sequence. Note that after application of π^0 on 16^3 points, the first 16 points are evenly distributed, as in upper row, whereas staged π_i^1 applied on 16^3 points keeps the first 16^2 points evenly distributed, as in the middle row. Only three levels of staged permutations are required to achieve an even distribution of all 16^3 points (rightmost sub-image, lower row). Please zoom in the sub-figures of the lower row to see fine details.

Fig. 6.1.: Illustration of our discrepancy preserving 2-D scrambling for spectrum optimization.

Our construction can be conceptually considered as a set of staged local permutations of tiles of K^2 samples of the initial Sobol LDS (the leftmost column in Figure 6.1 on the preceding page). The factor $K = 2^n, n \in \mathbb{N}$ is used to determine the number of samples of those tiles. Our basic 2-D scrambling is illustrated in Figure 6.1 on the facing page.

In this illustration, we work with tiled of $K^2 = 16$ samples. Note that similar constructions exist with tiles of $K^2 = 64$, $K^2 = 256$ or any K^2 with $K = 2^n, n \in \mathbb{N}*$ samples. In this illustration, we also used the pair of indices 3 and 7 for the Sobol sequence but note that that other pairs of indices can be used.

Our construction works by iteratively enriching an initial set. It starts at level $\lambda = 0$ with a set of K^2 Sobol samples. For those first K^2 points of the initial set (the top row in Figure 6.1 on the preceding page), a good pair of Owen's permutation trees [Owe95], chosen according to a given blue-noise criterion, can be found (see Section 6.1.2 on the following page for more details on how we compute them). Our pattern is then permuted using Owen's scrambling with those optimized permutation trees, to produce a more even distribution of sampling points (the rightmost sub-image of the top row in Figure 6.1 on the preceding page). Note that thanks to the properties of Owen's scrambling, this permutation preserves the t factor of the initial Sobol set of K^2 samples. (see Section 6.3 on page 122). In Figure 6.1 on the preceding page, we denote this permutation π^0. Please refer to Section 4.1.3 on page 76 for a detailed description of Owen's scrambling.

We then refine our set of K^2 samples to create a new set of $K^{2 \cdot 2} = 256$ samples. We do this by starting over from the Sobol set containing $K^{2 \cdot 2} = 256$ samples (the middle row in Figure 6.1 on the preceding page). We first apply the same permutation π^0 on the entire point set. Then, we subdivide our domain into K^2 tiles, containing K^2 samples each (see Section 6.2 on page 117). For each tile, denoted T_i^λ ($\lambda = 1$), we apply a new optimized permutation π_i^1. This generates an even distribution of all $K^{2 \cdot 2}$ samples, as shown in the rightmost sub-images of the middle row in Figure 6.1 on the preceding page. Note that after the first stage of permutation π^0, the $K^2 = 16$ first samples of the set are the same as the optimized samples from the previous optimized set (shown as large colored dots in the $\mathcal{Q}^1$ and $\mathcal{Q}^2$ images from Figure 6.1 on the facing page). As the second stage of permutations π_i^1 also does not move those first K^2 samples, we will refer to those points as pivots, because they are immovable samples w.r.t. previous stages.

It then goes similarly for the following iteration (the lower row in Figure 6.1 on the preceding page). We start with the set of $K^{2 \cdot 3} = 4096$ Sobol samples. We then undergo 3 stages of permutations, in order to keep first K^2 and $K^{2 \cdot 2}$ samples well-distributed, and achieve good distribution of all $K^{2 \cdot 3}$ samples. This process can be continued to higher orders λ of $K^{2(\lambda+1)}$ samples. Note that the first λ stages of permutations (that keeps the distribution of the first K^2, K^4, $\cdots$, $K^{2\lambda}$ samples) can be applied with a single xoring between the initial Sobol point set from the previous level $\mathcal{S}^{\lambda-1}$ and the optimized point set $\mathcal{P}^{\lambda-1}$ obtained from this Sobol set (this will be detailed in Section 6.2 on page 117). Applying those xor vectors on the initial Sobol set of the current level $\mathcal{S}^\lambda$ creates a temporary set $\mathcal{Q}^\lambda$,. This set is then locally scrambled to generate the optimized set $\mathcal{P}^\lambda$. This process will be detailed in section 6.2 on page 117.

Algorithm 5: Refining $\mathcal{P}^{\lambda-1}$ to $\mathcal{P}^{\lambda}$

```
input  : Point Set $\mathcal{P}^{\lambda-1}$ at level $\lambda - 1$ and the lookup table $LUT(\cdot)$.
output : Point Set $\mathcal{P}^{\lambda}$.
   // Construct $\mathcal{Q}^{\lambda}$ from $\mathcal{S}^{\lambda-1}$, $\mathcal{S}^{\lambda}$, and $\mathcal{P}^{\lambda-1}$
1  forall tiles $T_i^{\lambda}$ do
2      $\mathbf{s}^{(i)} \leftarrow \mathcal{S}^{\lambda-1} \cap T_i^{\lambda}$;
3      $\mathbf{p}^{(i)} \leftarrow \mathcal{P}^{\lambda-1} \cap T_i^{\lambda}$;
4      $\mathbf{v}_{\mathbf{r}}^{\lambda} \leftarrow \mathbf{s}^{(i)} \oplus \mathbf{p}^{(i)}$;
5      forall points $\mathbf{s} \in \mathcal{S}^{\lambda} \cap T_i^{\lambda}$ do
6          $\mathcal{Q}^{\lambda} \leftarrow \mathcal{Q}^{\lambda} \cup (\mathbf{s} \oplus \mathbf{v}_i^{\lambda})$;
7      end
8  end
   // Apply local permutations
9  forall tiles $T_i^{\lambda}$ do
10     $\{\mathbf{q}\}_i^{\lambda} \leftarrow \mathcal{Q}^{\lambda} \cap T_i^{\lambda}$;
11     $\tilde{\pi}_i^{\lambda}$ is randomly selected from LUT($\{\mathbf{q}\}_i^{\lambda}$);
12     $\{\mathbf{p}\}_i^{\lambda} \leftarrow \pi_i^{\lambda}(\mathcal{Q}^{\lambda})$;
13     $\mathcal{P}^{\lambda} \leftarrow \mathcal{P}^{\lambda} \cup \{\mathbf{p}\}_i^{\lambda}$;
14 end
15 return $\mathcal{P}^{\lambda}$
```

Algorithm 5 sums up the general framework for the application of our optimized Owen's permutations on Sobol point sets. We will now see how we can carefully choose our permutation trees, to ensure that our 2-D scrambling will have a spectrum close to the Blue Noise spectrum while ensuring that our local scramblings will not change the position of the points that were generated at the previous level (to preserve the sequentiality).

6.1.2 Local permutations with spectral control

In this section, we see how we can choose our 2-D local permutations to ensure that our scrambling preserves the discrepancy while creating a Blue Noise spectrum. Our permutations are based on Owen's scrambling [Owe95] (Section 4.1.3 on page 76) which means that a 2-D permutation is represented as a pair of Boolean trees. Our goal is thus to figure how to choose the flags of those trees, in order to both preserve the sequentiality of the whole global set and create locally Blue Noise distributions of samples. To do so, we follow a method similar in essence to the one presented in Section 4.4.2 on page 87 (that optimizes the Boolan trees using genetic algorithms). However, in our particular case of local independant permutations, further constraints are needed to ensure that the discrepancy of our global set is preserved.

Let us first consider that we have a point set $\mathcal{Q}^{\lambda}$ with the following property: each tile T_i^{λ} contains K^2 samples (we remind that $K = 2^n, n \in \mathbb{N}$), with a single pivot sample (see Figure 6.1 on page 108). We further assume that $\mathcal{Q}^{\lambda}$ is a $(t, \lambda, 2)$-net in base K^2, and that each pattern inside each tile T_i^{λ} of $\mathcal{Q}^{\lambda}$ is a $(t, 2n, 2)$-net in base 2 (we will come back to this in Section 6.2 on page 117 and 6.3 on page 122).

To obtain the point set $\mathcal{P}^\lambda$, we locally scramble the points inside each tile T_i^λ of $\mathcal{Q}^\lambda$ using optimized permutations. Each tile T_i^λ contains a different set of samples. Each unique set of samples requires its own permutation π_i^λ

To preserve sequentiality and discrepancy of the original 2-D set, each optimized permutation π_i^λ needs to fulfil the three following conditions (see Lemmas 4 on page 126 and 3 on page 125):

(i) π_i^λ does not move the pivot sample.

(ii) π_i^λ is $(t, 2n, 2)$-net preserving in base 2.

(iii) π_i^λ does not change the 1-D projections of the input pattern.

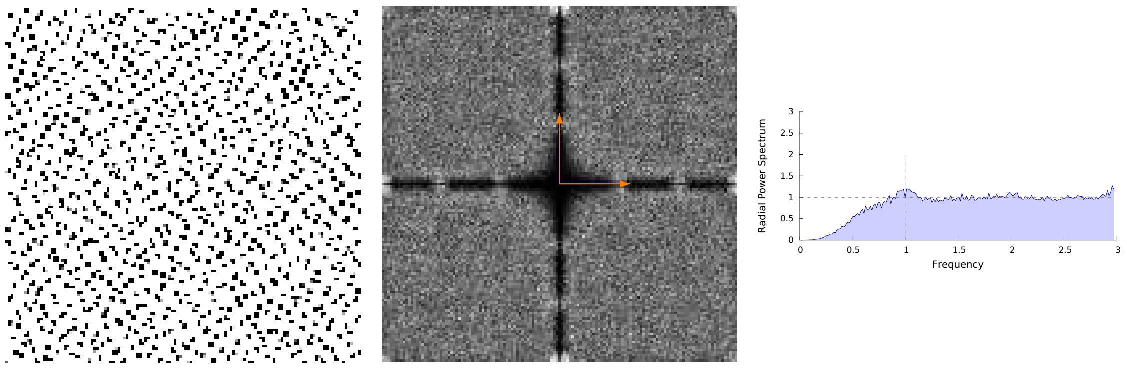
Property (i) is mandatory to ensure the sequentiality, Property (ii) is mandatory to ensure that the discrepancy is preserved within each tile, and Property (iii) ensures that the discrepancy of the global 2-D set is also preserved. We say that a permutation π_i^λ is an *admissible permutations* for a given pattern of samples if it fulfils the above mentioned constraints.

Our permutation applies an Owen's scrambling over the point set within T_i^λ. If we consider any set of s static Boolean trees of depth m, Owen's scrambling is known to be $(t, m, 2s)$-net preserving in base 2. Therefore, any permutation with $s = 2$ and $m = 2n$, validates property (ii).

However, it does not necessarily preserve the 1-D projections of samples (property (iii)) when coordinates are expressed on more than m digits, or when there are fewer than 2^m samples. To alleviate this, we apply the scrambling on the first m bits and exchange the trailing ones in order to preserve the 1-D projections (see Section 6.2 on page 117). As a consequence, for each tile T_i^λ, any such tree with $m = \log(K)^2$ induces a permutation π_r^λ satisfying (ii) and (iii).

To satisfy (i), we need to be able to scramble the point set while preserving the position of the pivot sample. This can be done using Owen's scrambling. If we list all the possible branches within the permutation tree, each of them will apply on a single pair of samples. Therefore, if along the path leading to the pivot sample, we set all the flags to 0, this sample will not be permuted when all others will move. We thus have m flags set to 0, while the remaining $K^2 - 1 - m$ flags are free, which gives us enough degrees of freedom to still have a control over the Fourier spectrum.

Using such admissible permutations, we can generate a multidimensional sequence with low discrepancy 2-D projections presenting a stratified spectrum (Fig. 6.2 on the following page). The computations of such random permutation trees is quite efficient as we simply fill the tree with random values apart from a branch that is fixed to 0. Furthermore, applying such is as computationally efficient as the original Owen's scrambling with finite small permutation trees. However, it does not lead to the Blue Noise spectrum we expect.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 6.2.: Characteristics of our sequence using admissible permutations

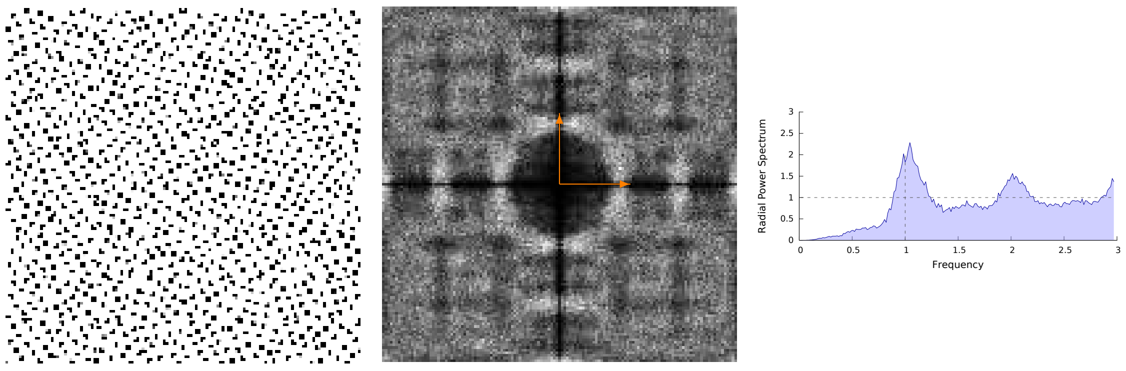
We will denote an *ad-hoc* permutation for a given pattern as an admissible permutation for this pattern that leads to a Fourier spectrum as close as possible to a Blue Noise spectrum.

Identifying such permutations among the set of all admissible permutations of a pattern can be very challenging. We look for transformations that optimize the spectral content of the point set, *e.g.* by increasing the minimum distance between points or obtaining a pattern who is as close as possible to a blue noise one. To characterize a pattern as a realization of a stationary and ergodic stochastic point process, we consider its pair correlation functions (PCF) which can be related to the spectral content of the point set (Section 2.3.1 on page 26 and [Ill+08]). To evaluate the quality of a pattern with respect to a blue noise one, we compute the l_∞ or l_2 norm of the two PCFs as discussed in [OG12] (in our experiments, both measures give similar results).

In dimension 2 for a given K, the point set $\{\mathbf{q}\}_i^\lambda$ (defined as the sub set of samples from $\mathcal{Q}^\lambda$ that belong to a tile T_i^λ, formally $\{\mathbf{q}\}_i^\lambda = \mathcal{Q}^\lambda \cap T_i^\lambda$) contains K^2 sample. For each tile, the permutation is encoded with s Boolean trees of depth m with $(K^2 - 1 - m)$ free flags. This gives us a total of 2^{K^2-1-m} permutations among which we want to find the ones that lead to the best Blue Noise spectrum.

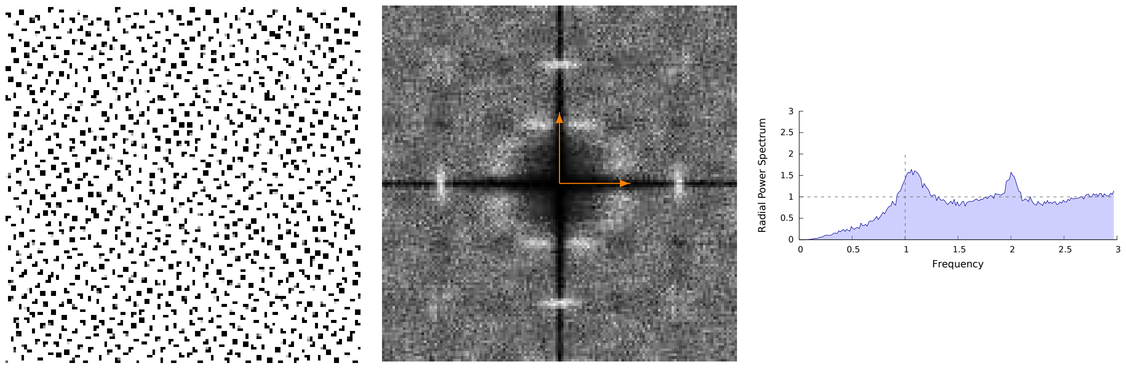
As an example, for $K = 4$, we have 2048^2 different admissible permutations to explore for a single pattern. This lets us perform an exhaustive listing of all those permutations on the GPU and keep the best one (which takes about 1 second on a GeForce GTX 760). For each permutation, we evaluate its quality using its PCF. We compare its PCF to the PCF of Blue Noise, and measure the l_∞ distance between the two. The one with the lowest distance is kept.

For $K = 4$ (16 points), an exhaustive search is possible. For $K > 4$, we optimize exhaustively the first 16 bits of the permutation tree and fill the underlying digits at random. As our GPU implementation performs this search in 1s, we can brute force the last digits to ensure that, even when randomly chosen, they do not harm the quality of the pattern. This whole process takes up to 2s, still on a GPU GeForce GTX 760.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 6.3.: Characteristics of our sequence using ad-hoc permutations and K=4. Note how a hierarchical construction increases the width of the black cross in the spectrum, this is due to the fact that the unaligned samples from the previous levels are more and more spaced out as we refine.



P_{1024} Fourier Spectrum on 8192 samples Radial Average on 8192 samples

Fig. 6.4.: Characteristics of our sequence using ad-hoc permutations and K=8.

Figure 6.3 and 6.4 present the characteristics of our scrambling when using ad-hoc permutations (for $K = 4$ and $K = 8$). Note that applying a given permutation is still very efficient. However, computing this permutation has become quite costly. Especially if you point out that generating K^{2n} samples implies that we have $1 + K^2 + K^4 + \cdots + K^{2(n-1)}$ tiles to scramble. However, as will be detailed in Section 6.1.3 on the following page, the recursive structure of Sobol implies that several tiles present similar sets of samples, allowing to reuse at times previously optimized permutations. A smaller K value increases the amount of similarity within tiles.

Once we have computed the set of *ad-hoc* permutations regarding a specific pattern, we store them in the lookup table, indexed by the pattern $\{\mathbf{q}\}_i^\lambda$. Note that multiple *ad-hoc* permutations can be stored for the pattern $\{\mathbf{q}\}_i^\lambda$. This allows us to randomize the scrambler by randomly picking a good permutation (see Algorithm 5 on page 110). The number of entries in the LUT is the number of distinct $\{\mathbf{q}\}_i^\lambda$ patterns.

6.1.3 Look Up Table

The LUT stores the *ad-hoc* permutations for each pattern $\{\mathbf{q}\}_i^\lambda$ in order to generate the set $\mathcal{P}^\lambda$. Since the set $\mathcal{Q}^\lambda$ is defined from $\mathcal{P}^{\lambda-1}$ (Fig. 6.1 on page 108), we fill the LUT while following the hierarchical construction. We start from $\lambda = 1$, and we generate successive $\mathcal{P}^\lambda$ and $\mathcal{Q}^\lambda$ sets. Each time we have to permute a pattern $\{\mathbf{q}\}_i^\lambda$, whose optimized permutation is not in the LUT, we explore admissible permutations and add the best one to the LUT (see Section 6.1.2 to see how we pick this best one).

All patterns $\{\mathbf{q}\}_i^\lambda$ have the same number of points K^2. The total number of possible distinct patterns, (and thus the upper bound on the LUT size) is difficult to estimate (*e.g.* $O((K^2!)^2)$ if we enumerate the set of Latin hypercube configurations, from which we must remove non-dyadic sets). However, as we start from the Sobol sequence which present a very repetitive structure, the number of distinct sets is limited. Empirically, as λ increases, the number of entries in the lookup table stabilizes quickly. As shown in Figure 6.5 on the next page, for $K = 2$ only 12 configurations exist up to 10^7 samples ($\lambda = 10$). For $K = 4$, only 512 configurations exist up to 10^6 samples ($\lambda = 6$). For $K = 8$, we found up to 1899 configurations for about 2.10^5 samples ($\lambda = 3$), but the table does not seem to have reached saturation (Fig. 6.5 on the facing page)).

In 2-D, a permutation is defined by 2 Boolean trees with $K^2 - 1$ nodes for $K := 2^n$. Hence, for $K = 4$ (resp. $K = 8$), a straightforward encoding of a permutation π_i^λ requires 30 bits (resp. 126 bits). Each entry in the lookup table is indexed by a pattern containing K^2 samples, each sample being encoded onto $2 \cdot 2 \cdot n$ bits. If we allow the LUT to contain m ad-hoc permutations for each pattern, we need

$$K^2 \cdot 4 \cdot n + m \cdot 2(K^2 - 1)$$

bits.

For $K = 4$ and $m = 1$, the LUT size is $512 \cdot (86 + 30) = 59392$ bits which is about 7 kbytes (about 485 kbytes for $K = 8$).

Note that this LUT contains optimized permutations for 2-D sampling patterns only. Unfortunately, our scrambling does not properly extend to higher dimensions (more details in Section 6.2). Therefore, to use our method with s-D point set, we can only use our optimized scrambling in some 2-D projections.

6.1.4 Using Our Sampler in Higher Dimensions

In many rendering applications, higher dimensional samples may be required. The permutation described above is only defined in 2-D. However, even though it only performs operations that are perfectly defined in s-D, our construction cannot be straightforwardly extended to s-D, as it would no longer preserve the discrepancy. The condition (*iii*) for admissible permutations only holds for 2-D samples. We instead extend our method to s-dimensions by relying on independent scrambling of several 2-D projections (see Figure 6.6 on page 116). In short, we can use a s-D

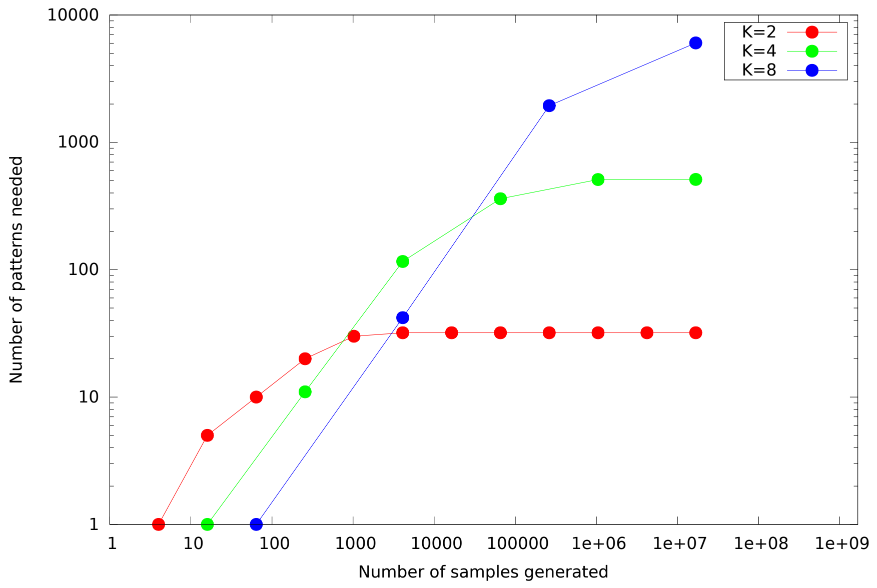


Fig. 6.5.: Size of our lookup table in number of entries need to generate a certain amount of samples, depending on the K factor used. Please note that this is in logarithmic scale.

Sobol sequence, optimize some 2-D projections with respect to pre-computed lookup tables and perform an *on-the-fly* random permutation of the remaining dimensions. Such an approach is made possible since the initial sequence is consistent through the dimensions and has LD properties. The selected 2-D projections will have low discrepancy properties and an optimized spectra. In rendering applications, selected projections could be the image plane, the lens, area lights, etc.

In order to have the $(t, \lambda, 2)$-net in base K^2 property on the optimized pair of dimensions, we need each set $\{\mathbf{s}\}^{\lambda}_{\mathbf{r}} := \mathcal{S}^{\lambda} \cup T^{\lambda}_{i}$ to be a $(t, ns, 2)$-net in base 2 in these dimensions (see Prop. 4 on page 126). In our implementation, we have used certain combinations of primitive polynomials' indices, found empirically. When K=4, we used for our 4 dimensions the indices $(1, 2)$ and the indices $(3, 7)$. When K=8, we used for our 6 dimensions the indices the indices $(1, 4)$, $(2, 3)$ and the indices $(5, 7)$. Note that as K increases, more and more dimensions can be optimized.

For the remaining dimensions, when the number of samples is fixed, one can use a classical Owen's scrambling of the point coordinates with a tree of fixed depth (Fig. 6.7 on the next page). In our hierarchical setting when the number of points is unknown and increases on the fly, we simulate a tree of infinite depth by iteratively expanding a tree of fixed depth, as detailed in Section 4.2 on page 79.

From here, you should have all the intuitions required to understand how our method works and why it works. However, to formally prove how our scrambling preserves the Low Discrepancy property of Sobol projections, we need a much more formal and detailed description of our sampler. This will be the focus of the next section.

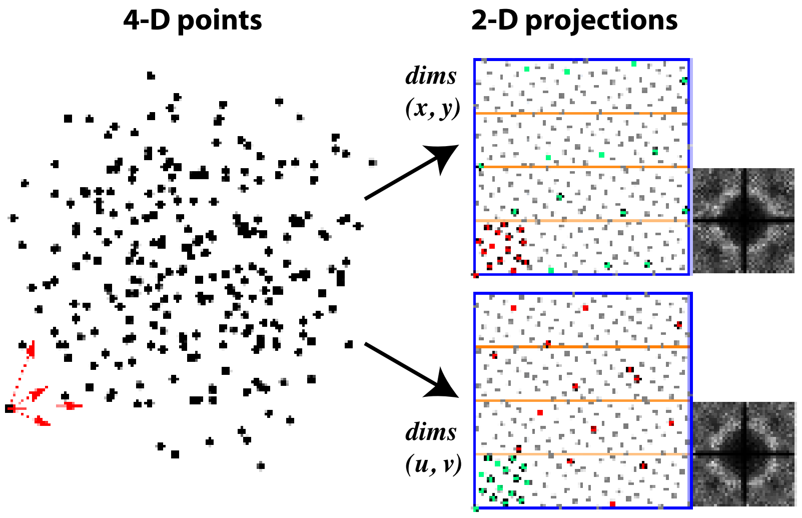


Fig. 6.6.: 4-D point set generated with our system, together with two 2-D projections and their Fourier power spectra. Note that subsets of 16 points that occupy the same strata in one projection (red subsets in $\{(x, y)\}$ projections and green subsets in $\{(u, v)\}$ projections), occupy different strata in the other projection.

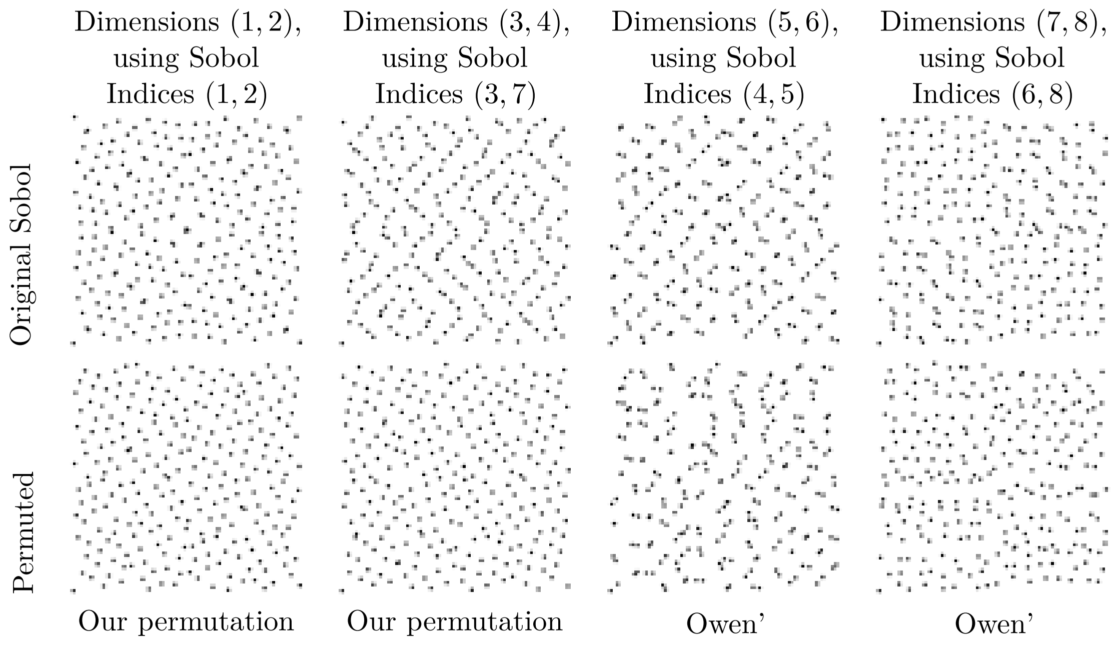


Fig. 6.7.: Comparison between an original 8-D Sobol set and this point set scrambled using our method. Note that we used $K = 4$, and thus can only apply our scrambling to the the indices $(1, 2)$ and $(3, 7)$. The remaining dimensions were permuted using Owen', as described in Chapter 4 on page 74.

We will further validate our method in Chapter 7 on page 133 by performing spectral/discrepancy/aliasing analysis of the achieved distributions, and provide variance analysis for several target integrands of theoretical and practical interest.

6.2 Technical description

In this section, we will present our method in much deeper details. Thanks to those formal definitions, we will be able to demonstrate that it preserves the discrepancy of the samples in Lemmas 3 on page 125 , 4 on page 126 and 5 on page 129 (Section 6.3 on page 122). Note that we will detail our method in a supposedly s-D context, but in practice, it only preserves the discrepancy for 2-D point sets. We also stress that all illustrations in Section 6.2 and 6.3 are given for $K = 2$ (4 samples per tile), to ease understanding. In practice, our method is usually applied with $K = 4$ and $K = 8$.

As a short preliminary, we remind that we denote the binary representation of a sample as follows. Any s-D sample $\mathbf{x} \in [0,1)^s$ can be expressed as $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_s)$ with

$$\mathbf{x}_d := \sum_{i=1}^{\infty} x_d^i 2^{-1-i}$$

$$\mathbf{x}_d = .x_d^1 x_d^2 x_d^3 \dots,$$

where x_d^i is the i^{th} bit of $\mathbf{x}_d$. Note that the samples are defined in $[0,1)^s$, therefore, x_d^1 is the most significant bit, meaning x_d^1 corresponds to 0.5, x_d^2 to 0.25, ...

We stress that the proofs in this section require knowledge of the notations and definitions of (t,k,s)-nets and sequences that were presented in Section 3.3 on page 57.

6.2.1 Domain subdivision

Our scrambling considers a regular subdivision of the domain $[0,1)^s$ with a subdivision factor $K := 2^n$, $n \in \mathbb{N}^*$. We denote λ the level of subdivision of the domain (Fig 6.8 on the following page).

This subdivision leads to square tiles. Each time we subdivide the domain, we subdivide the tiles at level λ by the factor K, creating $K^{(\lambda+1)s}$ new tiles at level $\lambda + 1$. A tile $T_{\mathbf{r}}^{\lambda}$ is a s-D interval of space defined at a level λ spanning over $\left[\frac{\mathbf{r}_d}{K^\lambda}, \frac{\mathbf{r}_d+1}{K^\lambda}\right)^s$. The union of all the tiles at a level λ forms a partition of the domain $[0,1)^D$ (Fig 6.9 on the next page).

To sample the domain subdivided at a level λ, we generate the first $K^{s(\lambda+1)} = 2^{ns(\lambda+1)}$ samples from the Sobol sequence [Sob67] (Fig 6.10 on the following page).

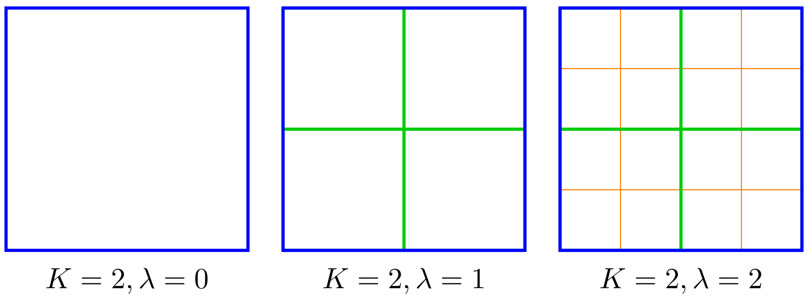


Fig. 6.8.: Subdivision of the domain $[0,1)^s$ with increasing λ.

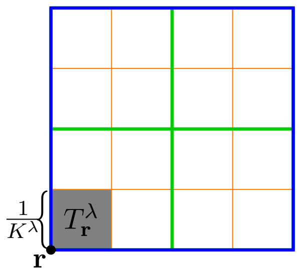
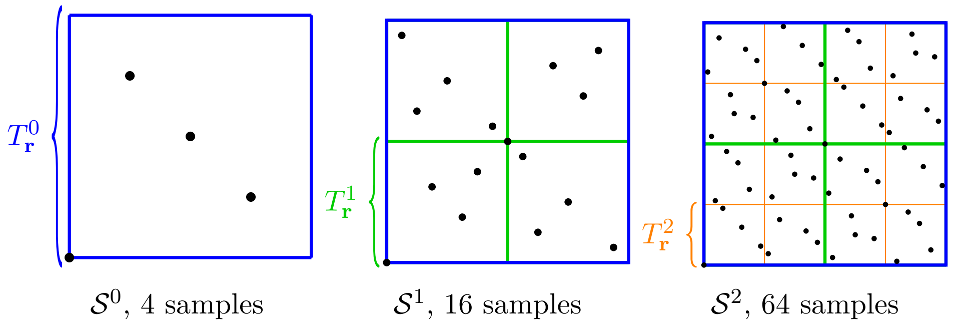


Fig. 6.9.: $\lambda = 2, K = 2, s = 2$. Each interval $T^\lambda_{\mathbf{r}}$ will contain K^s samples.



$T^0_{\mathbf{r}}$
$T^1_{\mathbf{r}}$
$T^2_{\mathbf{r}}$
$\mathcal{S}^0$, 4 samples
$\mathcal{S}^1$, 16 samples
$\mathcal{S}^2$, 64 samples

Fig. 6.10.: $s = 2, K = 2$. Generating the first $K^{s(\lambda+1)}$ samples of the Sobol sequence with $t = 0$ fills each tile $T^\lambda_{\mathbf{r}}$ with K^s samples.

By definition of the Sobol sequence, $\mathcal{S}^\lambda$ is a (t, k, s)-net in base $b = 2$ (Section 3.3 on page 57). Since $\mathcal{S}^\lambda$ contains $2^{ns(\lambda+1)}$ samples, we have $b^k = 2^{ns(\lambda+1)}$ and therefore $\mathcal{S}^\lambda$ is a $(t, ns(\lambda+1), s)$-net in base $b = 2$. Our set $\mathcal{S}^\lambda$ (defined as the set containing the first $K^{s(\lambda+1)}$ samples of the sobol sequence with $K := 2^n$) is a $(t, ns(\lambda+1), s)$-net in base $b = 2$. In our case, we assume that the value t of our (t, k, s)-nets is the minimal possible value, for which (and over which) all intervals are guaranteed, we have the following property:

Property 1 *$\forall \lambda \in \mathbb{N}$, if $\mathcal{S}^\lambda$ is a $(t, ns(\lambda+1), s)$-net in base $b = 2$, $\mathcal{S}^\lambda$ is a $(\rceil \frac{t}{ns} \lceil, \lambda+1, s)$-net in base K^s, with $K := 2^n$.*

From Property 1, we have that if the t value for the Sobol sequence verifies $t \leq ns$, we are guaranteed to have K^s samples inside each tile $T^\lambda_{\mathbf{r}}$. If $t > ns$, we can still have K^s samples in each tile, there is just no guarantee for it. Note that in practice, we also never have $t \leq ns$.

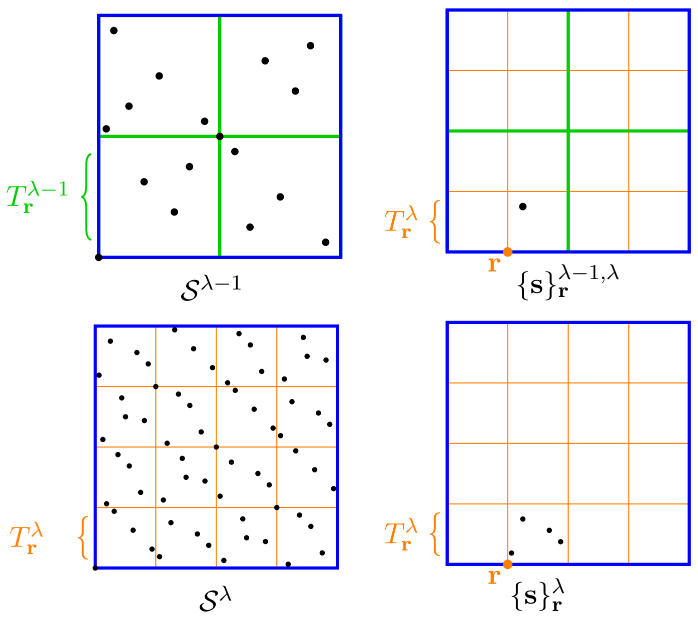

$T_{\mathbf{r}}^{\lambda-1}$
$\mathcal{S}^{\lambda-1}$
$T_{\mathbf{r}}^{\lambda}$
$\mathbf{r}$
$\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}$
$T_{\mathbf{r}}^{\lambda}$
$\mathcal{S}^{\lambda}$
$T_{\mathbf{r}}^{\lambda}$
$\mathbf{r}$
$\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}$

Fig. 6.11.: $s = 2, K = 2, \lambda = 2$. $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}$ is the intersection between $\mathcal{S}^{\lambda}$ and the tile $T_{\mathbf{r}}^{\lambda}$. Similarly, $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}$ is the intersection between $\mathcal{S}^{\lambda-1}$ and the tile $T_{\mathbf{r}}^{\lambda}$.

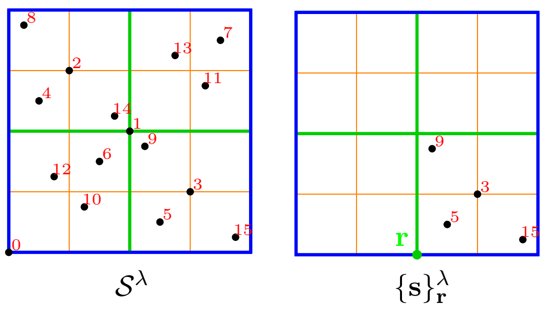


Fig. 6.12.: $s = 2, K = 2, \lambda = 1$. Indices of the samples in $\mathcal{S}^{\lambda}$ and of the samples in $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}$.

We now define $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda} := \mathcal{S}^{\lambda} \cap T_{\mathbf{r}}^{\lambda}$ the set of samples from $\mathcal{S}^{\lambda}$ that belong to the tile $T_{\mathbf{r}}^{\lambda}$. We also define $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda_0,\lambda_1} := \mathcal{S}^{\lambda_0} \cap T_{\mathbf{r}}^{\lambda_1}$ the set of sample from $\mathcal{S}^{\lambda_0}$ that belong to the tile $T_{\mathbf{r}}^{\lambda_1}$ (see Figure 6.11).

It is important to note that since the samples of $\mathcal{S}^{\lambda}$ come from the Sobol sequence, they are indexed [Sob67]. We will denote $\mathbf{s}^{(i)}$ the i^{th} sample of the sequence $\mathcal{S}^{\lambda}$. A reader familiar with the Sobol sequence may note that the samples of $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}$ are also indexed but their indices i are not consecutive (Fig 6.12).

We then permute the samples from $\mathcal{S}^{\lambda}$, where $\mathcal{S}^{\lambda}$ is a set containing the first $K^{s(\lambda+1)}$ samples of the Sobol sequence. We do this in such a way that the newly generated set of samples $\mathcal{P}^{\lambda}$ in $[0,1)^s$ has the following properties:

- If $s = 2$, $\forall\lambda$, if $\mathcal{S}^{\lambda}$ is a $(t, \lambda+1, s)$-net in base K^s, $\mathcal{P}^{\lambda}$ is a $(t, \lambda+1, s)$-net in base K^s (Lemma 5 on page 129).

- $\forall \lambda$, $\mathcal{P}^\lambda$ presents a user defined Fourier spectrum.
- $\forall \lambda > 0$, $\mathcal{P}^{\lambda-1} \subset \mathcal{P}^\lambda$ (otherwise P wouldn't be a sequence), and the samples in $\mathcal{P}^\lambda$ are indexed (Lemma 6 on page 129).

To ensure those properties, our algorithm creates a set $\mathcal{P}^\lambda$ by permuting the set $\mathcal{S}^\lambda$ while taking as input the two sets, $\mathcal{P}^{\lambda-1}$ and $\mathcal{S}^{\lambda-1}$. Note that we define $\mathcal{P}^{-1}$ as the point set containing a single sample $\mathbf{s} = 0$, and $\mathcal{S}^{-1} = \mathcal{P}^{-1}$. The sets $\mathcal{S}^\lambda$ and $\mathcal{S}^{\lambda-1}$ are coming from the same specific Sobol sequence, with indices chosen such that $\mathcal{S}^\lambda$ will be stratified $\forall\lambda$. The use of the Sobol sequence also ensures that $\mathcal{S}^{\lambda-1} \subset \mathcal{S}^\lambda$.

Then, our permutation is done in two steps (Fig. 6.1 on page 108)

1. A first permutation, denoted Ψ, applies local digital shifts on $\mathcal{S}^\lambda$ to create a new point set $\mathcal{Q}^\lambda$ such that $\mathcal{P}^{\lambda-1} \subset \mathcal{Q}^\lambda$.
2. A second permutation, denoted Π, applies local permutations $\pi^\lambda_{\mathbf{r}}$ on $\mathcal{Q}^\lambda$ to create a new point set $\mathcal{P}^\lambda$ such that it presents the targeted Fourier spectrum.

6.2.2 Creating $\mathcal{Q}^\lambda$ from $\mathcal{S}^\lambda$ and $\mathcal{P}^{\lambda-1}$

The aim of our first permutation Ψ is to create a set $\mathcal{Q}^\lambda$ from $\mathcal{S}^\lambda$, so that $\mathcal{P}^{\lambda-1} \subset \mathcal{Q}^\lambda$. To achieve this, we first compute a set of vectors, such that each vector represent the displacement between each sample of $\mathcal{S}^{\lambda-1}$ and $\mathcal{P}^{\lambda-1}$. Since the samples are indexed, both in $\mathcal{S}^\lambda$ and in $\mathcal{P}^\lambda$, we can match those sets on a one to one basis. This will let us xor each sample of $\mathcal{S}^{\lambda-1}$ with its corresponding sample in $\mathcal{P}^{\lambda-1}$ (the one with the same index).

Therefore, our new vector set, denoted $\mathcal{V}^{\lambda-1}$ is also ordered, and its i^{th} vector, denoted $\mathbf{v}^{(i)}$ is formally defined as

$$\mathbf{v}^{(i)} := \mathbf{s}^{(i)} \oplus \mathbf{p}^{(i)}, \tag{6.1}$$

where $\mathbf{s}^{(i)}$ is the i^{th} sample of $\mathcal{S}^{\lambda-1}$ and $\mathbf{p}^{(i)}$ is the i^{th} sample of $\mathcal{P}^{\lambda-1}$.

$\mathcal{S}^{\lambda-1}$ is stratified with a single sample inside each tile $T^\lambda_{\mathbf{r}}$. Thus, each vector $\mathbf{v}^{(i)}$ is associated with one and only one tile $T^\lambda_{\mathbf{r}}$. As each vector $\mathbf{v}^{(i)}$ is composed from a sample $\mathbf{s}^{(i)}$, which uniquely belongs to a tile $T^\lambda_{\mathbf{r}}$. We denote $\mathbf{v}^\lambda_{\mathbf{r}}$ the vector $\mathbf{v}^{(i)}$ associated with the tile $T^\lambda_{\mathbf{r}}$. We can now define formally $\mathcal{V}^{\lambda-1}$ as

$$\mathcal{V}^{\lambda-1} := \bigcup_{\mathbf{r}} \left[\mathbf{v}^\lambda_{\mathbf{r}}\right], \tag{6.2}$$

which is illustrated Figure 6.13 on the facing page.

This leads to the formal definition of $\mathcal{Q}^\lambda$ as

$$\mathcal{Q}^\lambda := \bigcup_{\mathbf{r}} \{\{\mathbf{s}\}^\lambda_{\mathbf{r}} \oplus \mathbf{v}^\lambda_{\mathbf{r}}\}, \tag{6.3}$$

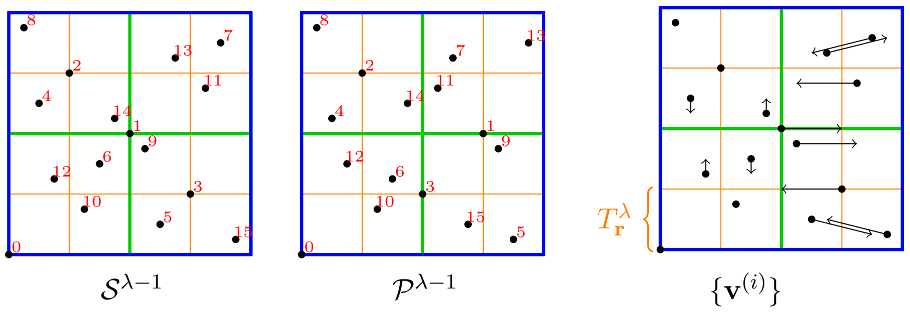


Fig. 6.13.: $s = 2, K = 2, \lambda = 2$. Each vector $\mathbf{v}^{(i)}$ is made by xoring the samples $\mathbf{s}^{(i)} \in \mathcal{S}^{\lambda-1}$ and $\mathbf{p}^{(i)} \in \mathcal{P}^{\lambda-1}$

where we define the $\oplus$ operation between a set of sample $\mathcal{X}$ and a single vector $\mathbf{v}$ as $\mathcal{X} \oplus \mathbf{v} := \{\mathbf{x}^{(i)} \oplus \mathbf{v}\}$ where $\mathbf{x} \in \mathcal{X}$. This is illustrated Figure 6.14.

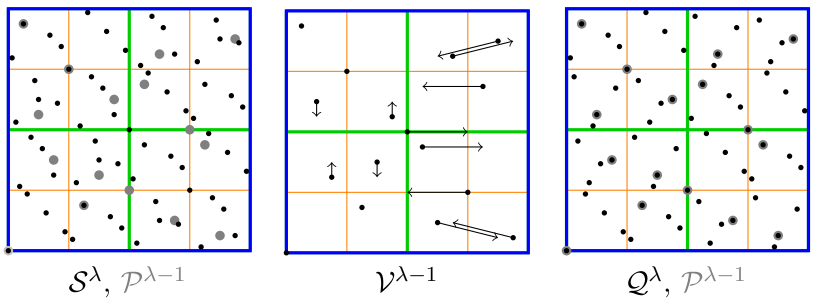

$\mathcal{S}^\lambda$, $\mathcal{P}^{\lambda-1}$
$\mathcal{V}^{\lambda-1}$
$\mathcal{Q}^\lambda$, $\mathcal{P}^{\lambda-1}$

Fig. 6.14.: $s = 2, K = 2, \lambda = 2$. The set $\mathcal{Q}^\lambda$ is made by xoring each samples of each set $\{\mathbf{s}\}_\mathbf{r}^\lambda$ with the vector $\mathbf{v}_\mathbf{r}^\lambda$ associated to the tile $T_\mathbf{r}^\lambda$

We also note that each pattern $\{\mathbf{s}\}_\mathbf{r}^\lambda$ is xored with a single vector to create $\{\mathbf{q}\}_\mathbf{r}^\lambda$, thus, as the `xor` operator is net preserving [Lem09], we have the following property for $\mathcal{Q}^\lambda$

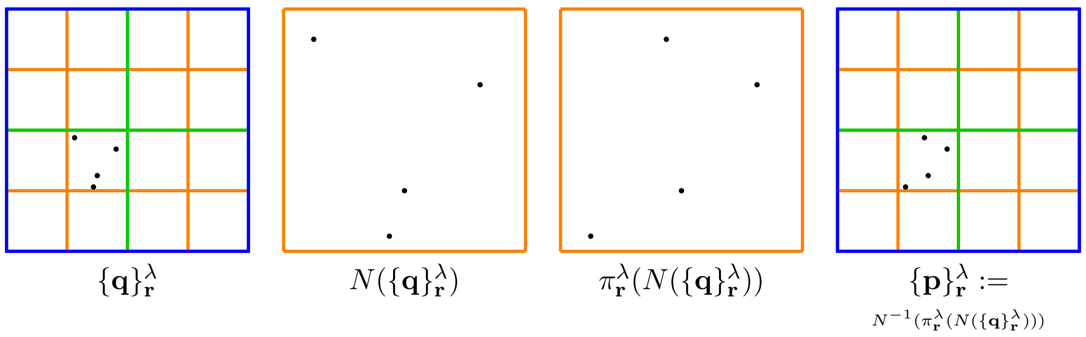
Property 2 *If* $\forall T_\mathbf{r}^\lambda$, $N(\{\mathbf{s}\}_\mathbf{r}^\lambda)$, *is a* (t, ns, s)*-net in base* 2, *then* $\forall T_\mathbf{r}^\lambda$, $N(\{\mathbf{q}\}_\mathbf{r}^\lambda)$, *is a* (t, ns, s)*-net in base* 2.

Then, from the point set $\mathcal{Q}^\lambda$, we can create our optimized point set $\mathcal{P}^\lambda$ by locally scrambling the samples.

6.2.3 Creating $\mathcal{P}^\lambda$ from $\mathcal{Q}^\lambda$

Then, we permute the samples inside each tile $T_\mathbf{r}^\lambda$ of $\mathcal{Q}^\lambda$ to create $\mathcal{P}^\lambda$. Note that there are K^s samples of $\mathcal{Q}^\lambda$ inside $T_\mathbf{r}^\lambda$, forming a point set denoted $\{\mathbf{q}\}_\mathbf{r}^\lambda$. Each set $\{\mathbf{q}\}_\mathbf{r}^\lambda$ contains a unique sample $\mathbf{p}$ such that $\mathbf{p} \in \mathcal{P}^{\lambda-1}$.

Note that each set $\{\mathbf{q}\}_\mathbf{r}^\lambda$ contains samples with coordinates that only cover the tile $T_\mathbf{r}^\lambda$ (Fig. 6.15 on the following page). We denote $N(\{\mathbf{q}\}_\mathbf{r}^\lambda)$ the operation that scale this set to the domain $[0, 1)^s$, with $N^{-1}(N(\{\mathbf{q}\}_\mathbf{r}^\lambda)) = \{\mathbf{q}\}_\mathbf{r}^\lambda$.



$\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}$
$N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda})$
$\pi_{\mathbf{r}}^{\lambda}(N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}))$
$\{\mathbf{p}\}_{\mathbf{r}}^{\lambda} :=$
$N^{-1}(\pi_{\mathbf{r}}^{\lambda}(N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda})))$

Fig. 6.15.: $s = 2, K = 2, \lambda = 2$. We permute each tile $T_{\mathbf{r}}^{\lambda}$ from $\mathcal{Q}^{\lambda}$ to create a new arrangement of samples within each tile with the desired spectral properties.

Once we have this set, scaled to the domain $[0,1)^s$, we can associate with each tile $T_{\mathbf{r}}^{\lambda}$ a permutation $\pi_{\mathbf{r}}^{\lambda}$, that have the following properties (denoted as (*i*), (*ii*) and (*iii*) in Section 6.1):

Property 3 *If $N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda})$, is a (t,k,s)-net in base 2, $N(\pi_{\mathbf{r}}^{\lambda}(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}))$ is also a (t,k,s)-net in base* 2.

Property 4 *If $\mathbf{p} \in \{\mathbf{q}\}_{\mathbf{r}}^{\lambda}$ and $\mathbf{p} \in \mathcal{P}^{\lambda-1}$, $\mathbf{p} \in \pi_{\mathbf{r}}^{\lambda}(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda})$.*

Property 5 *$\pi_{\mathbf{r}}^{\lambda}$ preserves the 1-D projections of $\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}$.*

We can now formally define $\mathcal{P}^{\lambda}$ with

$$\mathcal{P}^{\lambda} := \bigcup_{\mathbf{r}} N^{-1}(\pi_{\mathbf{r}}^{\lambda}(N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}))) \tag{6.4}$$

An possible construction algorithm to generate admissible permutations π is given in Section 6.1 on page 107 and is detailed in Section 6.3.6 on page 130.

6.3 Proofs

We will now prove that our scrambling preserves the net properties of $\mathcal{S}^{\lambda}$. We will first define some notations, and then we will prove that both Ψ and Π are $(t, \lambda+1, 2)$-net preserving in base K^s. Finally, we will prove that $\mathcal{P}^{\lambda}$ does contain the samples of $\mathcal{P}^{\lambda-1}$, identically indexed.

6.3.1 Notations

In this section we present all the notations that we use in the following proofs. Please note that even though our permutation is valid in s-D, it only preserves the net properties in 2-D. Therefore all those notations consider a 2-D domain.

Each 2-D sample (x, y) in a point set $\mathcal{X}$, where all the samples in $\mathcal{X}$ are defined on $ns(\lambda + 1)$ bits, and where $\mathcal{X}$ is a $(t, \lambda + 1, 2)$-net, can be expressed in binary as

$$x = \sum_{i=0}^{ns(\lambda+1)} x^i 2^{-1-i},$$

$$y = \sum_{i=0}^{ns(\lambda+1)} y^i 2^{-1-i},$$

where x^i is the i^{th} bit of x and y^i is the i^{th} bit of y.

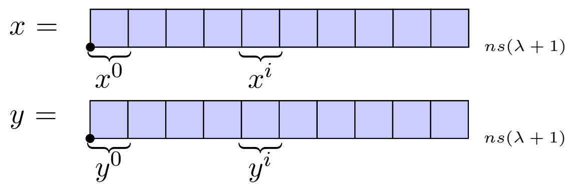


Fig. 6.16.: Binary representation of a sample $(x, y) \in [0, 1)^s$

We define $x|_a$ (resp. $y|_a$) as a binary truncating of the x (resp. y) coordinate of a sample (x, y) where

$$x|_a = \sum_{i=0}^{a} x^i 2^{-1-i},$$

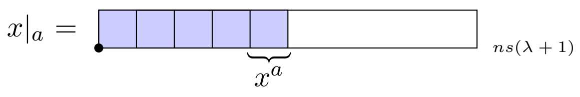


Fig. 6.17.: Binary representation of $x|_a$

We extend this definition to $(x, y)|_a$, for $(x, y) \in \mathcal{X}$ in a specific way. We thus define $(x, y)|_a$ as

$$(x, y)|_a = (x|_a, y|_{ns(\lambda+1-t)-a})$$

This means that we truncate a total of $ns(\lambda + 1 - t)$ bits from (x, y), truncating a bits from x and $ns(\lambda + 1 - t) - a$ bits from y.

Finally, we define $\mathcal{X}|_a$ as $\{(x, y)|_a, \forall (x, y) \in \mathcal{X}\}$, meaning that $\mathcal{X}|_a$ contains truncated samples from $\mathcal{X}$.

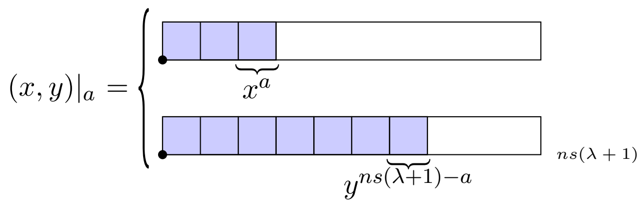


Fig. 6.18.: Binary representation of $(x,y)|_a$

Each elementary interval $\mathcal{J}^{q_0}$ of $\mathcal{X}$, is of size $\frac{1}{K^{sq_0}}, \frac{1}{K^{s(\lambda+1-t-q_0)}}$, with $0 \leq q_0 \leq \lambda+1-t$,

$$\begin{aligned} q_0 + q_1 &= \lambda + 1 - t \\ \Leftrightarrow q_1 &= \lambda + 1 - t - q_0. \end{aligned}$$

It can thus be encoded in binary with nsq_0 bits for x and $ns(\lambda+1-t-q_0) = ns(\lambda+1-t) - nsq_0$ for y. Therefore, for all sample $(x,y) \in \mathcal{X}$, their truncated version $(x,y)|_{nsq_0}$ is equivalent to the minimal corner of the elementary interval $\mathcal{J}^{nsq_0}$ that contains it.

Following this, a point set $\mathcal{X}$ is a $(t,k,2)$-net in base K^s if each sample $(x,y)|_{nsi}$ is present K^{st} times in the set $\mathcal{X}|_{nsi}$, with $0 \leq i < k-t$.

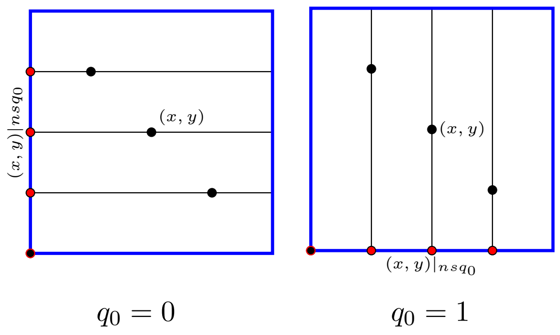


Fig. 6.19.: $K = 2, n = 1$. $\mathcal{Q}^0$ is a $(0,1,s)$-net in base 2^2 iff $\forall q_0 \in \mathbb{N}$ with $0 \leq q_0 \leq 1$, each sample $(x,y)|_{2*2*q_0}, (x,y) \in \mathcal{Q}^0$ appears exactly one time in $\mathcal{Q}^\lambda|_{2\cdot 2\cdot q_0}$.

The direct corollary is that any permutation Π is $(t,k,2)$-net preserving if $\forall a \in \mathbb{N}, a \leq k-t$,

$$\Pi(\mathcal{X})|_a = \mathcal{X}|_a. \tag{6.5}$$

Finally, we have one last major property, illustrated Figure 6.20 on the next page.

Property 6 *$\forall (x,y) \in [0,1)^2$, if $(x|_a, y|_{a'}) = (x'|_a, y'|_{a'})$ then $\forall b \leq a$ and $\forall b' \leq a'$, $(x|_b, y|_{b'}) = (x'|_b, y'|_{b'})$*

This property means that if we truncate the x coordinate (resp. y) of the samples of a set to a (resp. b) digits, and if this truncated set is equal to the truncated set from another point set, then the subsets of those sets that are more truncated, with $a' < a$ (resp. $b' < b$), are also equal.

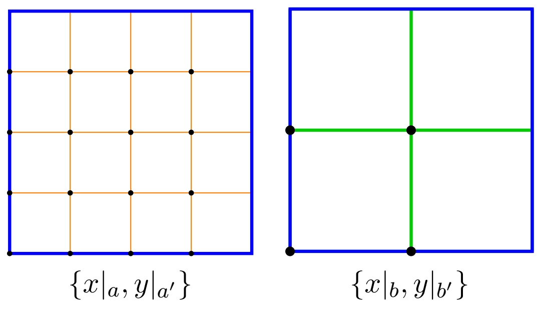


Fig. 6.20.: Geometric illustration of Property 6

6.3.2 Our permutation Π is $(t, \lambda+1, 2)$ preserving in base K^2

Lemma 3 *For $s = 2$, if $\mathcal{Q}^\lambda$ is a $(t, \lambda+1, 2)$-net in base K^2, $\mathcal{P}^\lambda$ is a $(t, \lambda+1, 2)$-net in base K^2.*

Proof 2 *The point set $\mathcal{P}^\lambda$ is defined as $\Pi(\mathcal{Q}^\lambda)$, where Π applies a scrambling $\pi^\lambda_{\mathbf{r}}$ over each set $N(\{\mathbf{q}\}^\lambda_{\mathbf{r}})$ from $\mathcal{Q}^\lambda$ (Equation 6.4 on page 122). The operator N removes the first $n\lambda$ bits of the samples in $\{\mathbf{q}\}^\lambda_{\mathbf{r}}$. Therefore, those first $n\lambda$ bits will be unaffected by the permutation. We apply N^{-1} by re-adding the first $n\lambda$ bits of $\{\mathbf{q}\}^\lambda_{\mathbf{r}}$.*

In our particular case, the permutations $\pi^\lambda_{\mathbf{r}}$ are not any random permutation but are part of a set of admissible permutations (see Section 6.1.2 on page 110 for details). Each permutation $\pi^\lambda_{\mathbf{r}} \in \Pi$ has two important properties.

First, $\pi^\lambda_{\mathbf{r}}$ preserves the net properties of $N(\{\mathbf{q}\}^\lambda_{\mathbf{r}})$, and second, each set $N(\{\mathbf{q}\}^\lambda_{\mathbf{r}})$ is a $(t', ns, 2)$-net in base 2 as it comes from the Sobol sequence. From here, we can translate property 3 on page 122 in binary as

$$(x|_{n\lambda+q'_0}, y|_{n\lambda+ns-t'-q'_0}) = \Pi\left((x|_{n\lambda+q'_0}, y|_{n\lambda+ns-t'-q'_0})\right) \tag{6.6}$$

$\forall q'_0, q'_0 \in \mathbb{N}, 0 \le q'_0 \le ns - t', q'_0 + q'_1 = ns - t' \Leftrightarrow q'_1 = ns - t' - q'_0$.

Furthermore, since $\pi^\lambda_{\mathbf{r}}$ preserves the 1-D projections of samples we have for $q'_0 = 0$,

$$(x|_{ns(\lambda+1)}, y|_{n\lambda+ns-t'-q'_0}) = \pi^\lambda_{\mathbf{r}}\left((x|_{ns(\lambda+1)}, y|_{n\lambda+ns-t'-q'_0})\right) \tag{6.7}$$

and

$$(x|_{n\lambda}, y|_{ns(\lambda+1)}) = \pi^\lambda_{\mathbf{r}}\left((x|_{n\lambda}, y|_{ns(\lambda+1)})\right). \tag{6.8}$$

In 2-D, the elementary intervals in $\mathcal{Q}^\lambda$ for $q_0 + q_1 = \lambda + 1 - t$ are of size $\frac{1}{2^{nsq_0}} \times \frac{1}{2^{nsq_1}}$ with $q_1 = \lambda + 1 - t - q_0$. This means that each elementary interval can be encoded using nsq_0 bits for x and $ns(\lambda+1-t-q_0)$ bits for y. As $\mathcal{Q}^\lambda$ is a $(t, \lambda+1, s)$-net in base K^s, each sample $(x, y)|_{nsq_0}, (x, y) \in \mathcal{Q}^\lambda$ appears exactly K^{st} times in $\mathcal{Q}^\lambda|_{nsq_0}$. Therefore, $\forall q_0 \in \mathbb{N}$ with $0 \le q_0 \le \lambda + 1 - t$, each sample from the set $\mathcal{Q}^\lambda|_{nsq_0}$ stands for an elementary interval.

From here, we have 3 possibilities

- *Either* $nsq_0 \leq n\lambda$, *in which case from 6 on page 124 and equation 6.8 on the preceding page, we have* $\mathcal{Q}^\lambda|_{nsq_0} = \mathcal{P}^\lambda|_{nsq_0}$
- *Either* $ns(\lambda+1-t-q_0) \leq n\lambda$, *in which case from 6 on page 124 and equation 6.7 on the preceding page, we also have* $\mathcal{Q}^\lambda|_{nsq_0} = \mathcal{P}^\lambda|_{nsq_0}$
- *Or* $nsq_0 \geq n\lambda$ *and* $ns(\lambda+1-t-q_0) \geq n\lambda$. *In which case, we seek if there is a* ${q'}_0$ *such that* $\mathcal{Q}^\lambda|_{nsq_0} = \mathcal{Q}^\lambda|_{n\lambda+{q'}_0}$, *meaning such that* $nsq_0 = n\lambda + {q'}_0$ *and* $ns(\lambda+1-t-q_0) = n(\lambda+s) - t' - q'_0$

  $$\begin{aligned} nsq_0 &= n\lambda + {q'}_0 \\ \Leftrightarrow {q'}_0 &= n(sq_0 - \lambda) \end{aligned}$$

  then

  $$\begin{aligned} n(\lambda+s) - t' - q'_0 &= n(\lambda+s) - t' - n(sq_0-\lambda) \\ &= n(\lambda + s - sq_0 + \lambda) - t' \\ &= n(2\lambda + s - sq_0) - t' \\ &= ns(\lambda + 1 - \frac{t'}{ns} - q_0) \end{aligned}$$

  which means that for $t' = nst$, *from equation 6.6 on the previous page and property 6 on page 124,* $\mathcal{Q}^\lambda|_{nsq_0} = \mathcal{P}^\lambda|_{nsq_0}$. *Note that as* t'' *is the minimal value such that* $N(\{\mathbf{q}\}_\mathbf{r}^\lambda)$ *is a* $(t'', ns, 2)$*-net, if* $t'' \leq nst$, *we have that* $N(\{\mathbf{q}\}_\mathbf{r}^\lambda)$ *is a* $(t', ns, 2)$ *with* $t' = nst$.

Therefore from Eq (6.5 on page 124), Π *preserves the net properties of* $\mathcal{Q}^\lambda$ *and thus if* $\mathcal{Q}^\lambda$ *is a* $(t, \lambda+1, 2)$*-net in base* K^s, $\Pi(\mathcal{Q}^\lambda) = \mathcal{P}^\lambda$ *is a* $(t, \lambda+1, 2)$*-net in base* K^2.

6.3.3 The permutation Ψ is $(t, \lambda+1, 2)$ preserving in base K^2

Lemma 4 *For* $s = 2$, *if* $\mathcal{S}^\lambda$ *is a* $(t, \lambda+1, 2)$*-net in base* K^2, $\mathcal{Q}^\lambda$ *is a* $(t, \lambda+1, 2)$*-net in base* K^2.

Proof 3 $\mathcal{Q}^\lambda$ *is a* $(t, \lambda+1, s)$ *if* $\forall (x, y) \in \mathcal{S}^\lambda$ *and* $\forall (x', y') \in \mathcal{Q}^\lambda$ *we have*

$$(x|_{nsa_0}, y|_{ns(\lambda+1-t-q_0)}) = (x'|_{nsa_0}, y'|_{ns(\lambda+1-t-q_0)}), \tag{6.9}$$

$\forall q_0 \in \mathbb{N}$ *and* $0 \leq q_0 \leq \lambda + 1$.

From Eq (6.3 on page 120), we have

$$\mathcal{Q}^\lambda := \bigcup_\mathbf{r} (\{\mathbf{s}\}_\mathbf{r}^\lambda \oplus \mathbf{v}_\mathbf{r}^\lambda) \tag{6.10}$$

For simplicity, we will extend the xor operator $\oplus$ *to indexed sets, by defining that for two indexed sets* P, P'*, defined at a subdivision level* λ*, with* $\mathbf{p} \in P$*, and* $\mathbf{p}' \in P'$ *with* $Card(P) = Card(P')$*,*

$$P \oplus P' = \bigcup \mathbf{p}^{(i)} \oplus \mathbf{p}'^{(i)}$$

where $\mathbf{p}^{(i)}$ *and* $\mathbf{p}'^{(i)}$ *belong to the same tile* $T_{\mathbf{r}}^{\lambda}$*.*

Furthermore, we will define the operator $\phi_{\lambda'}^{\lambda}$*. This operator takes as input a stratified point set at a level* $\lambda' \leq \lambda$*, and duplicates the vector inside each tile* $T_{\mathbf{r}}^{\lambda'}$*, in order to repeat this vector inside all the subtiles* $T_{\mathbf{r}}^{\lambda}$ *of* $T_{\mathbf{r}}^{\lambda'}$*.*

More formally, we define $\phi_{\lambda'}^{\lambda}$ *so that for two ordered sets,* $\mathcal{S}^{\lambda}, \mathcal{S}^{\lambda'}$*, where* $Card(\mathcal{S}^{\lambda}) = K^{xs} Card(\mathcal{S}^{\lambda'})$*,* $x \in \mathbb{N}$*, and with* $\mathbf{s} \in \mathcal{S}^{\lambda}$*,* $\mathbf{s}' \in \mathcal{S}^{\lambda-1}$*,*

$$\mathcal{S}^{\lambda} \oplus \phi^{\lambda}(\mathcal{S}^{\lambda-1}) = \bigcup_{\mathbf{r}} (\{\mathbf{s}\}_{\mathbf{r}}^{\lambda} \oplus \{\mathbf{s}'\}_{\mathbf{r}}^{\lambda}) \tag{6.11}$$

where as $\mathcal{S}^{\lambda-1}$ *is stratified at level* $\lambda - 1$*,* $\{\mathbf{s}'\}_{\mathbf{r}}^{\lambda}$ *is a singleton.*

This operator will let us denote easily the xoring between two point sets defined at different subdivision level (thus with different cardinalities).

From here, we can rewrite Equation 6.10 on the preceding page as

$$\mathcal{Q}^{\lambda} := \mathcal{S}^{\lambda} \oplus \phi_{\lambda-1}^{\lambda} \mathcal{V}^{\lambda-1}, \tag{6.12}$$

since the set $\mathcal{S}^{\lambda}$ *contains* $K^{s(\lambda+1)}$ *samples. From Equation 6.2 on page 120, the set* $\mathcal{V}^{\lambda-1}$ *contains* $K^{s\lambda}$ *elements, one for each tile* $T_{\mathbf{r}}^{\lambda}$*.*

With those new definitions, we denote

$$\pi^{\lambda-1} = \mathcal{V}^{\lambda-1} \oplus \phi(\mathcal{V}^{\lambda-2}),$$

we can thus derive from $\mathcal{V}^{\lambda-1}$*:*

$$\begin{aligned}
\pi^{\lambda-1} &= \mathcal{V}^{\lambda-1} \oplus \phi_{\lambda-2}^{\lambda-1}(\mathcal{V}^{\lambda-2}) \\
\Leftrightarrow \mathcal{V}^{\lambda-1} &= \pi^{\lambda-1} \oplus \phi_{\lambda-2}^{\lambda-1}(\mathcal{V}^{\lambda-2}) \\
\Leftrightarrow \mathcal{V}^{\lambda-1} &= \pi^{\lambda-1} \oplus \phi_{\lambda-2}^{\lambda-1}(\pi^{\lambda-2}) \oplus \phi_{\lambda-3}^{\lambda-1}(\mathcal{V}^{\lambda-3}) \\
\Leftrightarrow \mathcal{V}^{\lambda-1} &= \oplus_{i=0}^{\lambda-1} \phi_i^{\lambda-1}(\pi^i)
\end{aligned}$$

with $\mathcal{V}^{-1} = 0$*.*

Therefore, $\mathcal{Q}^\lambda$ is made from the set $\mathcal{S}^\lambda$ through successive $\oplus$ with $\phi(\pi^i)$. From here, we derive $\forall \lambda', 0 < \lambda' \leq \lambda$,

$$
\begin{aligned}
&\mathcal{S}^{\lambda'-1} \oplus \mathcal{P}^{\lambda'-1} = \mathcal{V}^{\lambda'-1} \\
\Leftrightarrow &\mathcal{S}^{\lambda'-1} \oplus \phi^{\lambda'-1}_{\lambda'-2}(\mathcal{V}^{\lambda'-2}) \oplus \mathcal{P}^{\lambda'-1} = \mathcal{V}^{\lambda'-1} \oplus \phi^{\lambda'-1}_{\lambda'-2}(\mathcal{V}^{\lambda'-2}) \\
\Leftrightarrow &\mathcal{Q}^{\lambda'-1} \oplus \mathcal{P}^{\lambda'-1} = \pi^{\lambda'-1} \\
\Leftrightarrow &\mathcal{P}^{\lambda'-1} = \pi^{\lambda'-1} \oplus \mathcal{Q}^{\lambda'-1} \\
\Leftrightarrow &\Pi(\mathcal{Q}^{\lambda'-1}) = \pi^{\lambda'-1} \oplus \mathcal{Q}^{\lambda'-1}
\end{aligned}
$$

Therefore, xoring $\mathcal{Q}^{\lambda'}$ with $\pi^{\lambda'}$ is similar to applying the permutation Π that created $\mathcal{P}^{\lambda'}$ from $\mathcal{Q}^{\lambda'}$.

From Lemma 3 on page 125, if $(x, y) \in \mathcal{S}^\lambda$ and $(x', y') \in \mathcal{S}^\lambda \oplus \phi(\pi^{\lambda'})$,

$$(x|_{nsq'_0}, y|_{ns(\lambda'+1-t'-q'_0)}) = (x'|_{nsq'_0}, y'|_{ns(\lambda'+1-t'-q'_0)}) \tag{6.13}$$

Furthermore, since $\pi^{\lambda'} = \mathcal{Q}^{\lambda'} \oplus \phi(\mathcal{P}^{\lambda'})$, when written in binary we have for $\mathbf{p} \in \pi^{\lambda'-1}$

$$\mathbf{p}_d = \sum_{i=0}^{ns(\lambda'+1)} p_d^i 2^{-1-i}$$

where $\mathbf{p}_d$ is the d^{th} coordinate of $\mathbf{p}$ and p_d^i is the i^{th} bit of $\mathbf{p}_d$. We note that $\forall i > ns(\lambda'), p_d^i = 0$.

Geometrically, xoring with $\pi^{\lambda'}$ means that we move all samples that belong to an elementary interval $\mathcal{J}^{q'_0}$ to another $\mathcal{J}^{q'_0}$. However, since all digits of $\mathbf{p} \in \pi^{\lambda'}$ are 0 when $i > ns(\lambda')$, all samples in intervals that are smaller than $\frac{1}{2^{ns(\lambda')}}$ on every dimension, meaning all samples from intervals such that $nsq_0 > nsq'_0$ and $ns(\lambda + 1 - q_0) > ns(\lambda' + 1 - q'_0)$, are preserved.

$$(x|_{nsq_0}, y|_{ns(\lambda+1-t-q_0)}) = (x'|_{nsq_0}, y'|_{ns(\lambda+1-t-q_0)}) \tag{6.14}$$

Furthermore, since we xor with 0 for every digit $> ns(\lambda')$, similarly to the previous proof, we can always reduce the size of the interval on a single dimension, leading to the following equations

$$(x|_{nsq'_0}, y|_{ns(\lambda+1)}) = (x'|_{nsq'_0}, y'|_{ns(\lambda+1)}) \tag{6.15}$$

$$(x|_{ns(\lambda+1)}, y|_{ns(\lambda'+1-t'-q'_0)}) = (x'|_{ns(\lambda+1)}, y'|_{ns(\lambda'+1-t'-q'_0)}) \tag{6.16}$$

$\forall q_0 \in \mathbb{N}, 0 \leq q_0 \leq ns(lvl + 1) - n(\lambda' + 1)$.

From here we have several possibilities for q_0

- *$q_0 = q'_0$, then equation 6.13 equals equation 6.9 on page 126.*

- *$nsq'_0 > nsq_0$ and $ns(\lambda' + 1 - t' - q'_0) > ns(\lambda + 1 - t - q_0)$ then from property 6 on page 124, equation 6.9 on page 126 is implied by 6.13 on the facing page.*
- *$nsq'_0 < nsq_0$ and $ns(\lambda' + 1 - t' - q'_0) < ns(\lambda + 1 - t - q_0)$ then 6.14 on the preceding page* applies.
- *$nsq'_0 > nsq_0$ and $ns(\lambda' + 1 - t' - q'_0) < ns(\lambda + 1 - t - q_0)$ (or the opposite), equation 6.15 on the facing page (or 6.16 on the preceding page) can be used jointly with Property 6 on page 124 to imply 6.9 on page 126*

6.3.4 $\mathcal{P}^\lambda$ is a $(t, \lambda + 1, s)$-net in base K^s

Lemma 5 *For $s = 2$, if $\mathcal{S}^\lambda$ is a $(t, \lambda + 1, 2)$-net in base K^2, $\mathcal{P}^\lambda$ is a $(t, \lambda + 1, 2)$-net in base K^2.*

Proof 4 *$\mathcal{P}^\lambda$ is made from $\mathcal{S}^\lambda$ by successively applying two permutations. First, the Ψ permutation is applied, creating a point set $\mathcal{Q}^\lambda$, which is a $(t, \lambda + 1, 2)$-net in base K^2 from Lemma 4 on page 126. Then, we create $\mathcal{P}^\lambda$ by applying Π to $\mathcal{Q}^\lambda$, and as from Lemma 3 on page 125, Π preserves the net properties of $\mathcal{Q}^\lambda$, $\mathcal{P}^\lambda$ is a $(t, \lambda + 1, 2)$-net in base K^2.*

6.3.5 The samples from $\mathcal{P}^{\lambda-1}$ are also into $\mathcal{P}^\lambda$

Lemma 6 *$\mathcal{P}^{\lambda-1} \subset \mathcal{P}^\lambda$*

Proof 5 *Since $\mathcal{S}^{\lambda-1}$ is a $(t, \lambda + 1, s)$-net in base K^D from property 1 on page 118 with $t < ns$ and both $\mathcal{Q}^{\lambda-1}$ and $\mathcal{P}^{\lambda-1}$ are $(t, \lambda + 1, s)$-nets in base K^D from Lemmas 4 and 3, for $t = 0$, each of those sets contains one and only one sample inside each tile $T^\lambda_{\mathbf{r}}$.*

$$\mathcal{S}^{\lambda-1} = \bigcup_{\mathbf{r}} \{\mathbf{s}\}^{\lambda-1,\lambda}_{\mathbf{r}} \tag{6.17}$$

$$\mathcal{Q}^{\lambda-1} = \bigcup_{\mathbf{r}} \{\mathbf{q}\}^{\lambda-1,\lambda}_{\mathbf{r}} \tag{6.18}$$

$$\mathcal{P}^{\lambda-1} = \bigcup_{\mathbf{r}} \{\mathbf{p}\}^{\lambda-1,\lambda}_{\mathbf{r}} \tag{6.19}$$

For the same reason, we have that $\mathcal{S}^\lambda$ contains K^s samples inside each tile $T^\lambda_{\mathbf{r}}$. As $\mathcal{S}^{\lambda-1} \subset \mathcal{S}^\lambda$ from [Sob67], we have that $\{\mathbf{s}\}^{\lambda-1,\lambda}_{\mathbf{r}} \in \mathcal{S}^\lambda$.

From Eq (6.3 on page 120), we have

$$\mathcal{Q}^{\lambda} = \bigcup_{\mathbf{r}} \{\mathbf{s}\}_{\mathbf{r}}^{\lambda} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda} \tag{6.20}$$

where $\mathbf{v}_{\mathbf{r}}^{\lambda}$ is a singleton defined as

$$\mathbf{v}_{\mathbf{r}}^{\lambda} = \{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda} \oplus \{\mathbf{p}\}_{\mathbf{r}'}^{\lambda-1,\lambda} \tag{6.21}$$

with $\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}$ the i^{th} sample in $\mathcal{S}^{\lambda-1}$ and $\{\mathbf{p}\}_{\mathbf{r}'}^{\lambda-1,\lambda}$ the i^{th} sample in $\mathcal{P}^{\lambda-1}$.

Therefore

$$\begin{aligned}
\{\mathbf{s}\}_{\mathbf{r}}^{\lambda} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda} &= (\{\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}/\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}\} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda}) \cup (\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda}) \\
&= (\{\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}/\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}\} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda}) \cup (\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda} \oplus \{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda} \oplus \{\mathbf{p}\}_{\mathbf{r}'}^{\lambda-1,\lambda}) \\
&= (\{\{\mathbf{s}\}_{\mathbf{r}}^{\lambda}/\{\mathbf{s}\}_{\mathbf{r}}^{\lambda-1,\lambda}\} \oplus \mathbf{v}_{\mathbf{r}}^{\lambda}) \cup (\{\mathbf{p}\}_{\mathbf{r}'}^{\lambda-1,\lambda})
\end{aligned}$$

Therefore, $\mathcal{P}^{\lambda-1} \subset \mathcal{Q}^{\lambda}$.

Then, from equation 6.4 on page 122 we have

$$\mathcal{P}^{\lambda} := \bigcup_{\mathbf{r}} N^{-1}(\Pi_{\mathbf{r}}^{\lambda}(N(\{\mathbf{q}\}_{\mathbf{r}}^{\lambda}))) \tag{6.22}$$

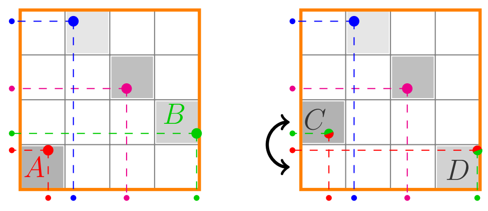
and by property 4 on page 122, we know that Π preserves the position of the samples from $\mathcal{P}^{\lambda-1}$. Therefore, is $\mathcal{P}^{\lambda-1} \subset \mathcal{Q}^{\lambda}$, $\mathcal{P}^{\lambda-1} \subset \mathcal{P}^{\lambda}$.

6.3.6 Our local permutations are admissible permutations

In this section, we demonstrate that the set of permutation $\{\pi_r^{\lambda}\}$ built by our method (See Section 6.1.2 on page 110) is a subset of the admissible set of permutation Π.

Lemma 7 *The local permutations $\{\pi_r^{\lambda}\}$ as defined in Section 4.1 of the main article define an admissible Π.*

Proof 6 *We already explained how our permutation can preserve the position of a given marked sample (constraint (i) leading to Lemma 6 on the previous page). We give here more details on how we exchange the trailing bits, to demonstrate that we preserve the net properties of the initial set and its 1-D projections.*



Initial set *Our permutation*

When we remove the trailing bits of a sample, i.e. *when we quantize this sample to the corner of its elementary interval T_r^λ (gray cells in above figure), we do not change the LD net properties at level λ. We then apply Owen's scrambling on this quantized set, which is also net-preserving ([Owe95], Proposition 1). And finally, when we reapply the trailing bits, even if we exchange those bits between the samples, those bits are not significant enough to take a sample out of its elementary interval. As an example, if Owen's scrambling at level λ swaps rows containing $A := (x,y)$ and $B := (x,y)$, we construct the points $C := (x,y)$ and $D := (x,y)$. This exchange of trailing bits is then net-preserving, and therefore our permutation is net-preserving.*

Finally, if the initial set is a (t, k, s)-net, when we quantize it on k digits, all combinations of bits are present. Since Owen's scrambling is net-preserving, after scrambling, all combination of bits will still be represented. Then, simply by reapplying the trailing bits with the proper initial bits, we are guaranteed to preserve the 1-D projections of samples.

So our set $\{\pi_r^\lambda\}$ is a subset of the admissible set of permutation.

6.4 Conclusion

Following those proofs, we can fairly say that our new method scrambles Sobol sets to generate 2-D projections with proven Blue Noise and Low discrepancy property. By independently scrambling projections, we also allow the generation of s-D point sets with improved discrepancy and spectrum, as shown in Chapter 7 on page 133. Furthermore, our method is sequential, which would let us iteratively refine the approximation of an integrand if needed.

This projection-wise scrambling is not so much of a drawback for rendering as we solve a recursive set of independent integrands. However, only some particular pairs of Sobol indices can be scrambled with our method, which limits its usage to low dimensional sets ($s < 10$). Note that this can be alleviated with our sequential Owen' method (Section 4.2 on page 79), but the spectrum will present a lower quality. Another limitation of our method (which is inherent to the combination of Blue Noise and Low Discrepancy), is that it presents a proper spectrum and discrepancy only for sets of size $K^{2n}, n \in \mathbb{N}$ (this is further discussed in Chapter 7 on page 133).

Still, despite the amount of possible future work, our scrambling is the first known solution to create sequential sampling patterns with low discrepancy and improved spectrum in higher dimensions.

7 Experiments

In this section, we present the results of all experiments we performed when characterizing samplers and validating our methods. All of our evaluations were performed on uniform sampling patterns, the settings for each particular experiments are detailed within the sub sections.

We start by presenting graphs measuring the behaviour of the discrepancy and of the integration error. Those measures are presented for our methods and are thoroughly compared with the state of the art. Then, we present reconstructions and renderings obtained from our methods and some state of the art methods. Unfortunately, due to the complexity of modern rendering systems, this analysis is less thorough but is still very informative. Our final section presents a table of timings for the main samplers.

All those experiments were computed using tools from the UTK Framework [Utk]. For all state of the art samplers, we used, when available, their implementation within the UTK framework. For other samplers, we used code provided by the authors. Note that for all non-trivial samplers, their implementation within the UTK Framework is a copy of the authors code that was merely adapted to the UTK API. We compare those previously existing samplers to the new methods proposed in Chapter 5 on page 93 and Chapter 6 on page 107, also implemented in the UTK API.

7.1 Discrepancy

In this section, we compare the discrepancy of many samplers. Our main interest is to measure how their discrepancy evolves as the number of sample increases, the faster it converges to 0, the better. We start by presenting the l_∞ star discrepancy measure, for 2-D samplers. Then, we will present the l_2 star discrepancy for the same 2-D samplers, and for some 4-D sampling patterns.

7.1.1 l_∞ star discrepancy

In this subsection we present the l_∞ star discrepancy computed using the exact $O(n^s)$ algorithm described in Section 2.1.1 on page 17. The following graphs present the evolution of this discrepancy for many samplers. We first see how Low discrepancy samplers behave (Fig. 7.1 on page 135), then similarly for Blue Noise samplers (Fig. 7.2 on page 135), and finally for samplers that are neither Blue Noise nor Low Discrepancy (Fig. 7.3 on page 136). Then, the discrepancies of the main samplers will be compared to the discrepancy of our new methods

(Fig. 7.4 on page 136). Note that all those measures were done with 2-D samplers. On the following graphs, we also plotted the theoretical discrepancies for the 2-D Whitenoise sampler, $O(1/n)$, and for 2-D Low Discrepancy samplers $O(\log(n)/n)$.

The first graph (Fig. 7.1 on the facing page) presents the discrepancy of various Low Discrepancy samplers. As expected, all curves show the same convergence speed, following the theoretical curve in $O(\log(n)/n)$. It is interesting to note how the discrepancy of the Faure [Fau82] and Sobol [Sob67] samplers increases and decreases periodically. This is due to the fact that those sequences generate digital nets in base 2. Thus, any number of samples that is not a power of two cannot fill properly the net as there are not enough samples with respect to the number of congruent boxes (see Section 3.3.2 on page 59). For all samplers defined in a base b, we denote *octaves* the sets presenting b^k samples, $k \in \mathbb{N}$. Note that in base 2, the Faure and Niederreiter sequences [Nie88] are similar to the Sobol sequence (with a different ordering of samples). Therefore, we plotted the Neiderreiter sequence in base $b = 3$ here. Also note that in further graphs, we may no longer show the discrepancy outside the octaves.

Now if you look at the l_∞ star discrepancy of Blue Noise samplers (Fig. 7.2 on the facing page), it can be seen that for many such samplers, the discrepancy is close to the Whitenoise theoretical curve. Note that for such samplers, as they are stochastic, the discrepancy was presented a the maximal discrepancy over around 20 realisations. However, for some methods such as BNOT [DG+12], CVT [Du+99] or CapCVT [Che+12], the computational time is really expensive and, for higher number of samples, we sometimes are not able to generate more than one or two sets, hence the increase in variance for the discrepancy measures. In the case of CVT and CapCVT, it can be seen that passed 10000 samples, the discrepancy seems to stop decreasing as the number of samples increases. At this time, we are still not sure to why we observe this behaviour. There are two possible explanations to this phenomenon, either the implementation we have for such samplers (provided by the authors) is slightly flawed, or we are not configuring it properly. If this is not the case, this could be also explained by a unperceived shifting of the points, leading to visually perfect sets, but that is sufficient to affect the discrepancy (similarly to what happens with the sampler from [Fat11], as mentioned in Section 3.2.2 on page 44)

Then, the third graph presents the discrepancy of other samplers. It can be seen that as expected, the Whitenoise sampler has a discrepancy in $O(1/n)$, similarly to the discrepancy of regular and hexagonal grids (due to samples alignment). The N-Rooks sampler has an interesting behaviour, as its discrepancy seems to increase as the number of samples increase. That is most likely because N-Rooks shuffles a canonical arrangement to preserve its 1-D projections, and as the number of samples increases, samples are getting closer and closer to each other in those projections. Furthermore, in the 2-D plane, there is nothing that prevents samples from clumping. Thus, the discrepancy gets closer and closer from the discrepancy of the Whitenoise sampler as the number of samples increase. Another interesting point on this graph is the discrepancy of stratified sets, that is surprisingly good. Thanks to the stratification of samples, clumping is partly alleviated and aligning samples are less likely. Hence the improvements in the discrepancy.

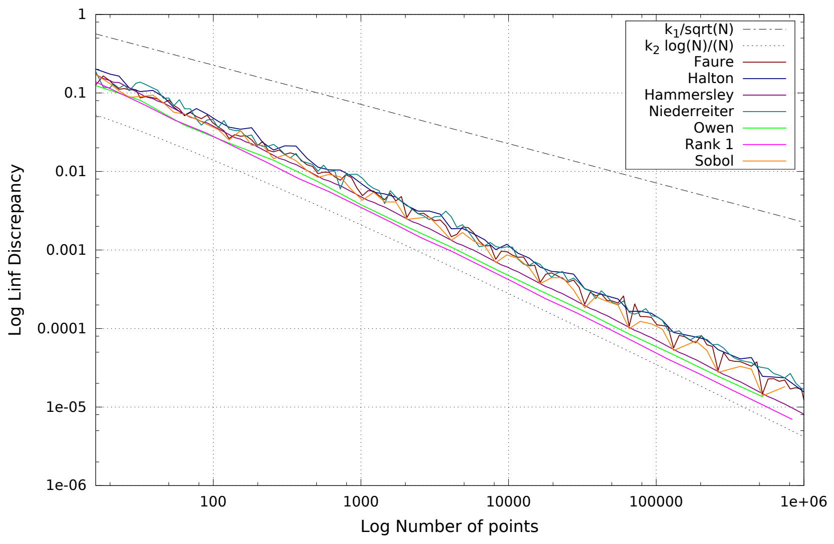


Fig. 7.1.: l_∞ Star Discrepancy of various 2-D Low Discrepancy samplers.

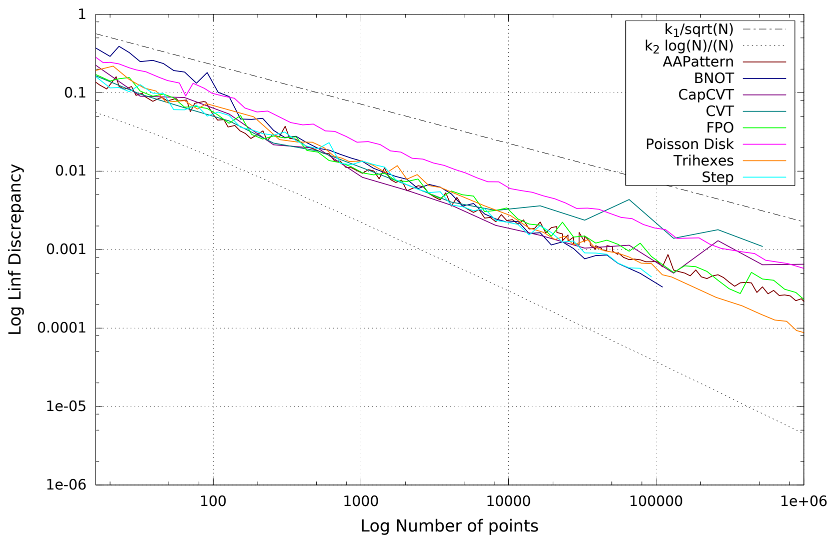


Fig. 7.2.: l_∞ Star Discrepancy of various 2-D Blue Noise samplers.

Finally, our last graph presents how our new samplers (described Chapter 5 on page 93 and Chapter 6 on page 107) behave regarding state of the art samplers. It can be seen that, as expected, they experimentally behave as Low Discrepancy Samplers. They thus present a better discrepancy than Blue Noise and Historical samplers.

However, Graph 7.5 on page 137 presents that, for our second method, the discrepancy is very degraded between the octaves. The size of such octaves depends on the subdivision factor K of our tile based system. A point set is therefore low discrepancy only if its size n verifies $n = a^{2^K}, a \in \mathbb{N}$. Therefore, the higher the K, the more difficult to obtain a set that is Low discrepancy. On the other hand, the higher the K, the more Blue Noise our sets can be. This will be discussed further in the Chapter 8 on page 165.

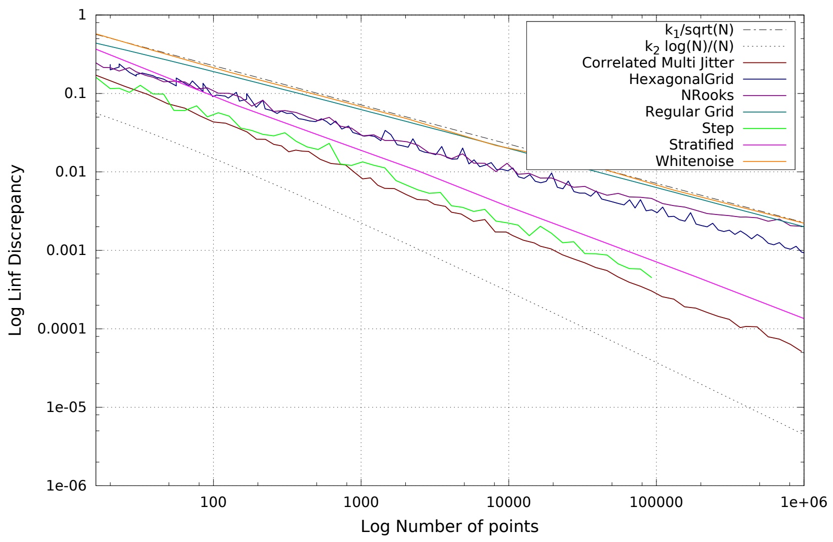


Fig. 7.3.: l_∞ Star Discrepancy of various 2-D samplers.

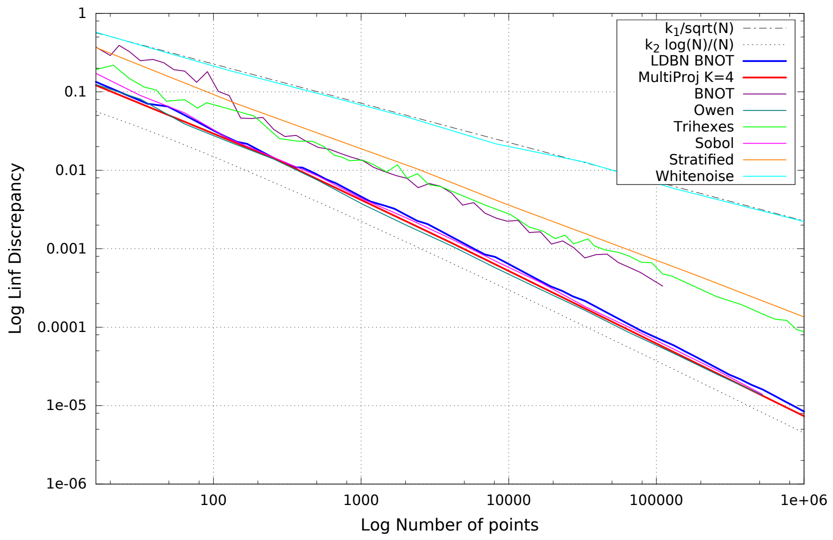


Fig. 7.4.: Comparing the l_∞ Star Discrepancy of our samplers with reference 2-D samplers.

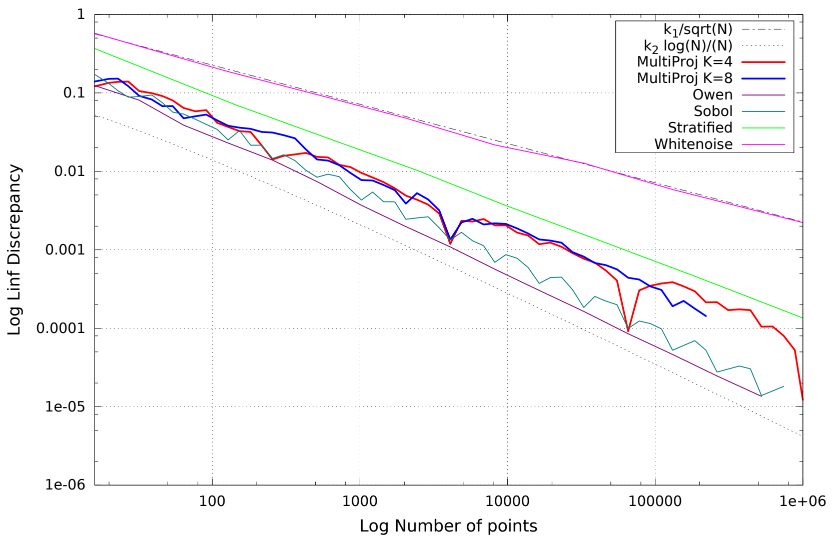


Fig. 7.5.: Comparing the l_∞ Star Discrepancy of our projection scrambling, outside the octaves, with reference 2-D samplers.

7.1.2 l_2 star discrepancy

As the l_∞ star discrepancy is not tractable for large sets in more than 2-D, we need another measure to compare how the discrepancy of a sampler evolves as the dimension increases. To do so, we used the l_2 star discrepancy measure, as described Section 2.1.2 on page 19. This discrepancy was computed from the *Generalized L2 Discrepancy* tool from UTK. This tool takes as input a pointset P_n, and computes its generalized l_2 star discrepancy using the following formula (previously given Equation 2.13)

$$(\mathcal{D}_2(P_n))^2 := \left(\frac{4}{3}\right)^s - \frac{2}{n}\sum_{i=1}^{n}\prod_{j=1}^{s}\frac{3-\left(x_j^{(i)}\right)^2}{2} + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\prod_{j=1}^{s}\left[2-\max\left(x_j^{(i)}, x_j^{(i')}\right)\right]. \qquad (7.1)$$

This lets us compute this discrepancy with a $O(sn^2)$ algorithm.

Similarly to the l_∞ star discrepancy, we first present the evolution of the l_2 star discrepancy for state of the art Low discrepancy samplers (Fig. 7.6 on the next page), then for Blue Noise samplers (Fig. 7.7 on the following page), and finally for other samplers (Fig. 7.8 on page 139). Then, the discrepancies of the main samplers will be compared to the discrepancy of our new methods (Fig. 7.9 on page 139).

On all those graphs, we presented, for comparison, the theoretical convergence speeds of the l_∞ star discrepancy. Note that we do not expect the convergence speeds of the l_2 star discrepancies of samplers to follow those curves. However, it can still be noted that the measurements are quite close from those curves.

Graph 7.6 on the next page presents the l_2 star discrepancy of low discrepancy samplers. It shows that the convergence speed is still the same for all those samplers, and we see the same

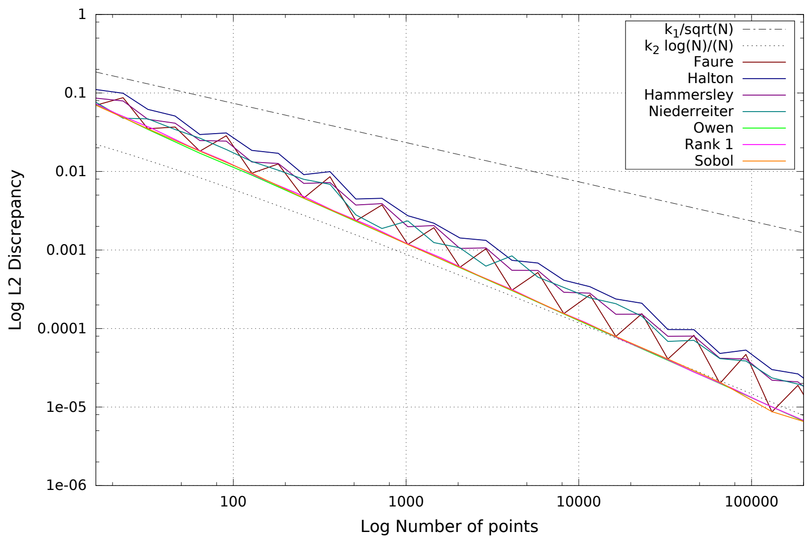


Fig. 7.6.: l_2 Star Discrepancy of various 2-D Low Discrepancy samplers.

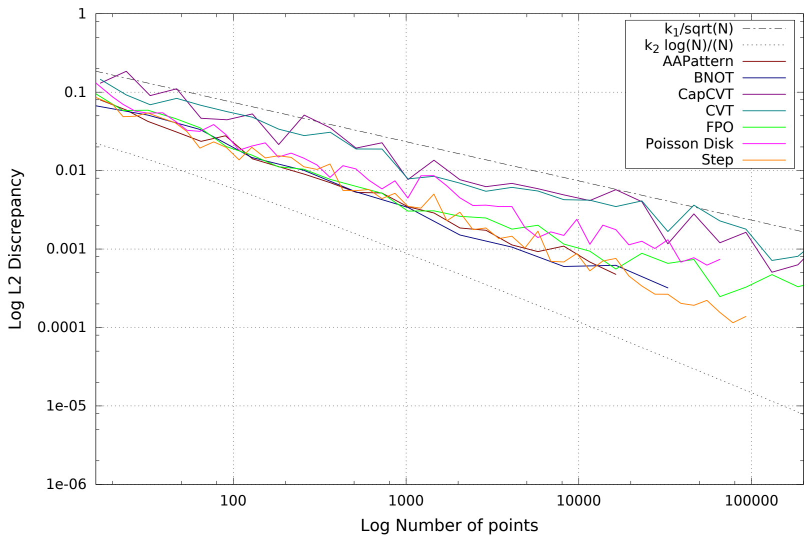


Fig. 7.7.: l_2 Star Discrepancy of various 2-D Blue Noise samplers.

periodical increase in the discrepancy of Sobol [Sob67] and Faure [Fau82] in between the octaves, as was observed for the l_∞ star discrepancy.

For Blue Noise samplers, presented Graph 7.7, we note that we no longer observe the weird behaviours that we could see with the l_∞ star discrepancy. This is most likely due to the fact that the l_∞ star discrepancy keeps the maximal difference between box area and fraction of samples in this box (Section 2.1.1 on page 17). Thus, this discrepancy is highly sensitive to outliers, if a single measure fails, the whole discrepancy behaves abnormally. The l_2 star discrepancy on the other hand averages the differences for all boxes. Outliers are thus invisible in the final measure.

The last two graphs, presenting the l_2 star discrepancy of other samplers (Graph 7.8 on the facing page), and comparing our methods to state of the art methods (Graph 7.9 on the next page), lead to roughly the same conclusions as the ones from l_∞ star discrepancy graphs.

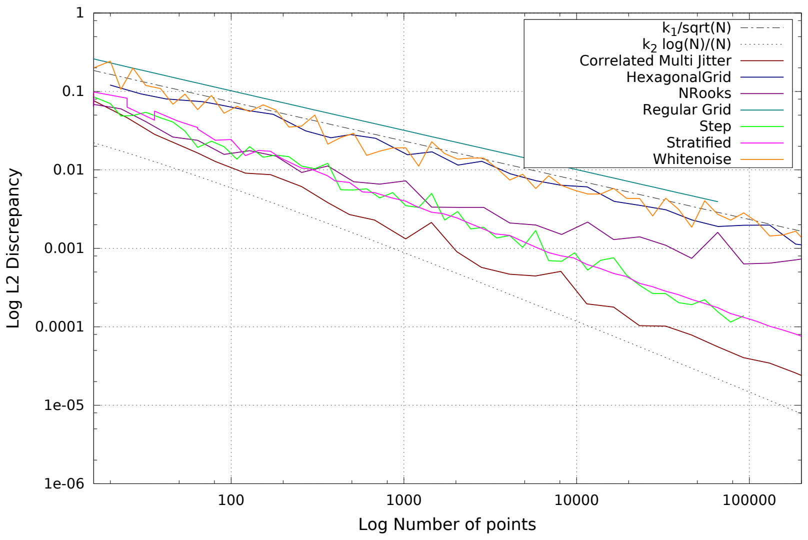


Fig. 7.8.: l_2 Star Discrepancy of various 2-D samplers.

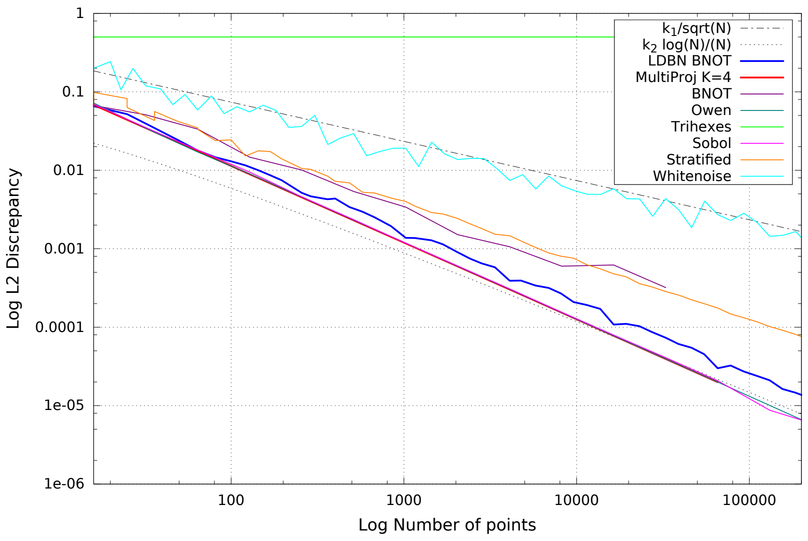


Fig. 7.9.: Comparing the l_2 Star Discrepancy of our samplers with reference 2-D samplers.

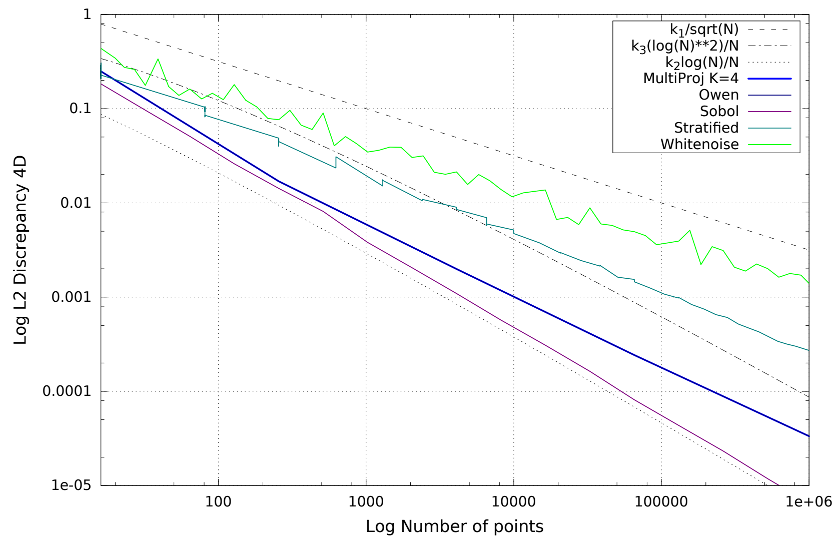


Fig. 7.10.: Comparing the l_2 Star Discrepancy of our Multi-dimensional sampler (Chapter 6) with reference 4-D samplers.

Contrary to the analysis of 2-D point sets using the l_2 star discrepancy, for higher dimensional sets, we cannot present measurements over Blue Noise samplers as those samplers are of very limited dimensionality. Figure 7.10 thus only present the evolution of the l_2 star discrepancy of state of the art 4-D samplers compared with the 4-D discrepancy of our higher-dimensional method (Chapter 6).

We see in Figure 7.10 that, as expected, our projection scrambling does not preserve the full 4-D discrepancy. However, its discrepancy is still better than the discrepancy of a stratified sampler for lower number of samples. It seems that each stage of scrambling messes up the 4-D discrepancy of the set a little bit more, and that as the number of samples increases, our set has a discrepancy with a similar behaviour as the stratified sampler.

7.2 Integration error

In this section, we compare how various samplers perform when used for the Monte Carlo estimation of a given integrand.

On the following graphs however, we did not plot the integration error as

$$\Delta = |\mathcal{I}_n - \mathcal{I}|. \tag{7.2}$$

Plotting this function for a single integrand would have led to very noisy curves and would have prevented an easy reading of the graphs. Instead, we used the theorem from [Pil+15] that states that for unbiased stochastic samplers, their integration error is equivalent to the square root of the variance in integration error. Plotting this measure leads to much smoother curves, even for a single integrand. For deterministic samplers, we use the Cranley-Patterson shift (Section 4.1 on page 74) to randomize them, and also plotted their variance. This is possible since this

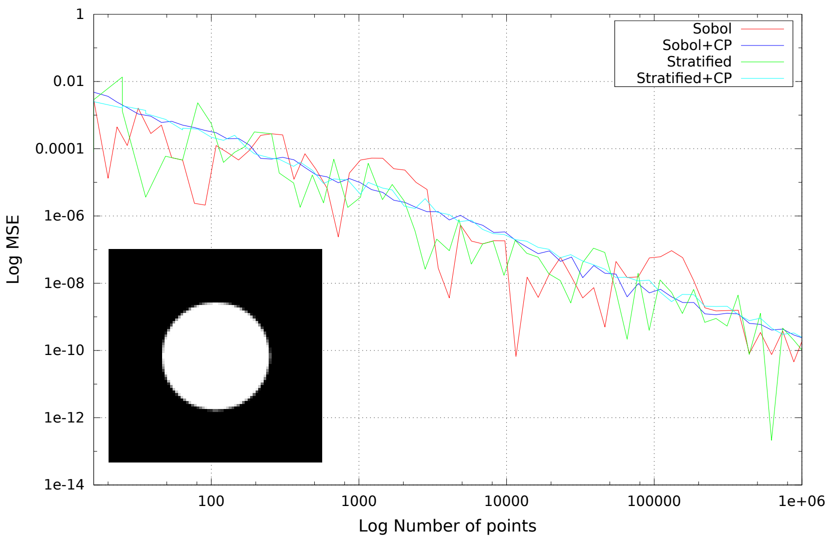


Fig. 7.11.: Integration MSE and Variance measured on a Disk with the Sobol and Stratified samplers, with and without Cranley-Patterson shift.

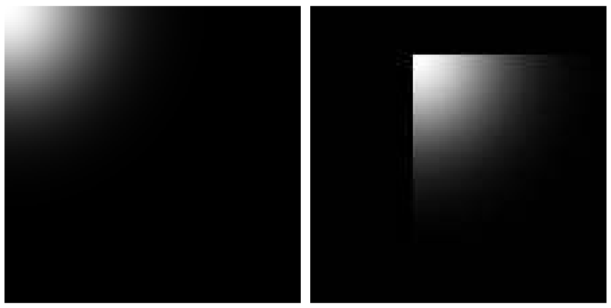


Fig. 7.12.: Cornered Gaussian function without and with shifting. Discontinuities appear on the shifted function that did not exist in the original function.

particular randomization does not affect the Fourier spectrum nor the discrepancy of the set. Graph 7.11 presents a comparisons of the MSE measured when integrating from a single Sobol and a single Stratified sets and the MSE measured from averaging 25 Cranley Patterson shifted realisation of those sets. Those graphs show that, apart from the smoothness of the curves, no notable differences can be observed between the MSE of a set and the averaged MSE of this set with Cranley Patterson shift.

Note however that integrating a single function with a Cranley-Patterson shifting of samples can be seen as integrating a Cranley-Patterson shifted function with a single set of samples. This may, in particular cases, create discontinuities in the function that were not present in the original integrand (Fig. 7.12). Graph 7.13 on the next page presents a comparisons of the MSE measured when integrating a cornered Gaussian function from a single Sobol and a single Stratified set and the MSE measured from averaging several Cranley-Patterson shifted realisation of those sets. Such integrand when shifted creates discontinuities which changes the frequency radial profile of the function. In all those integrands used for our following test, discontinuities were already present in the original integrand and thus, shifting it has no impact on the MSE.

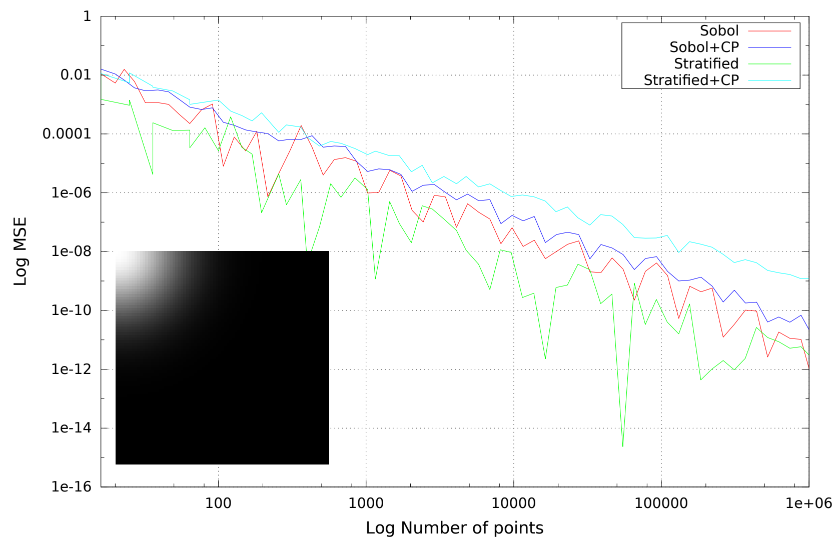


Fig. 7.13.: Integration MSE and Variance measured on a Cornered Gaussian with the Sobol sampler, with and without Cranley-Patterson shift.

We observed that integration errors for functions with higher exponents (in absolute value) in their mean spectral profile are similar for all samplers. This is consistent with the results from [Pil+15], as a higher exponent means that the radial power spectrum from the integrand will be zero almost immediately and thus only the very low frequencies will be present. As a vast amount of samplers try to be as close to 0 as possible in the low frequencies the Fourier spectrum of all those samplers will have a similar impact on the observed error. Therefore, measuring a difference between those samplers for such integrands is difficult. The differences appear when integrating over functions having a mean spectral profile with a higher exponent.

We start by presenting this measured integration error when integrating an analytical 2-D disk and analytical 4-D spheres. Then, we present how samplers perform when integrating over a discretization of a natural scene, given as a HDR image. Finally, we present a new tool (still at an early development stage) to allow analytical integration of complex arrangements, that we hope may, with further work, allow to simulate the spectral properties of natural scenes, but with an analytical representation.

7.2.1 Integration over analytical functions

We start by integrating an analytical 2-D disk defined with

$$f_{Disk}(x) := \begin{cases} \frac{4}{\sqrt{\pi}} & if|x| < 0.25, \\ 0 & otherwise. \end{cases} \tag{7.3}$$

This Disk is an interesting analytical integrand as it presents discontinuities which are very common in rendered scenes. One could also use a Triangle or a Square, but their Power spectra converge faster than the Power spectrum of the Disk, which converges in $O(\rho^{-3})$.

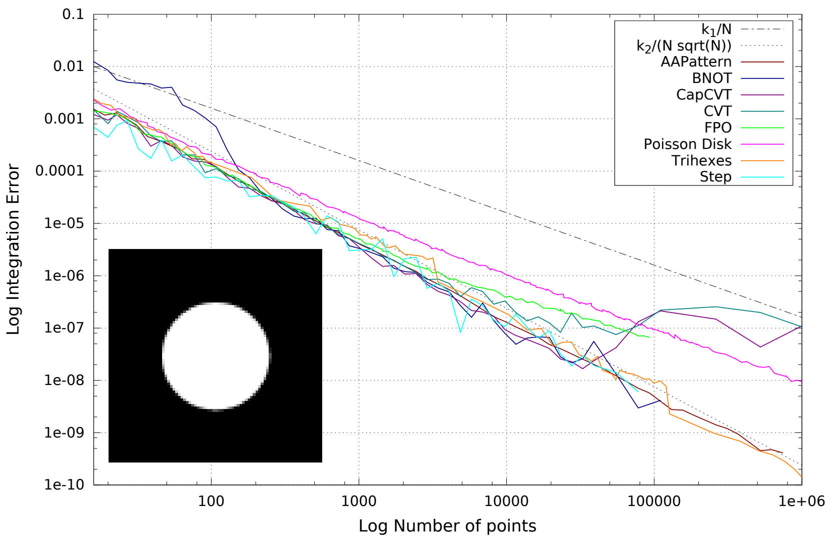


Fig. 7.14.: Variance of Monte Carlo estimators from Blue Noise samplers when integrating an analytical 2-D disk.

We can see on Figure 7.14 that not all Blue Noise samplers are equal for Monte Carlo estimators. Consistently with what was observed for the discrepancy of the CVT [Du+99] and CapCVT [Che+12] samplers, their integration error suddenly increases drastically as the number of samples gets too high. Similarly for the discrepancy, we do not know at this time why this behaviour occurs but we assume it is either due to a misprogramming/misusage of the code (we are using the code from the original authors), or to a very minor shifting of the samples that may not be visible to the eye but that causes the integration error (and discrepancy) to increase drastically. It can also be noted that the Poisson Disk sampler, consistently with the results of [Pil+15], performs quite poorly. Similarly, the FPO sampler has a higher integration variance. This is still unexplained but our hypothesis is that this is due to similar reasons as for the Poisson Disk sampler.

When looking at the integration error of Low discrepancy samplers, Figure 7.15 on the next page, we can note that all samplers performs very well. This is consistent with the Koksma-Hlawka theorem which bounds the convergence of the integration error of 2-D Low discrepancy samplers to $O(\log(n)/n)$.

The graph Figure 7.16 presents the results from other samplers. The behaviour of Whitenoise and stratified samplers are consistent with their theoretical convergence. However, the behaviours of NRooks and of the Hexagonal grid are surprising. For NRooks, this is most likely due to the regular canonical input that, once shuffled, does not allow for a covering of the final set as uniform as a White noise set. For the Hexagonal grid, the high variance observed within the variance values themselves prevents a proper analysis of what happens, but it seems to suffer from a different issue than the NRooks sampler as its discrepancy behaved perfectly well.

Finally, Figure 7.17 on page 145 compare the behaviour of our methods with respect to the main state of the art samplers. It can be seen that our integration error converges as fast as for usual Low discrepancy samplers, which was to be expected as our samplers are Low discrepancy.

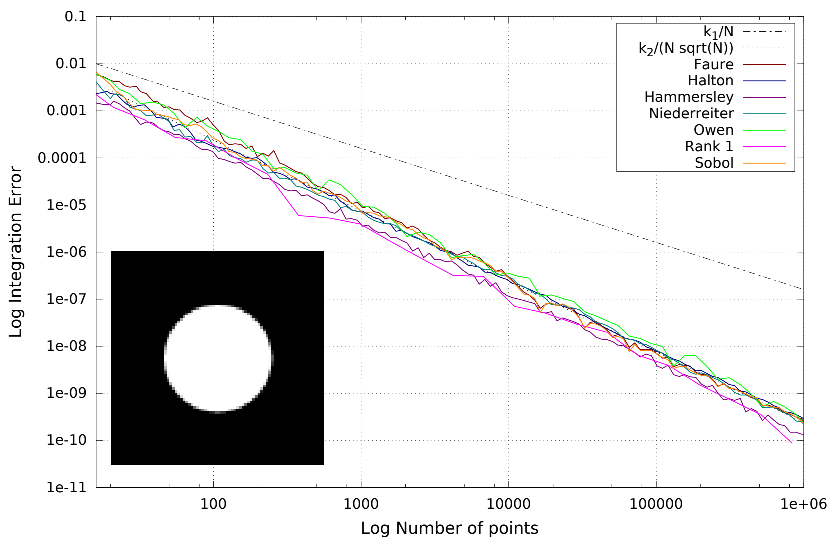


Fig. 7.15.: Variance of Monte Carlo estimators from Low Discrepancy samplers when integrating an analytical 2-D disk.

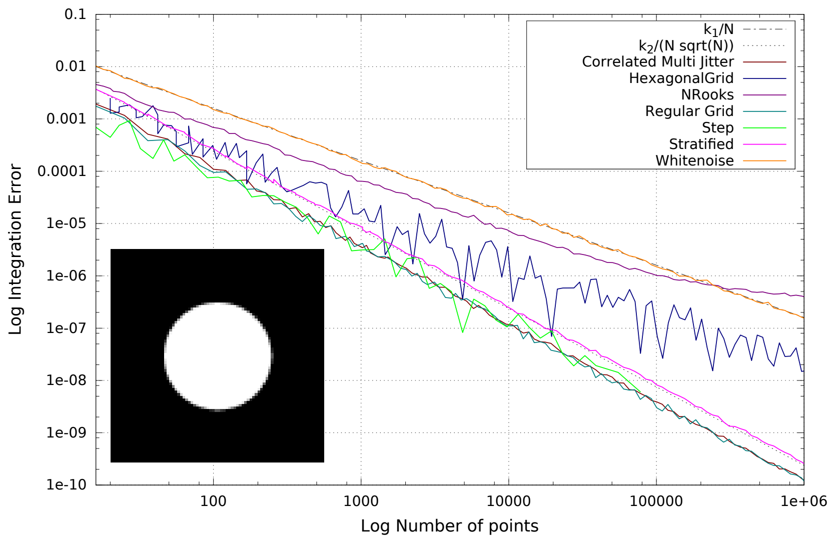


Fig. 7.16.: Variance of Monte Carlo estimators from various samplers when integrating an analytical 2-D disk.

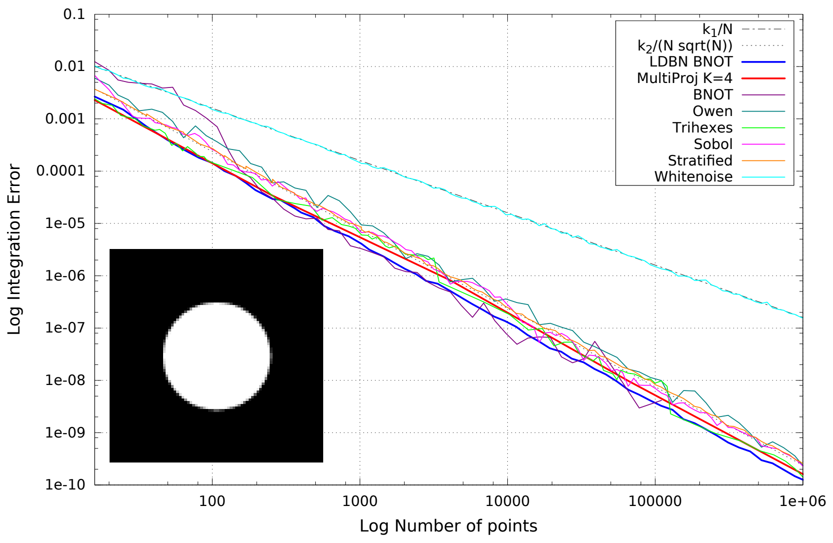


Fig. 7.17.: Comparing the variance of Monte Carlo estimators of our samplers with reference samplers when integrating an analytical 2-D disk.

We also performed measurements for the integration error of a 4-D sphere, with 4-D samplers. The results are given Figure 7.18 on the next page. It can be seen that, despite not being a Low Discrepancy 4-D sampler, our sampler performs very well, as, in the worst case, our set resumes as a 4-D stratified set which has a good performance regarding the integration error.

7.2.2 Integration over discrete natural scenes

In this section, we show our results when integrating over HDR images. We chose two HDR images with very different Fourier spectra profiles. Natural images spectra can be roughly categorized into two categories [TO03]: The ones showing man made scenes (buildings, streets, etc.) and the ones showing natural scenes (forests, grass, etc.). Man made structures have a much more anisotropic spectra and present a mean spectral profile in $O(\rho^{-1.8})$ while natural images are much more isotropic with a mean spectral profile in $O(\rho^{-2})$. The two gray scale HDR images are taken from the sIBL archive with a size of 1600x1600. Figure 7.19 on the following page presents those images along with their Fourier spectrum, radial profiles, and exponent.

The results from this section are similar to the results observed for analytical integrands. They are presented in Figures: 7.21, 7.20, 7.22, 7.23, 7.25 on page 149, 7.24, 7.26 and 7.27 on the following pages. Note that as the exponent of their radial profile is lower, distinction between samplers is made easier.

7.2.3 Integration over analytical arrangements

In this section, we present a new integration test. When integrating from natural HDR images, we only have so many images we can integrate from. Furthermore, those images are already discretized and we do not have any analytical expression for the integrand. Here, our final goal

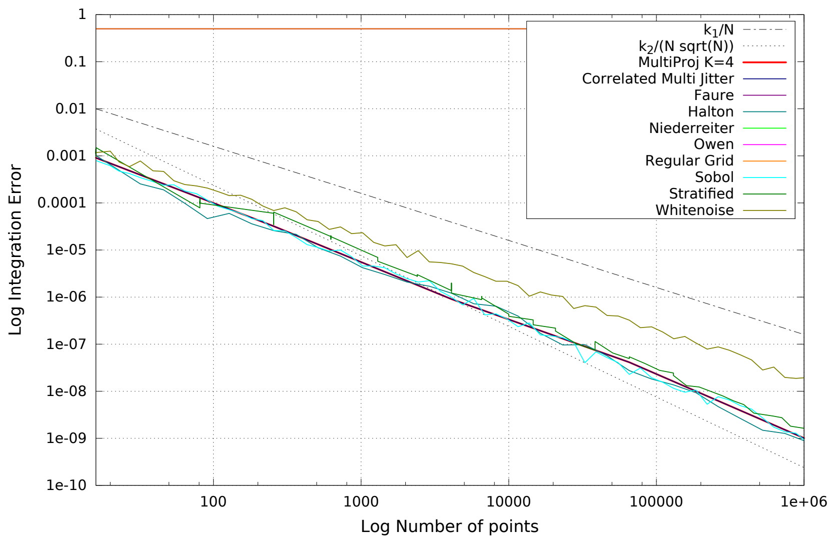
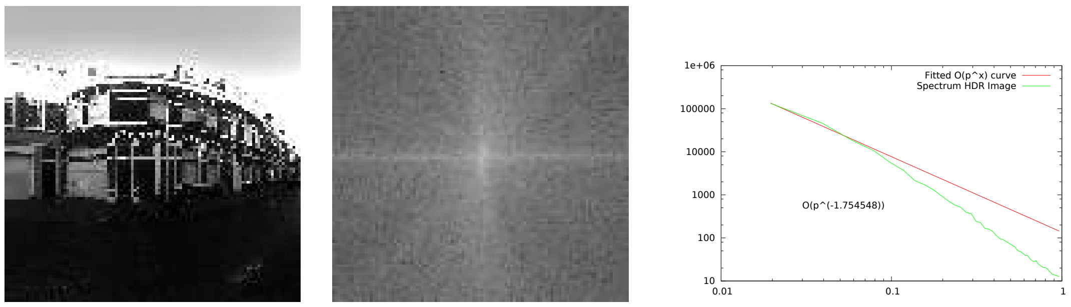
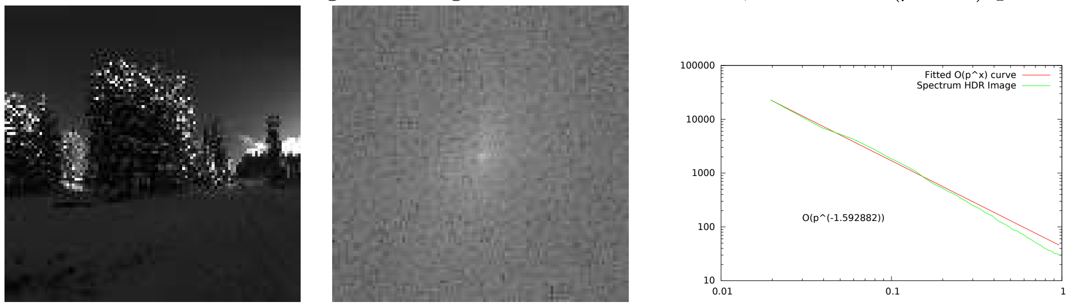


Fig. 7.18.: Comparing the variance of Monte Carlo estimators of our samplers with reference samplers when integrating an analytical 4-D sphere.



Results from a HDR Image showing man made structures, with an $O(\rho^{-1.75})$ profile.



Results from a HDR Image showing a natural scene, with an $O(\rho^{-1.59})$ profile.

Fig. 7.19.: Spectral characteristics of HDR integrands.

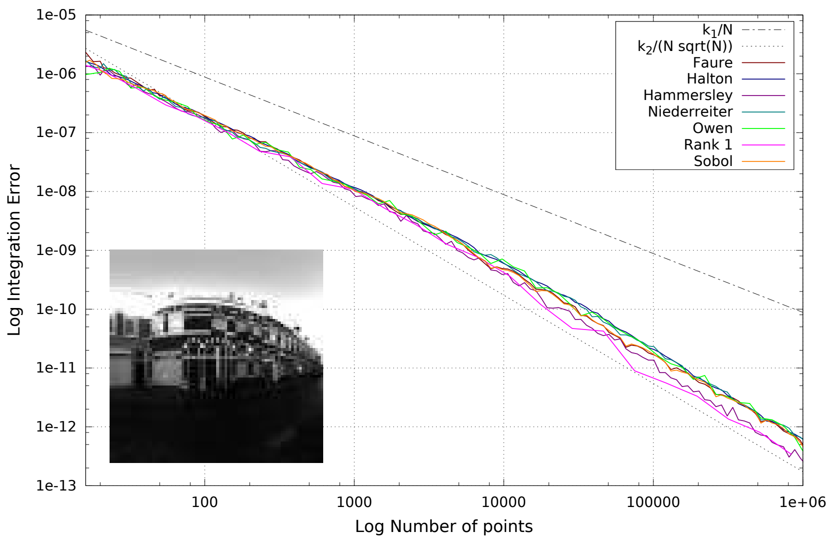


Fig. 7.20.: Variance of Monte Carlo estimators from Low Discrepancy samplers when integrating the inset hdr image.

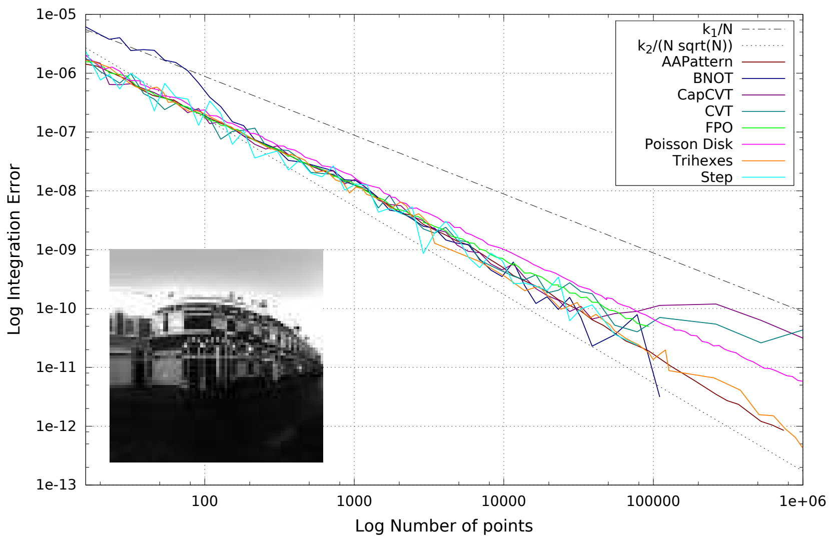


Fig. 7.21.: Variance of Monte Carlo estimators from Blue Noise samplers when integrating the inset hdr image.

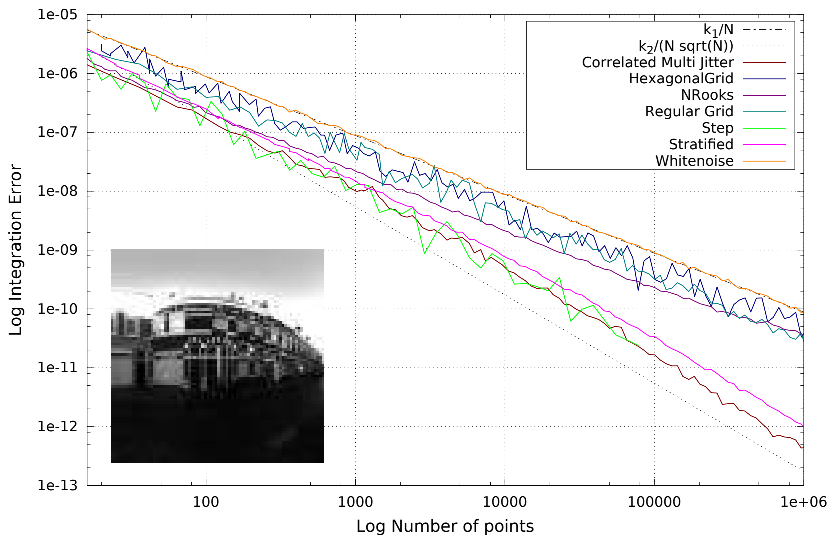


Fig. 7.22.: Variance of Monte Carlo estimators from various samplers when integrating the inset hdr image.

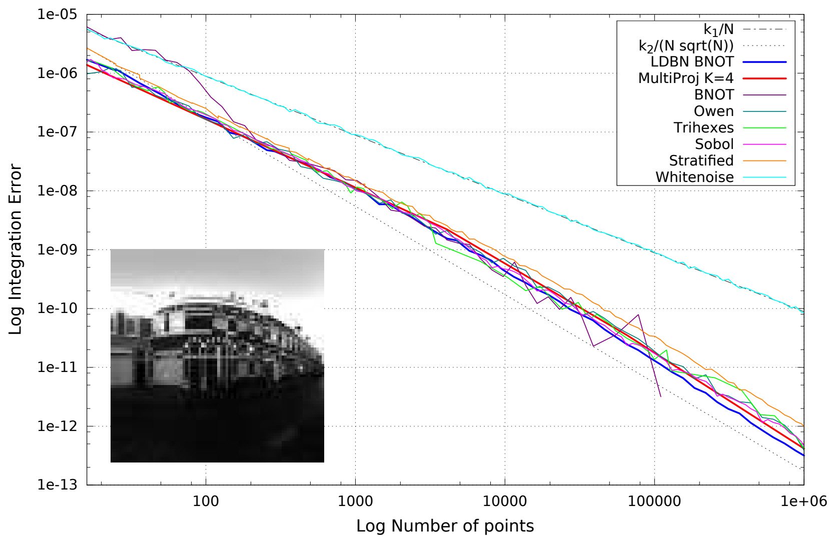


Fig. 7.23.: Comparing the variance of Monte Carlo estimators of our samplers with reference samplers when integrating the inset hdr image.

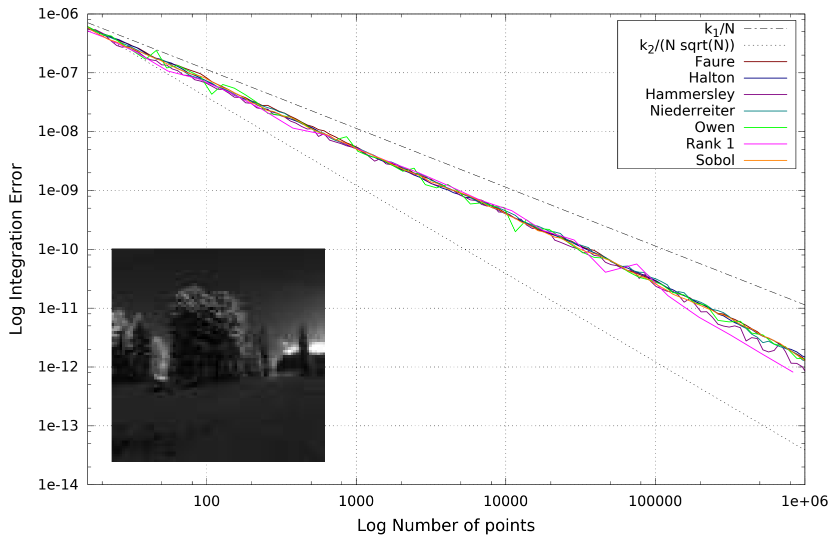


Fig. 7.24.: Variance of Monte Carlo estimators from Low Discrepancy samplers when integrating the inset hdr image.

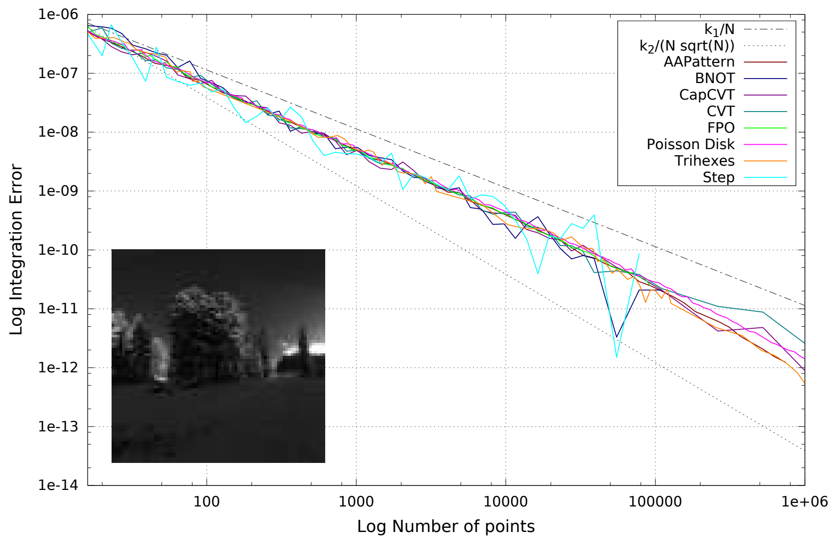


Fig. 7.25.: Variance of Monte Carlo estimators from Blue Noise samplers when integrating the inset hdr image.

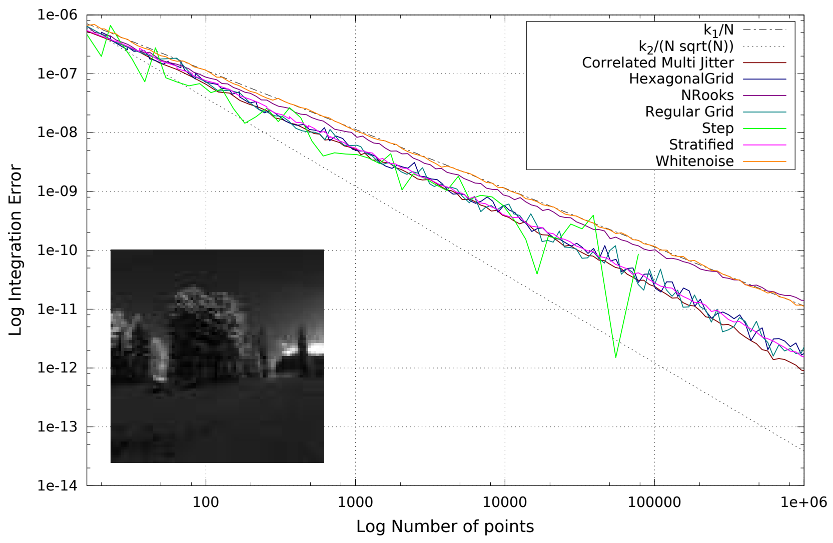


Fig. 7.26.: Variance of Monte Carlo estimators from various samplers when integrating the inset hdr image.

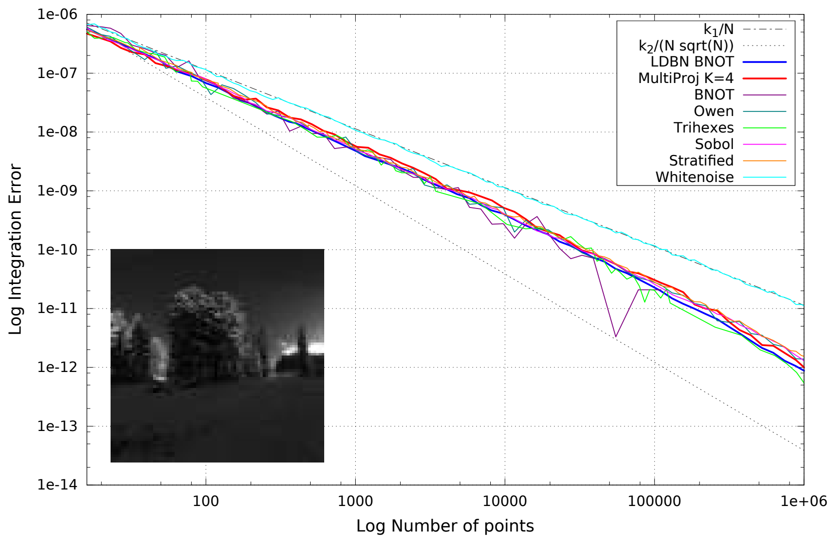
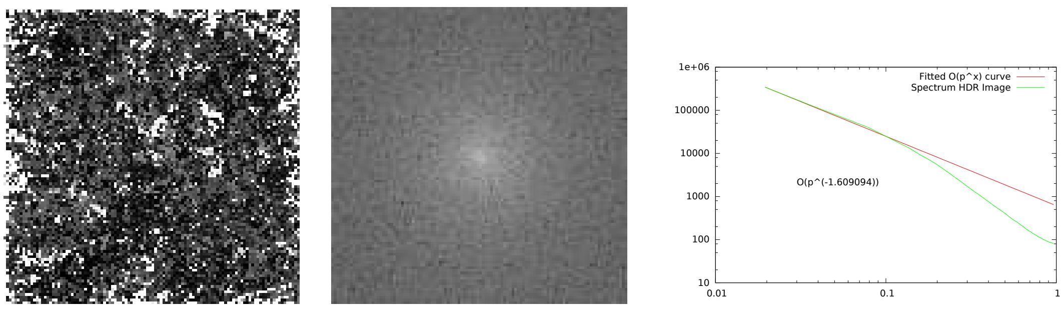
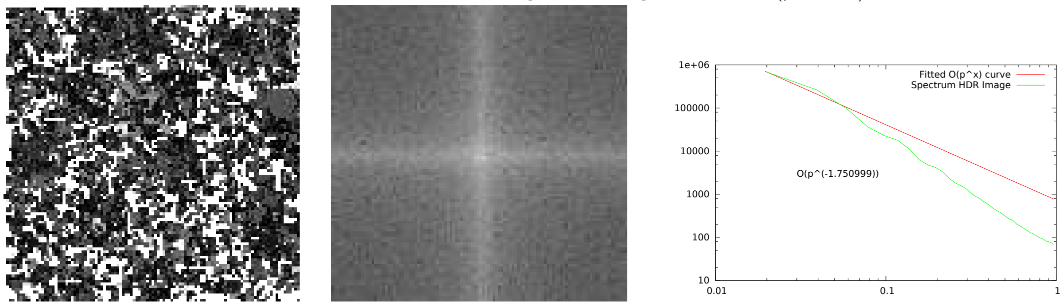


Fig. 7.27.: Comparing the variance of Monte Carlo estimators of our samplers with reference samplers when integrating the inset hdr image.



Properties of ideal random triangle arrangements $O(\rho^{-1.60})$ profile.



Properties of ideal aligned box arrangements $O(\rho^{-1.75})$ profile.

Fig. 7.28.: Spectral properties of complex analytical arrangements. It is clearly visible in log scale that compared to the natural image from Figure 7.19, we still lack high frequencies.

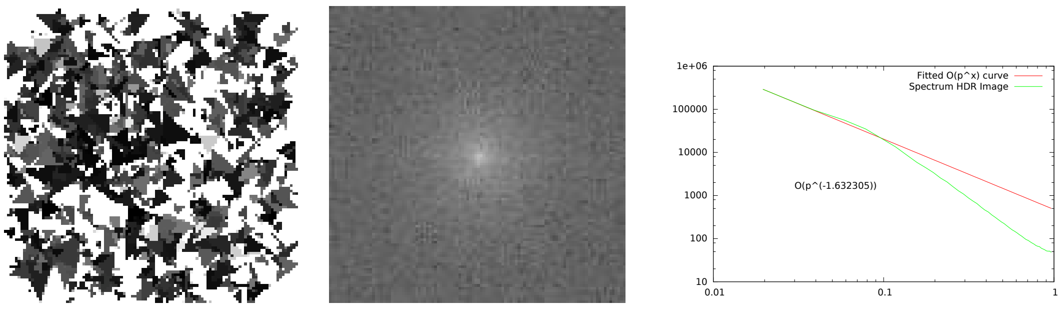
is to generate random synthetic analytical images, with a controllable profile spectrum, with arrangements of boxes and triangles.

We therefore generate randomly 2-D boxes (or triangles) of various sizes within a given domain. By controlling the number of shapes of a given size, we can control the amount of higher/lower frequency in the final image. This control could be made finer by using white,blue,pink or green distributions of shapes to increase a given range of frequencies. As each shape is analytically integrable, we have the analytical integrand for the first shape, which is simply updated each time we add a new shape within the arrangement. Technically, we do this using the CGAL library [Cga], which provides efficient tools to handle such 2-D arrangements.

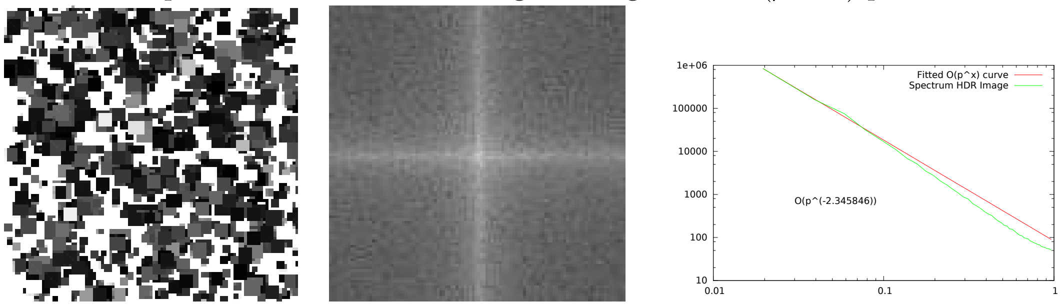
However, in our current implementation, to create integrands with a radial profile similar to natural images (between $O(\rho^{-1.5})$ and $O(\rho^{-2})$), we need to generate very complex arrangements (containing around 15000 elements of various sizes and orientations (Fig. 7.28)). Unfortunately, when used for integration, such arrangements lead to biased results. We do not know for sure today what causes this bias but it seems that the analytical evaluation of those arrangements leads to numerical errors.

In this section we present integration error as measured for simpler and unbiased analytical arrangements. Figure 7.29 on the next page, with radial profiles between $O(\rho^{-2})$ and $O(\rho^{-2.5})$ (Fig. 7.29 on the following page).

The Box arrangement presents more anisotropy than the Triangle arrangement and emphasizes the behaviour of samplers when integrating axis aligned discontinuities. When using those synthetic images to measure convergence speed in integration, we get consistent results as those



Properties of our random triangle arrangements $O(\rho^{-2.25})$ profile.



Properties of our aligned box arrangements $O(\rho^{-2.34})$ profile.

Fig. 7.29.: Spectral properties of simpler analytical arrangements.

that would be obtained from true HDR images (Figures 7.30 on the next page and 7.31 on the facing page). Note that, as computing those arrangements takes some time, in this section we plotted the true MSE instead of the variance over several randomized realisations of the samplers/the integrands, hence the noisy curves.

7.3 Aliasing

In the previous sections, we studied how approximating a single integrand converges. However, this is insufficient in a rendering context, where many integrands are solved (one for each pixel), and where the set of all approximations values are then reconstructed into a full image. We study in this section how the integration error from each samplers is distributed within this full image. A stochastically distributed error leads to noise, and a regularly distributed error leads to aliasing.

We present here results from reconstructing a $2-D$ function with a wide range of frequencies, and results of various samplers in a true rendering context.

7.3.1 Zoneplate

Our first test to measure this is the reconstruction of the 2-D Zoneplate function. This reconstruction is done by sampling the function with a 2-D sampling pattern and then using a Mitchell-Netravali filter [MN88] to retrieve the original function from all the measured discrete

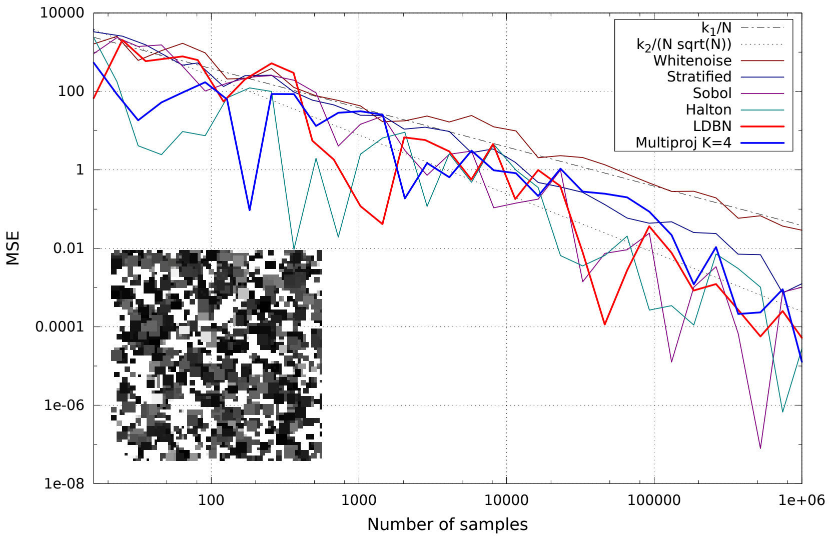


Fig. 7.30.: Comparing the MSE of Monte Carlo estimators of our samplers with reference samplers when integrating synthetic box arrangements.

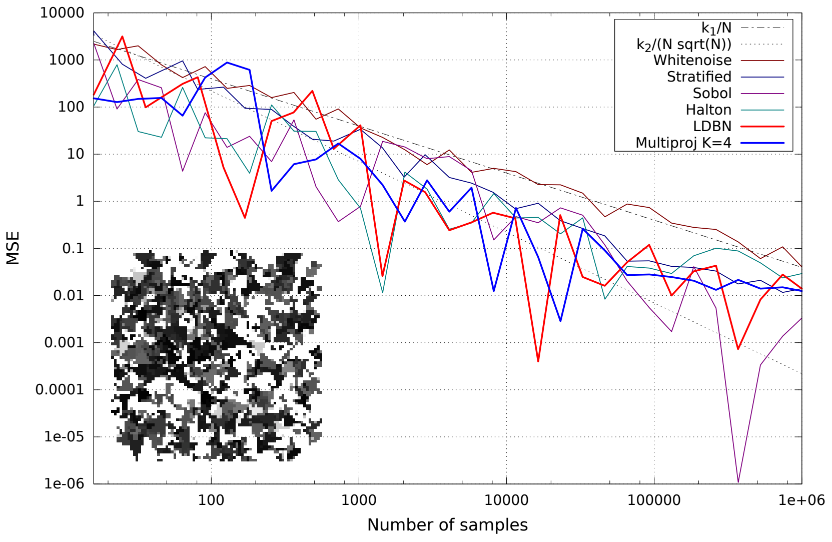


Fig. 7.31.: Comparing the MSE of Monte Carlo estimators of our samplers with reference samplers when integrating synthetic triangle arrangements.

values. See Appendix A on page 168 for more informations. This Zoneplate function f_Z is illustrated Figure 7.32 and is formally expressed by

$$f_Z(x, y) := sin(x^2 + y^2). \tag{7.4}$$

It is interesting to us as this function presents a very wide range of frequencies, which is very difficult to capture when discretizing. Aliasing becomes obvious when using this function, and its relationship to the Fourier spectrum is also easily visible.

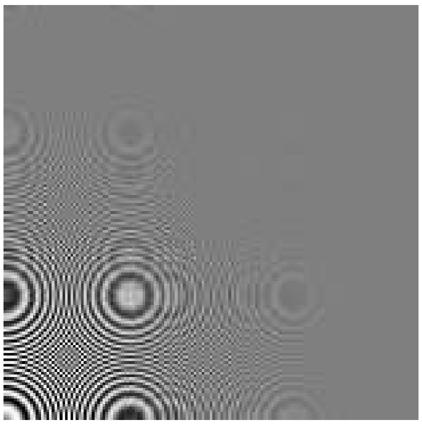


Fig. 7.32.: Ground truth reconstruction of the Zoneplate function.

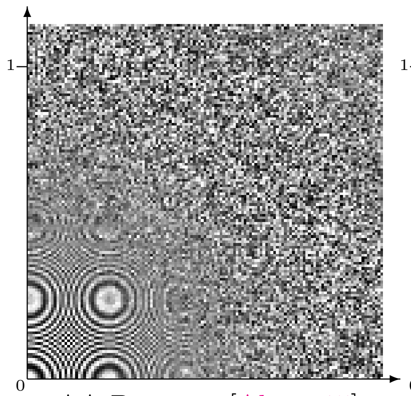
The zoneplate images from Figure 7.33 show that Blue Noise samplers allow for a good reconstruction up to a certain frequency. Above that frequency, the reconstruction turns to noise. Among this noise, an arc of stronger noise artefacts appears. This arc is due to the peak in the Blue Noise fourier spectrum. Similarly, in Figure 7.34, the Low discrepancy samplers would perform a very good reconstruction, as no noise is visible. The issue is that instead, it turned this noise into very visible artefacts. Those artefacts actually align with the peaks in the Fourier spectrum of the sampling pattern used to do this reconstruction. Finally, the samplers presented in Figure 7.35 usually generate more noise than Blue Noise sampling patterns, and the lattice samplers present artefacts stronger than the ones from the Low discrepancy samplers.

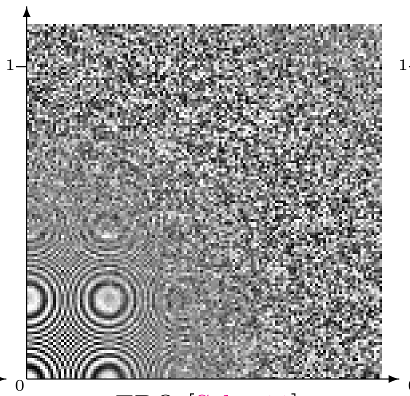
In Figure 7.33, the zoneplate reconstruction from the samplers of our methods are presented. Those reconstructions are very similar to the ones from classical Blue Noise sampler, which was to be expected as our methods present a Blue Noise Fourier spectrum.

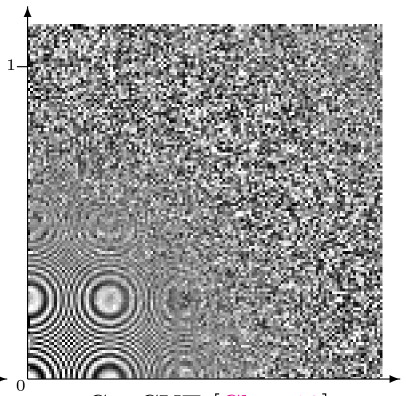
7.3.2 Renderings

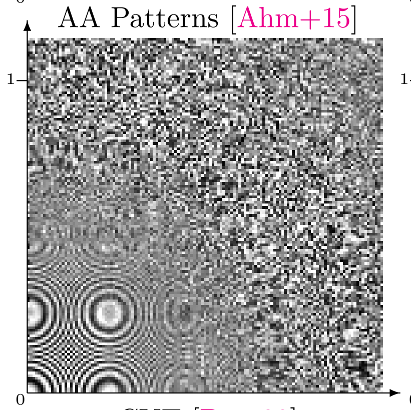
In this section, we present the performances of our second method (Chapter 6 on page 107) in a rendering context. All our rendering examples were performed using the PBRT engine [Pha+16]. For this whole section, we strongly advise the reader to refer to the pdf version of this document as the printing may have degraded the quality and readability of the results.

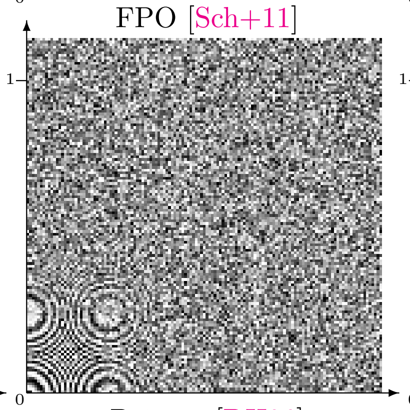
To use our sampler within this engine, we generated a set of N s-D samples (usually with $N = 2^{18}$ or $N = 2^{20}$), and tiled this set so that each pixel is covered by the expected number of samples. We could have instead generated independently but this would have led to correlation

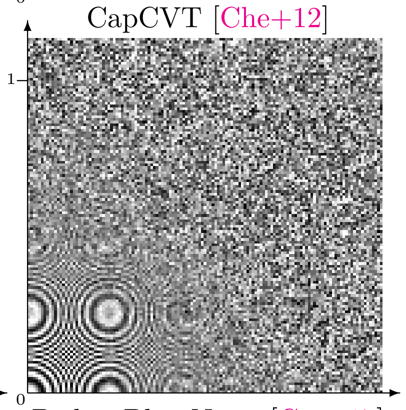

AA Patterns [Ahm+15]


FPO [Sch+11]


CapCVT [Che+12]


CVT [Du+99]


Poisson [DH06]


Dither Blue Noise [Cor+17]

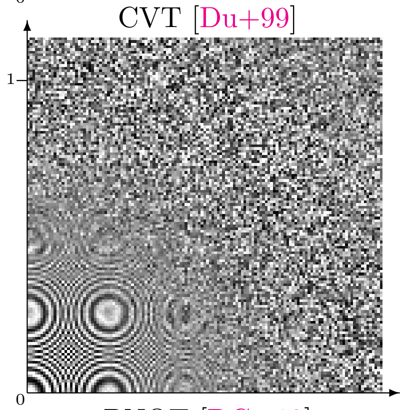

BNOT [DG+12]

Fig. 7.33.: Zoneplates of various Blue Noise samplers, using 262144 samples to reconstruct a 256x256 image.

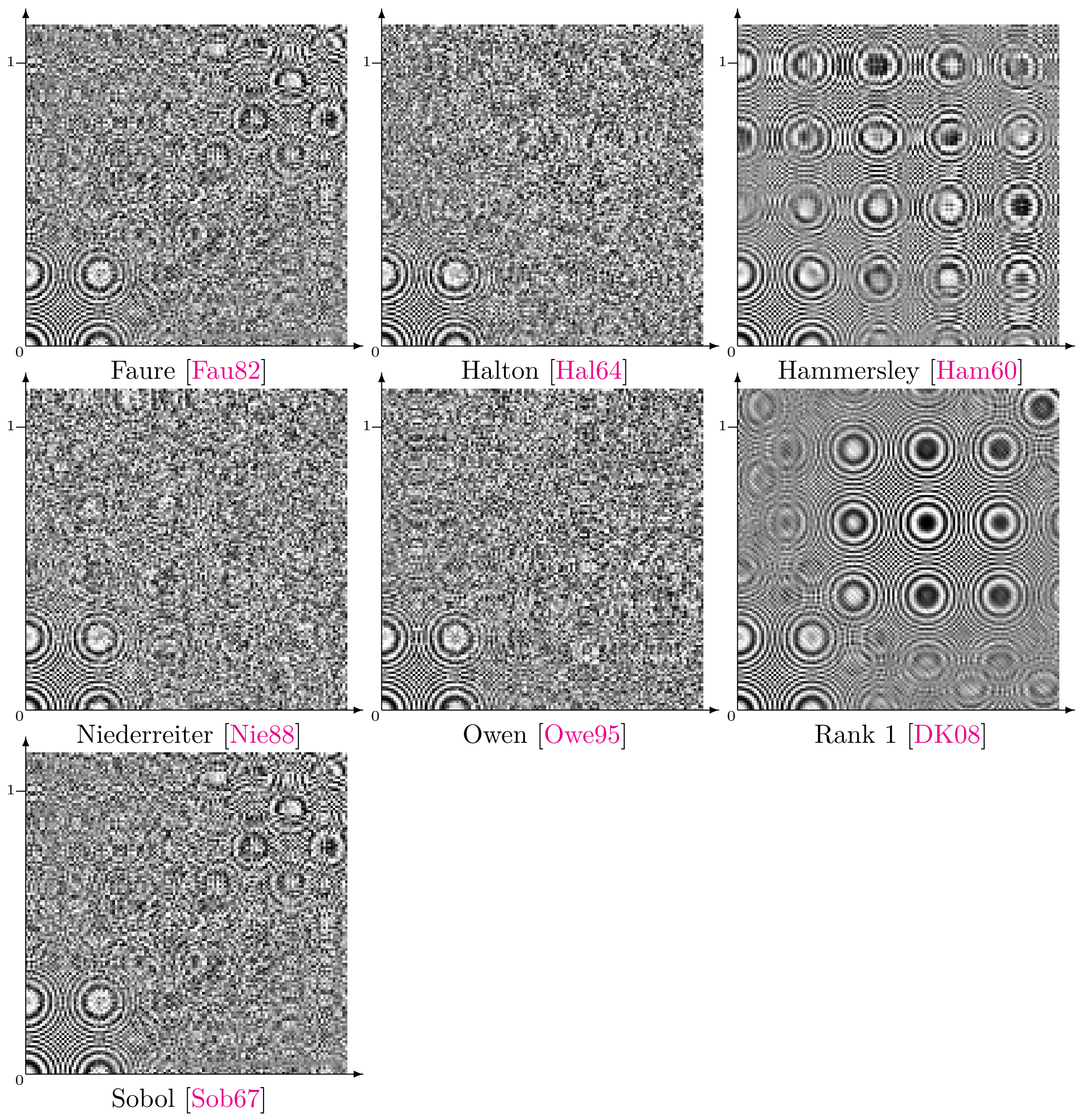


Fig. 7.34.: Zoneplates of various Low Discrepancy samplers, using 262144 samples to reconstruct a 256x256 image.

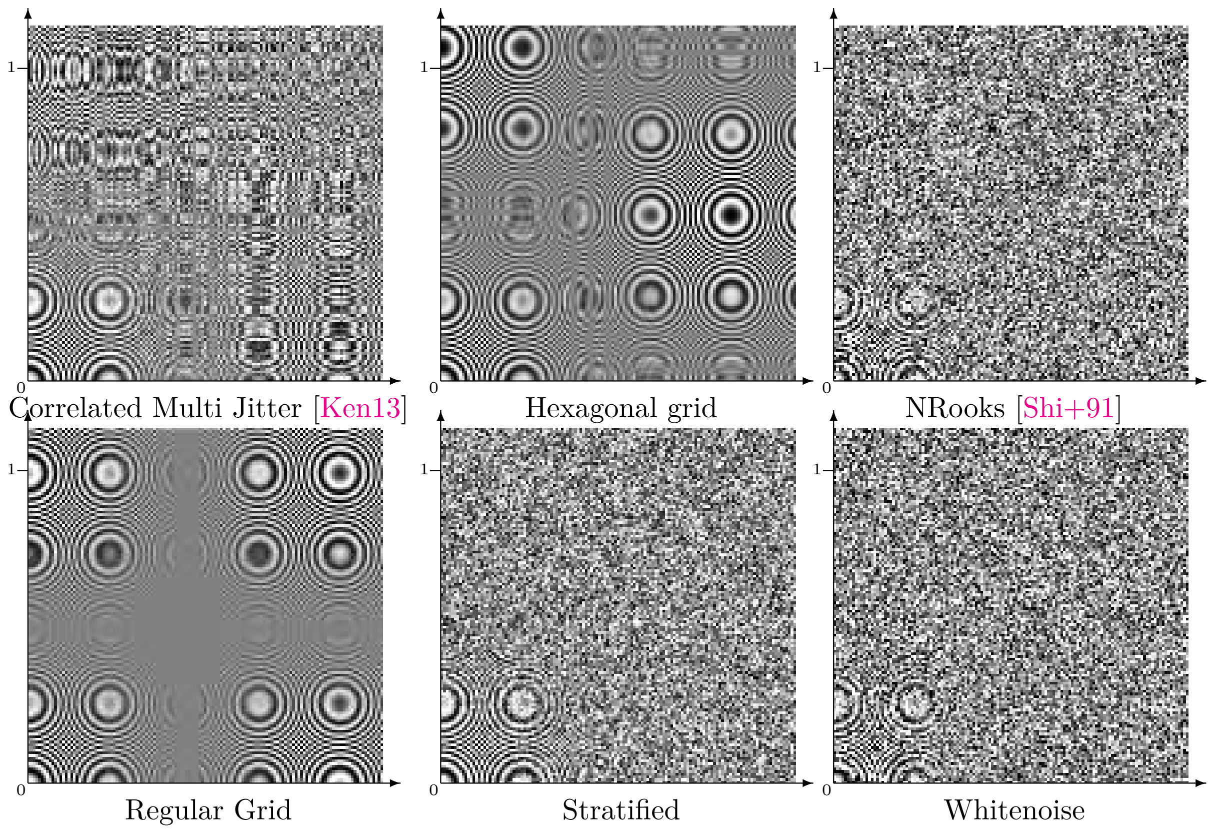


Fig. 7.35.: Zoneplates of various samplers, using 262144 samples to reconstruct a 256x256 image.

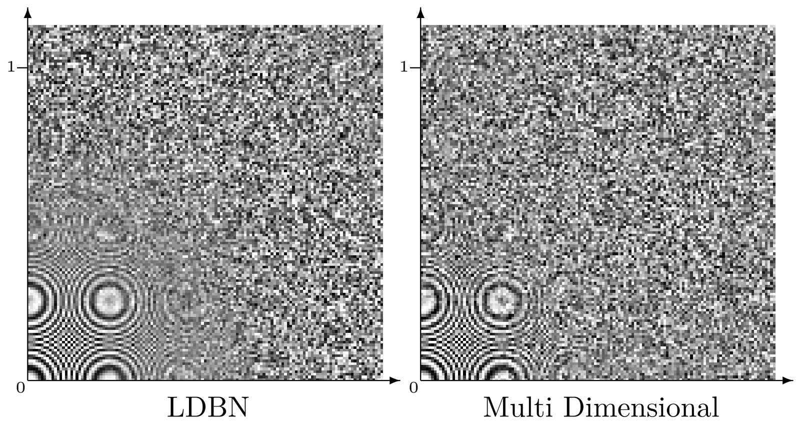


Fig. 7.36.: Zoneplates of our samplers, using 262144 samples to reconstruct a 256x256 image.

between pixels that would have led to very disturbing artefacts. We chose to tile our sampler as, for implementation simplicity, we are reading the samples from an input file instead of having them generated on the fly by the system. Thus, to avoid reading and storing huge files, we chose to tile a smaller set instead. We also used this method to perform renderings from Owen's scrambled Sobol sets.

We compared our sampler to Sobol and Halton, which are both generated using the same global stretching of N samples. The only difference being that as Sobol and Halton are defined analytically, it is possible to generate billions and billions of samples and thus no tiling is required. On the contrary, the Stratified sampler renders a scene by generating a new independent set of samples within each pixel.

We rendered several scenes, and compared the mean squared error (MSE) of the rendered scene to the ground truth scene. Note that the MSE given here is not the same as the MSE given in Section 7.2 on page 140. Section 7.2 on page 140 measures the error/variance when integrating a single scene, returning a single value as the integrand of the whole scene. The rendered scenes here are made from an aggregation of approximated integrand values, one for each pixel. The MSE measured here can therefore be perceived a sum of all the MSE from all the integration errors from all pixels. It does not take into account how the error is distributed, and is not affected by aliasing artefacts.

In Figure 7.37, we start by using our sampler solely for 4 dimensions. This scene uses 8 dimensional samples: 2 for the image plane, 2 for the point on the light, 2 for the light scattering, and 2 for the object reflectance function. We rendered it using various samplers to sample the 2D image plane and the 2D light source, while all other dimensions sampled using the Sobol sequence. Since the Sobol sequence is deterministic, differences between the images are only due to the difference in sampling for the image and light and therefore allow for an easy comparison of the different samplers. It can be seen that our sampler has a MSE similar to Sobol, but no longer presents aliasing in the shadow. Note that we used our sampler with 64 samples per tile ($K = 8$), and the Sobol indices (1, 4) and (2, 3).

In Figure 7.38, we render a complex scene with various samplers. In the case of our sampler, we used 64 ($K = 8$) samples per tile and optimized 3 2-D projections, suing the Sobol indices (1,4), (2,3) and (5,7). Those optimized projections were used for the image plane, the light and the lens (for the depth of field effect). We used our hierarchical Owen' scrambling (Section 4.2 on page 79 for the remaining dimensions. Similarly to Figure 7.37, we note that our approach's MSE is better than most methods as is is similar to Sobol's. However, we present much less aliasing.

Figure 7.39 presents the difference in renderings when using 16 ($K = 4$) or 64 ($K = 8$) samples per tile on a very complex scene requiring 20-D samples. We used the same setup as for Figure 7.38, which is optimizing the projections for the image plane, the lens, and the light and filling the other dimensions with Owen' scrambling. Note that when using 4 samples per pixel, the crown appears darker. This phenomenon is still unexplained.

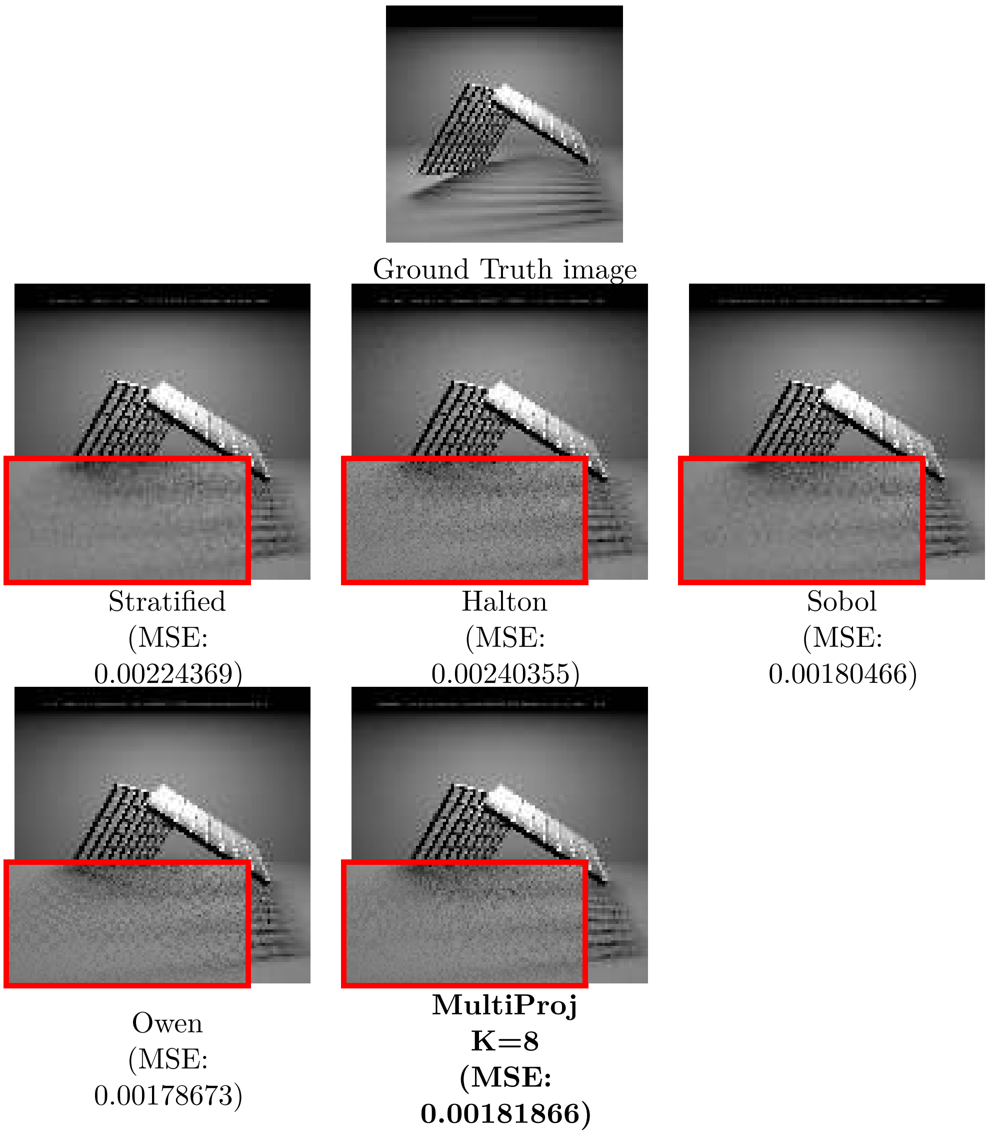


Fig. 7.37.: Direct lighting (single light bounce) with 16 samples per pixel. The deterministic Sobol sampler is used on every dimension apart from the light plane. This dimension was sampled using various samplers, allowing to compare the impact of each of them and compare it with our sampler.

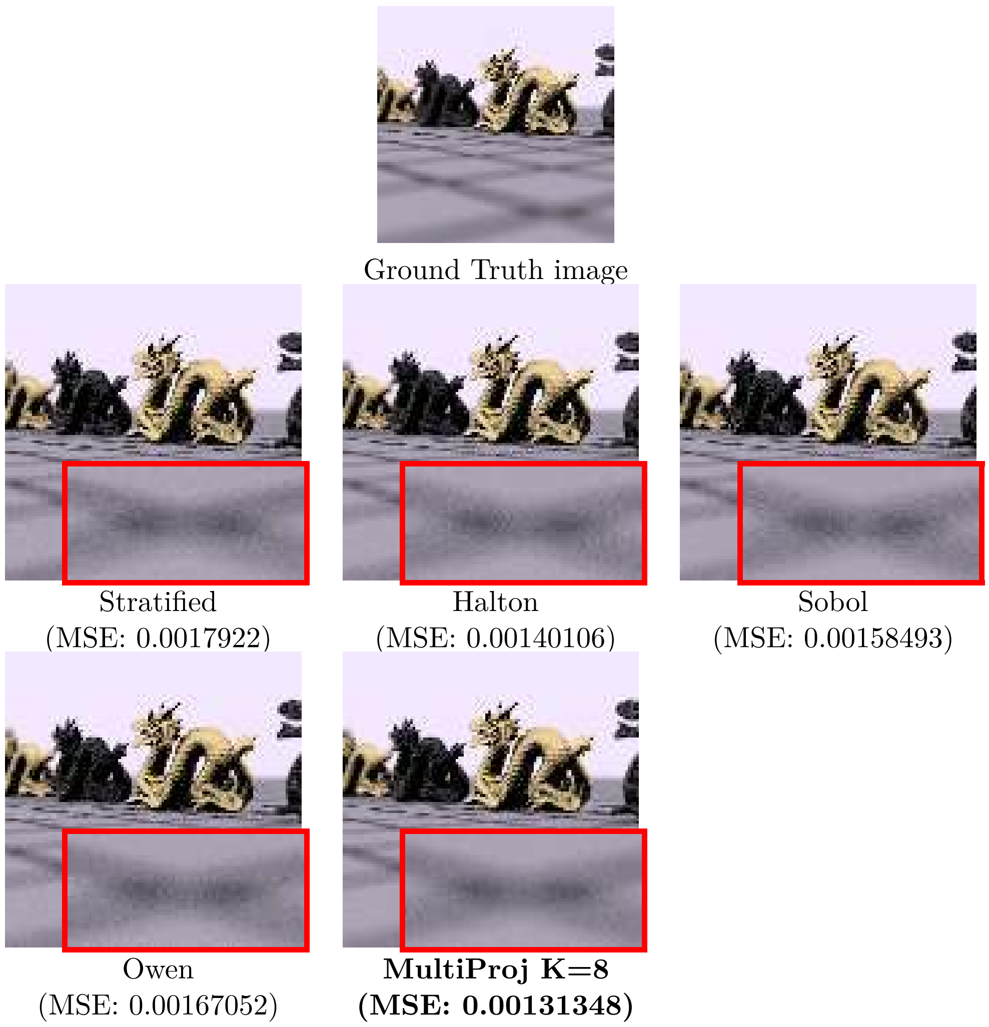


Fig. 7.38.: s-D Renderings with 16 samples per pixel and various samplers. The scene rendered presents depth of field effect, and uses a single light bounce. When using our sampler, the three optimized projections affect the sampling of the image plane, the sampling of the lens for the Depth of Field effect, and the light plane.

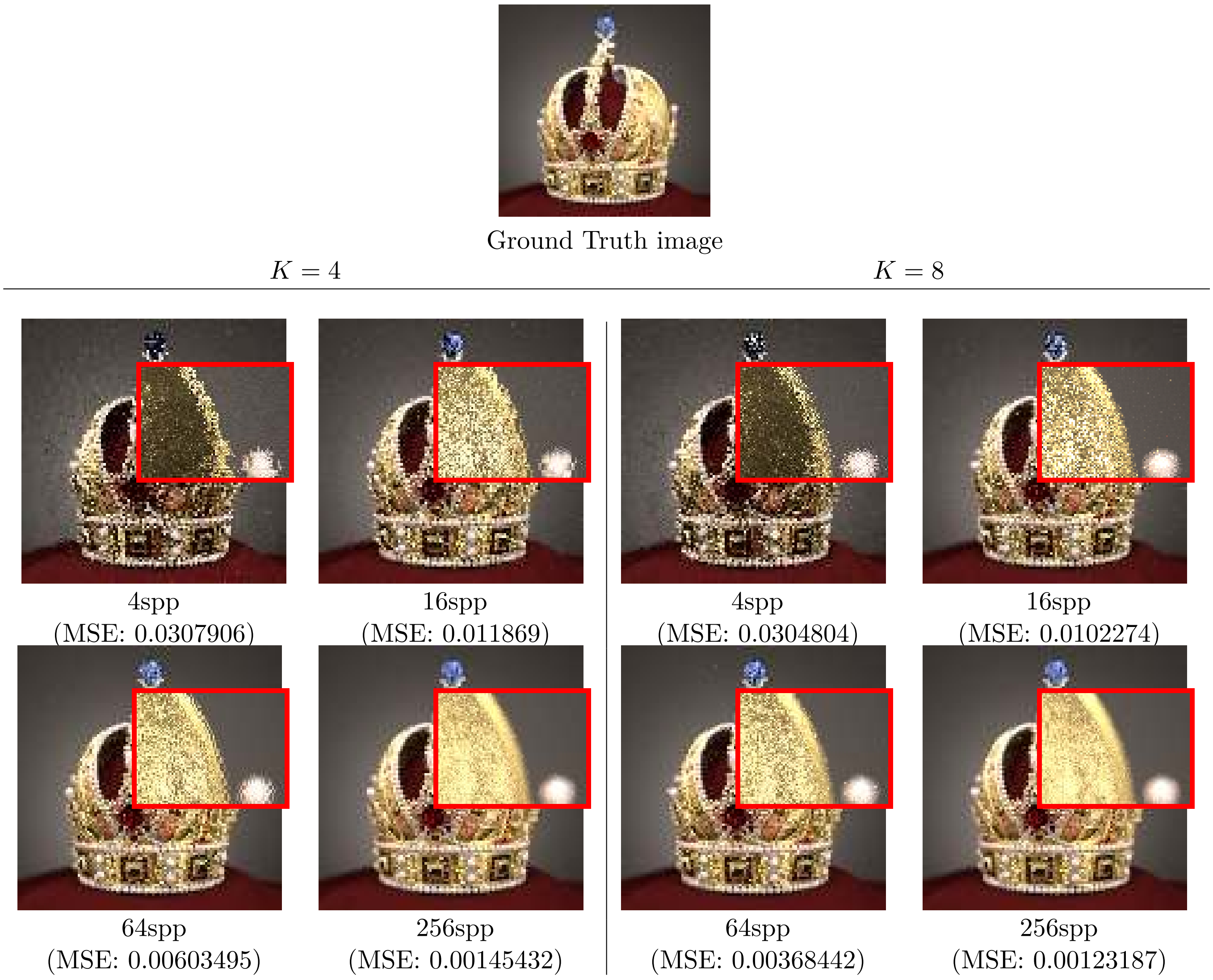


Fig. 7.39.: Renderings with our sampler when $K = 4$ and $K = 8$ with a various number of samples per pixel. The scene rendered presents depth of field effect, and uses 3 light bounces. The optimized projections affect the sampling of the image plane, the sampling of the lens for the Depth of Field effect, and the sampling of the light plane.

Sampler	10	10^2	10^3	10^4	10^5	10^6
			Low Discrepancy Samplers			
Faure	0.00000s	0.00002s	0.00028s	0.00383s	0.05199s	0.64523s
Halton	0.00000s	0.00002s	0.00020s	0.00287s	0.03884s	0.48352s
Hammersley	0.00000s	0.00000s	0.00001s	0.00014s	0.00132s	0.01135s
Niederreiter	0.00009s	0.00009s	0.00019s	0.00120s	0.01064s	0.10320s
Owen	0.00024s	0.00038s	0.00147s	0.01899s	0.23139s	2.93118s
Sobol	0.00005s	0.00003s	0.00007s	0.00049s	0.00534s	0.04965s
			Blue Noise Samplers			
CVT	0.02736s	0.14151s	1.52076s	24.34502s	392.36278s	
CapCVT	0.02611s	0.13227s	1.49233s	25.00627s	386.62341s	
FPO	0.01995s	0.01957s	0.30386s	7.92847s	419.76541s	
BNOT	0.05781s	0.76544s	15.77232s	577.65501s		
			Other Samplers			
CMJ	0.00002s	0.00001s	0.00005s	0.00037s	0.00362s	0.03723s
HexagonalGrid	0.00003s	0.00000s	0.00004s	0.00019s	0.00172s	0.01453s
NRooks	0.00000s	0.00001s	0.00006s	0.00051s	0.00674s	0.10854s
Rank1Fibo	0.00000s	0.00000s	0.00002s	0.00029s	0.00371s	0.01691s
RegularGrid	0.00007s	0.00001s	0.00006s	0.00010s	0.00093s	0.00690s
Stratified	0.00005s	0.00002s	0.00008s	0.00031s	0.00301s	0.02819s
Whitenoise	0.00000s	0.00000s	0.00002s	0.00021s	0.00229s	0.01978s
Step	0.93656s	0.83110s	7.32909s	620.91383s		
			Our Samplers			
BNLDS	0.01423s	0.00139s	0.00806s	0.14148s	2.78297s	2.78475s
LDBN	0.02936s	0.01231s	0.01199s	0.01315s	0.01409s	0.02416s

Tab. 7.1.: Timings table

7.3.3 Timings

In this section we present timings measured from the samplers implementation from the UTK framework. Table 7.1 presents the times in seconds need for a sampler to generate a given amount of samples. it can be noted that the analytical Low discrepancy samplers are very efficient, when the Blue Noise samplers based on optimization cannot generate more that 10^5 samples. All those tests were performed on a CPU Intel® Core™ i5-4440 CPU @ 3.10GHz $\times$ 4.

7.4 Conclusion

The experiments presented in this chapter led to the following conclusions. First, they experimentally confirm our intuition regarding the Koksma Hlawka theorem. Meaning that despite that our scenes are not of finite variations in the sense of Hardy and Krause (as required by the original Koksma Hlawka theorem), the discrepancy seem to still be related to the integration error as an upper bound. We observe thar the Low discrepancy samplers present an integration error that is related to their discrepancy, and decreases in $O(\log(n)/n)$ (in 2-D). It also confirmed the theoretical convergence speeds of various stochastic samplers from [Pil+15] and listed in Table 3.1 on page 70. The Zoneplate test allowed to see the strong aliasing artefacts that occurs when peaks are visible in the Fourier spectrum, and the renderings allow to see how a structured sampler leads to a structured repartition of the error in rendering. For all those experiments, our samplers performed as expected. Meaning that their Low Discrepancy gives them a convergence speed in $O(\log(n)/n)$ (in 2-D), and their Blue Noise spectrum removes all structured error from the renderings. This confirms that our new proposed solutions, contrary to previous sampling patterns, performs well both for aliasing and integration error.

Conclusion 8

Generating a photo-realistic image on a computer is a complex problem, that involves solving high dimensional recursive integrands. Unfortunately, those integrands cannot be solved analytically. Instead, we rely on a numerical approximation of the integrand, evaluated using a Monte Carlo estimator. Roughly, such estimators sample the integrand, generating several discrete values, and deduce the integration result as a sum of all those values. There are several ways to chose which sample of the integrand should be taken, each leading to different results. In this PhD, we have studied many sampling patterns and evaluated how they perform in a rendering context. A sampling pattern is considered a good sampling pattern if the Monte Carlo approximation converges as fast as possible to the true integration value, and if the integration error is not structured in the final image (if there aren't any aliasing artefacts).

It was shown that samplers that are said to be Low Discrepancy are the ones with the fastest convergence (in the worst theoretical case). However, they usually also come with a very regular structure, which also structures the error in the final image. Furthermore, the Low discrepancy guarantees a uniform coverage of the domain only asymptotically. As example, the Sobol sequence is Low discrepancy, but may present a distribution of samples aligned on the diagonal when the number of samples taken is too low.

On the other hand, the better non-aliased sampling patterns are the Blue Noise patterns. Such samplers ensure a better covering of the domain by forcing all samples to be equidistant from each other. However, they asymptotically converge slower than Low Discrepancy samplers.

Our contributions aimed at developing samplers that would be Low Discrepancy, and present a Blue Noise spectrum. We demonstrated that from this association we have sampling patterns that have an optimal convergence speed in integration, and an absence of aliasing in the final rendered image. Furthermore, compared to pure randomization of a low discrepancy set, the Blue Noise spectrum improves the approximation by ensuring a more uniform covering of the domain, even for a given low number n of samples, where the asymptotic low discrepancy property cannot apply.

Our first contribution (Chapter 5 on page 93), aimed at providing the theoretical and empirical proof that a Blue Noise spectrum is not incompatible from a Low Discrepancy property. However, this contribution is limited to 2-D point sets.

Our second contribution (Chapter 6 on page 107), allows to scramble specific 2D projections of a Sobol sequence to generate higher dimensional patterns with Blue Noise and Low discrepancy properties in those projections. Therefore, during rendering, the integration error decreases quickly, and no aliasing appears in the final image. Furthermore, our method generates sequences of samples.

However, this contribution still leaves the path open future work. First, the K factor is a difficult trade off. As noted in Section 7.1.1 on page 133, our projections are only low discrepancy for sets of size $n = a^{2^K}, a \in \mathbb{N}$. Therefore, the higher the K value, the more degraded the discrepancy. But on the other hand, with a higher K, we can hope for a better spectrum for the final set. An aware reader may have already noted that this drawback is due to the fact that our method generates $(0, a, 2)$-nets in base 2^K. This choice of a higher basis was motivated by the intuition that that the Sobol sequence is the only existing $(0, 2)$-sequence in base 2 (modulo the order of the samples). As I could not find formal proof of this, this is still conjecture. Nonetheless, creating Blue Noise sequences from digital nets seems to require looking among sequences defined in basis $b > 2$.

Still, the main reason we aim at having a Blue Noise spectrum comes from the fact that, for a long time, aliasing was considered worse that stochastic noise as it is much more noticeable to the human eye. However, such artefacts only occur for non-converged renderings, and such renderings are most the time used as input for denoising algorithms (such as the one from [Sch+17]). Such algorithms denoise images generated with as little as 1 sample per pixel. Therefore, choosing the type of noise to generate in an image should maybe be weighted against what denoising algorithms can efficiently remove, instead of what the human eye perceives. With their structured error, aliased low discrepancy samplers may be in fact easier to denoise than stochastic samplers.

Finally, in the particular case of rendering, many approaches rely on importance sampling to adapt the density of samples pixel-wise, depending on the complexity of the scene behind those pixels. Such methods have been further improved through the use of deep learning, leading to impressive results [DK17]. In our contributions, we either have no adaptivity, or an adaptivity constrained by the precomputed optimized sets within each tile. To accommodate importance sampling, such samplers need to be made more adaptive, maybe using optimization based techniques as the one presented in Section 4.3 on page 80, assuming such methods could be performed in real-time.

A Discretization and Reconstruction of a continuous signal

In this section, we present the theoretical basis to reconstruct a signal from a set of evaluations. We will start by defining what is a signal, and how we can discretize a continuous signal. Then, we will see under which conditions we can properly reconstruct the original signal from its discrete form.

All the notations we use regarding signal theory follow the ones from [Gla95]. We start by properly defining what is a continuous signal, and how it differs from a discrete signal. We denote continuous signals as *continuous-time* (CT) signals. Those signals (despite their name, which stands mostly for historical reasons) do not necessarily have anything to do with time. They are the family of signals defined from analytic functions; if f is a continuous signal, then an infinitesimal difference in x leads to an infinitesimal difference in $f(x)$. Note that those signals are not necessarily differentiable everywhere. Those signals are often used to modelize the real world and correspond to theoretical definitions. As their domain of definition is infinite, as soon as you try to evaluate such signals with a computer, you have to turn then into *discrete-time* (DT) signals.

A DT signal can be seen a set of s-dimensional samples with their associated values. It is not continuous, not defined on a infinite domain, and not differentiable. Since a computer cannot represent the infinite amount of existing numbers (due to the floating precision limit), they are the only kind of signals manageable on a computer. Numerizing a CT signal implies that this signal will be discretized over a set of computer-representable values. Even though most often this is ignored as this floating point precision is higher than what we consider negligible error, we will see that when handling signals with an infinite maximal frequency, this becomes a problem. The difference between CT and DT signals is illustrated Figure A.1.

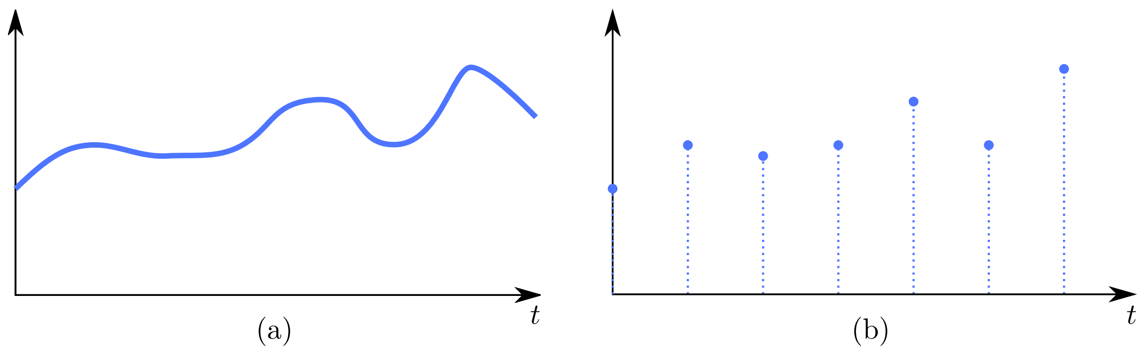


Fig. A.1.: (a) A 1-D CT signal $f(t)$ and (b) its equivalent DT signal $\dot{f}(t)$

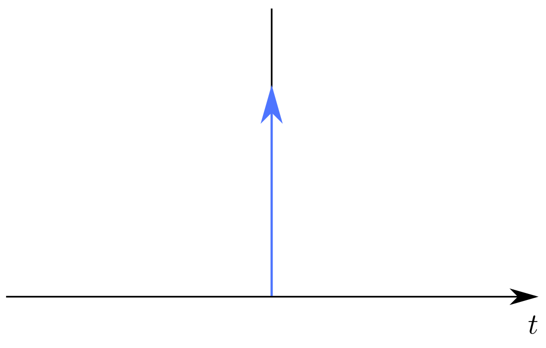


Fig. A.2.: Dirac impulse $\delta(t)$.

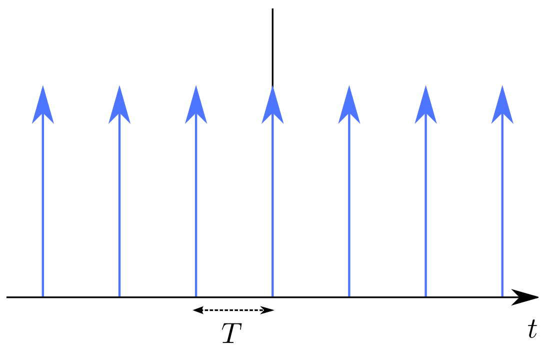


Fig. A.3.: Dirac Comb $\text{III}_T(t)$.

A DT signal is obtained from a CT signal by computing the product of the CT signal with particular signals, namely the Dirac impulse and the Dirac comb.

A.1 Discretization

The Dirac impulse (Fig. A.2) is a CT signal defined as

$$\delta(t) = 0 \quad \text{if } t \neq 0, \tag{A.1}$$

with

$$\int_{\infty}^{-\infty} \delta(t) dt = 1. \tag{A.2}$$

The issue here is that $\delta(0)$ has therefore an undefined value. Since the point 0 is infinitely small, to ensure both Equation (A.1) and Equation (A.2), we need to have $\delta(0)$ such that $\frac{\delta(0)}{\infty} = 1$ which is undefined. To solve this issue, the Dirac impulse can be redefined as a DT signal with

$$\delta(t) := \begin{cases} 1, & \text{if } t = 0 \\ 0, & \text{otherwise }. \end{cases} \tag{A.3}$$

We define a Dirac comb (Figure A.3, sometimes also called Dirac train) from a set of equally spaced Dirac impulses as:

$$\text{III}_T(t) := \sum_k \delta(t - kT), \tag{A.4}$$

where T is the interval between the Dirac impulses. What makes such signals so important is that to discretize a CT signal, we compute the product of this signal with a Dirac comb (Fig. A.4 on the following page).

A discrete signal can also be reconstructed back to a CT signal, but only under some conditions. To understand those conditions, we need to look at what happens in the frequency space when we discretize a signal.

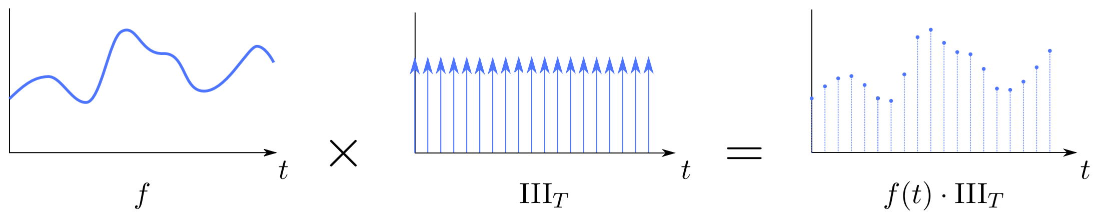
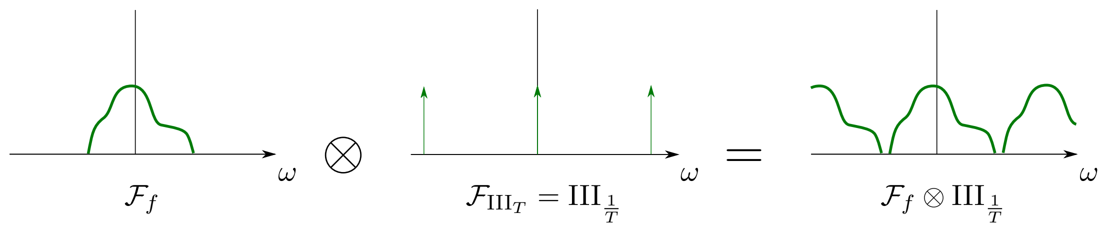


Fig. A.4.: Discretizing a signal in spatial space



ω
$\mathcal{F}_f$
$\otimes$
ω
$\mathcal{F}_{\text{III}_T} = \text{III}_{\frac{1}{T}}$
$=$
ω
$\mathcal{F}_f \otimes \text{III}_{\frac{1}{T}}$

Fig. A.5.: Discretizing a signal in frequency space

A.2 Spatial and Frequency spaces

Any CT or DT signal can be expressed with many different forms. The ones that will interest us here are its spatial-representation and its frequency-representation. Those forms are strictly equivalent, and represent the same signal in a different space (space is to be understood here as a choice of referential basis and not as a geometric space). The spatial-representation of a signal is its expression in the spatial-space, which is the most common one. The frequency-representation of a signal is its expression in the frequency space, where the referential is composed of complex exponentials [BB86].

The frequency representation of a signal is often referred to as its *Fourier spectrum*. To transform a signal from its spatial to its frequency representation, we use the *Fourier transform*. We denote $\mathcal{F}_f$, the frequency representation of a spatial signal f, obtained from computing the Fourier transform of f. The inverse operation can be done using the *Inverse Fourier transform*. We denote $\mathcal{F}_g^{-1}$ the spatial representation of a frequency signal g, obtained from computing the Inverse Fourier transform of g. Note that $f = \mathcal{F}_{\mathcal{F}_f}^{-1}$. We will not get into much more detail here as this is explained thoroughly Section 2.2.1 on page 20.

Even though those representations are strictly equivalent, applying an operator onto a signal in spatial space is not equivalent to applying the same operator to this signal in frequency space.

The property that interests us here is that a product applied in the spatial space is equivalent to applying a convolution in the frequency space, and vice versa. More formally,

$$\begin{aligned} f \cdot g &= \mathcal{F}_f \otimes \mathcal{F}_g \\ \mathcal{F}_f \cdot \mathcal{F}_g &= f \otimes g, \end{aligned} \tag{A.5}$$

where $\otimes$ denotes the convolution operator.

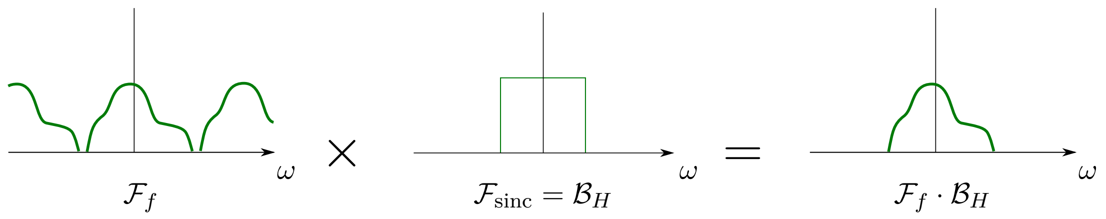

$\mathcal{F}_f$
ω
$\times$
$\mathcal{F}_{\text{sinc}} = \mathcal{B}_H$
ω
$=$
$\mathcal{F}_f \cdot \mathcal{B}_H$
ω

Fig. A.6.: Reconstructing a discretized signal in frequency space

This convolution operator performs the product of every value of a function f with every value of a function g

$$(f \otimes g)(t) := \int_{\Theta} f(\theta) g(t-\theta) d\theta, \tag{A.6}$$

where Θ is the support for the function f.

Furthermore, we will note that we have the following properties for the Dirac impulse and Dirac comb: a Dirac impulse in spatial space transforms to a Dirac impulse in frequency space, and similarly, transforming a Dirac comb only inverts the frequency of its set of impulses. More formally,

$$\delta = \mathcal{F}_{\delta} \tag{A.7}$$

$$\text{III}_T = \mathcal{F}_{\text{III}_{\frac{1}{T}}}. \tag{A.8}$$

This means that computing the product of a CT signal f with a Dirac comb III_T in the spatial space (to discretize it) resumes in the frequency space as a convolution of the Fourier transform of f with the Fourier transform of III_T. One of the consequence of this operation is that it repeats the signal creating *aliases* of the original signal in frequency space (Fig. A.5 on the facing page). It can be seen that in frequency space, if the frequency of the Dirac comb is too high (which means, if the frequency of the Dirac comb is too low in spatial space), the aliases will cover each other, leading to a different representation of the signal in the frequency space (Fig. A.10 on page 173). When reconstructing our signal, this will have a huge impact.

A.3 Reconstruction

In spatial space, we can reconstruct our signal by convolving it with a chosen reconstruction filter. This means, computing the product of the frequency representation of our signal with the frequency representation of our filter. As computing this product is much easier than performing a convolution, we will focus on what happens in the frequency space. We recall that we have a discretized version of our signal in frequency space, which

contains several aliases of the frequency representation of the original signal (Fig. A.5 on the facing page). Our goal is therefore to fully retrieve the original alias and only the original alias.

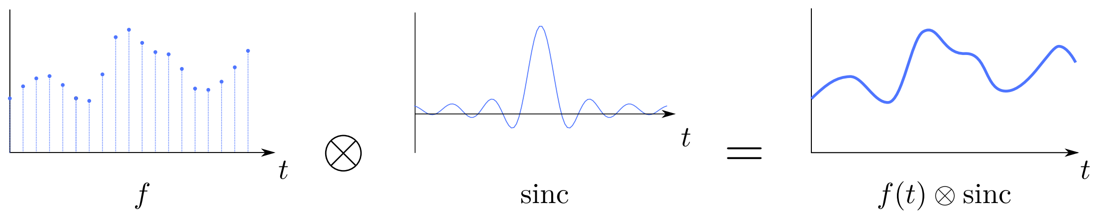


Fig. A.7.: Reconstructing a discretized signal in spatial space

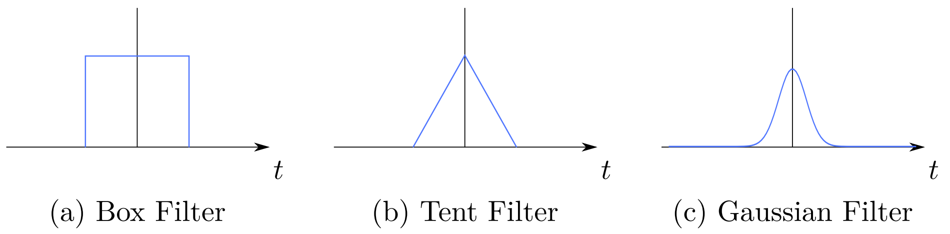


Fig. A.8.: Usual filters in their spatial representation

The easiest way to do so is to compute the product of our signal in frequency space with a box function $\mathcal{B}_H$ defined as:

$$\mathcal{B}_H(t) = \begin{cases} \frac{1}{H}, & \text{if } t \in [-\frac{H}{2}, \frac{H}{2}] \\ 0, & \text{otherwise .} \end{cases} \tag{A.9}$$

This lets us reconstruct the frequency signal from the original CT signal, which can be turned into the spatial original CT signal using the inverse fourier transform (Fig. A.6 on the previous page).

In spatial space, this means that we would have computed the convolution between a cardinal sinus function, $\text{sinc}(x) := \sin(x)/x$, whose frequency counterpart is the box function (Fig. A.7).

However, most of the time, we work in spatial space and as the cardinal sinus is a function of infinite support, it can thus hardly be used in practice as a reconstruction filter. Usual reconstruction filters in spatial space are a Box, Tent or Gaussian filter for the simplest ones. One also often uses a Mitchell-Netravali filter [MN88] as a more advanced solution.

The chosen reconstruction filter and the interval T chosen for the Dirac comb impacts greatly the consistency of the DT signal with its CT counterpart. The *Shannon and Nyquist theorem*

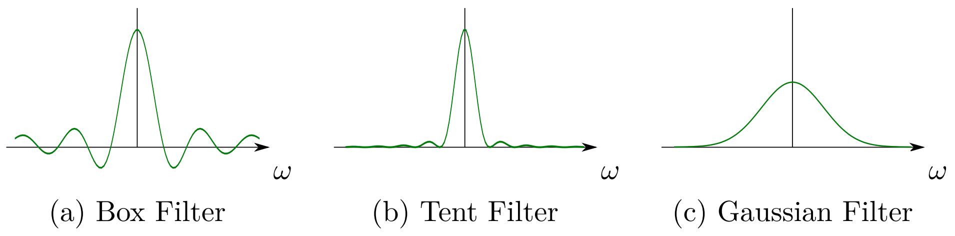
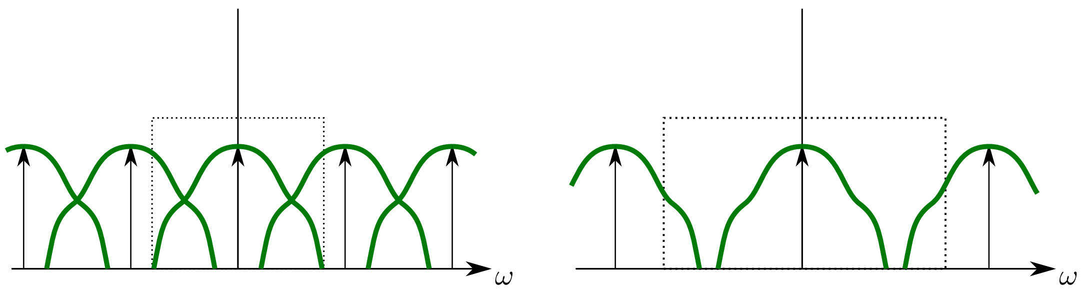


Fig. A.9.: Usual filters in their frequency representation



(a) $\mathcal{F}_f \otimes \text{III}_{\frac{1}{T'}} \cdot \mathcal{B}_H$ (b) $\mathcal{F}_f \otimes \text{III}_{\frac{1}{T}} \cdot \mathcal{B}_{H'}$

Fig. A.10.: Failure in the reconstruction process due to (a) undersampling or (b) poorly chosen filter

tells us that for all CT signals with a maximal frequency ω, their DT counterpart only allows a perfect reconstruction of the original CT signal if

$$T \geq 2\omega. \tag{A.10}$$

We call this limit the *Nyquist limit.* If a function has been discretized with $T < 2\omega$, it is said to be *undersampled.* It is easy to see that in frequency space, this causes the interval of the frequency Dirac comb to be too small; the aliases will cover each other, creating what are called *aliasing* artefacts. Aliasing might also occur on properly sampled signals, if the reconstruction process fails to capture correctly the Fourier spectrum of the original function (Fig. A.10).

All this theory allows proper reconstruction of a 2-D image from a set of discrete radiance values. However, computing those radiance values in itself is also a challenge, as detailed in Chapter 1 on page 1.

B Interactive Curvature Tensor Visualization

B.1 P.h.D. Context

In this appendix, we present a contribution we made in a very different field. During this PhD, we focused mainly on solutions to numerically approximate the rendering equation. However, we also worked on solutions to analytically simplify this equation, in particular cases when the surface follow a given statistical distribution. In those cases, the integration sums up as the CDF of the distribution characterizing the surface. Such methods did not give the expected results, but we were able to reuse this work for the particular case of digital curvature estimation.

In this appendix, we introduce what is curvature estimation of digital objects, and how thanks to the capacity of modern GPUs we could perform this computation in real time. Along with this, we also perform an on the fly triangulation of the digital object, with proper level of details, to ensure a good rasterization of the triangles (where there never are two triangles projecting into the same pixel).

B.2 Introduction

Volumetric objects are being more and more popular in many applications ranging from object modeling and rendering in Computer Graphics to geometry processing in Medical Imaging or Material Sciences. When considering large volumetric data, interactive visualization of those objects (or isosurfaces) is a complex problem. Such issues become even more difficult when dynamic volumetric datasets are considered. Beside visualization, we are also interested in performing geometry processing on the digital object and to explore different parameter settings of the geometry processing tool. Here, we focus on curvature tensor estimation (mean/Gaussian curvature, principal curvatures directions...). Most curvature estimators require a parameter fixing the scale at which the computation is performed. For short, such parameter (integration radius, convolution kernel size...) specifies a scale for analyzing the object surface, and is naturally related to the amount allowed perturbations on input data. As a consequence, when using such estimators, exploring different values of this parameter is mandatory. However, this is usually an offline process, due to the amount of computations that need to be done.

Contributions In this work, we propose a framework to perform interactive visualization of complex 3D digital structures combined with a dynamic curvature tensor estimation. We define a fully data parallel process on the GPU (Graphics Processor Unit) to both efficiently extract adaptive isosurface and compute per vertex curvature tensor using Integral Invariants estimators. This system allows us to visualize curvature tensor in real-time on large dynamic objects. Our approach combines a GPU implementation of pointerless octrees to represent the data, an adaptive viewpoint-dependent mesh extraction, and a GPU implementation of integral invariant curvature estimators.

Related works Extracting and visualizing isosurface on volumetric data has been widely investigated since the seminal Marching Cubes approach by Lorensen and Cline [LC87]. This approach being data parallel, GPU implementation of this method is very efficient [Tat+07]. However, such technique generates a lot of triangles which is not well suited to large digital data. Hence, adaptive approaches have been proposed in order to optimize the triangulated mesh resolution according to the object geometry or the viewpoint. In this topic, many solutions have been developed in Computer Graphics [Shu+95; SW04; LO10; Lew+10; Lob+14]. The method developed by Lengyel *et al.* [LO10] suits best our needs. It combines an octree space partitioning with new Marching Cubes configurations to generate adaptive meshes on CPU from static or near static data. We propose here a full GPU pipeline inspired by this method, that maintains a view dependent triangulation. Our high framerate allows us to inspect dynamic 3D data in real-time.

Curvature estimation on discrete or digital surface has also been widely investigated. In [Coe+14], authors propose digital versions of Integral Invariant estimators [Pot+07; Pot+09] in order to estimate the complete curvature tensor (mean/Gaussian curvatures, principal curvatures, principal curvature directions, normal vector field) on digital surfaces. Such approaches are based on an integration principle using a ball kernel of a given radius. Additionally, authors have demonstrated that these estimators have multigrid convergence properties. In this article, we also propose an efficient GPU implementation of such estimators to visualize such curvature fields in real-time and to interactively change the value of the kernel radius.

In this paper, we first present the previous works on curvature tensor estimation. Then, we propose an efficient GPU approach to extract a triangulated isosurface from a digital object. Finally, we propose a fully-parallel GPU pipeline to compute curvature tensor in real-time and present our results.

B.3 Curvature Tensor Estimation

In our context, we consider digital shapes (any subset Z of $\mathbb{Z}^d$) and boundaries of digital shapes $Bd(Z)$. We denote by $\mathtt{G}_h(X)$ the Gauss digitization of a shape $X \subset \mathbb{R}^d$ in a $d-$dimensional grid with grid step h, *i.e.* $\mathtt{G}_h(X) := \{\mathbf{z} \in \mathbb{Z}^d,\ h \cdot \mathbf{z} \in X\}$. For such digitized set $Z := \mathtt{G}_h(X)$, $[Z]_h$ is a subset of $\mathbb{R}^d$ corresponding to the union of hypercubes centered at $h \cdot \mathbf{z}$ for $\mathbf{z} \in Z$ with edge length h. By doing so, both ∂X and $\partial [Z]_h$ are topological boundaries of objects lying in the same space (see Fig. B.1 on the next page-b). Note that the combinatorial digital boundary $Bd(Z)$ of Z made of cells in a cellular Cartesian complex (*pointels*, *linels*, *surfels*, ...), can be trivially embedded into $\mathbb{R}^d$ such that it coincides with $\partial [Z]_h$.

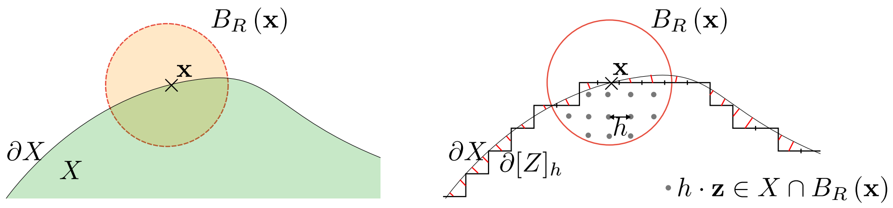


Fig. B.1.: Integral invariant computation (*left*) and notations (*right*) in dimension 2 [Coe+14].

In [Coe+13], authors define a 2D digital curvature estimator $\hat{\kappa}^R$ and a 3D digital mean curvature estimator $\hat{H}^R$ based on the digital volume estimator $\widehat{\mathrm{Vol}}(Y,h) := h^d\mathrm{Card}(Y)$ (area estimator $\widehat{\mathrm{Area}}(Y,h)$ in dimension 2):

Definition 1 *Given the Gauss digitization $Z := \mathtt{G}_h(X)$ of a shape $X \subset \mathbb{R}^2$ (or $\mathbb{R}^3$ for the 3D mean curvature estimator),* digital curvature estimators *are defined for any point $\mathbf{x} \in \mathbb{R}^2$ (or $\mathbb{R}^3$) as:*

$$\forall 0 < h < R, \quad \hat{\kappa}^R(Z,\mathbf{x},h) := \frac{3\pi}{2R} - \frac{3\widehat{\mathrm{Area}}(B_{R/h}(\mathbf{x}/h) \cap Z, h)}{R^3}, \tag{B.1}$$

$$\hat{H}^R(Z,\mathbf{x},h) := \frac{8}{3R} - \frac{4\widehat{\mathrm{Vol}}(B_{R/h}(\mathbf{x}/h) \cap Z, h)}{\pi R^4}. \tag{B.2}$$

where $B_{R/h}(\mathbf{x}/h)$ is the ball of digital radius R/h centered on digital point $(\mathbf{x}/h) \in Z$.

Such curvature estimators have multigrid convergence properties [Coe+13]: when the digital object becomes finer and finer, *i.e.* when the digitization step h tends to zero, the estimated quantities on $\partial[\mathtt{G}_h(X)]_h$ converges (theoretically and experimentally) to the associated one on ∂X in $O\left(h^{\frac{1}{3}}\right)$ for convex shapes with at least C^3-boundary and bounded curvature (setting $R := kh^{\frac{1}{3}}$ for some $k \in \mathbb{R}$).

In [Coe+14], authors have also defined 3D digital principal curvature estimators $\hat{\kappa}_1^R$ and $\hat{\kappa}_2^R$ on $Z \subset \mathbb{Z}^3$ based on digital moments:

Definition 2 *Given the Gauss digitization $Z := \mathtt{G}_h(X)$ of a shape $X \subset \mathbb{R}^3$,* 3D digital principal curvature estimators *are defined for any point $\mathbf{x} \in \mathbb{R}^3$ as:*

$$\forall 0 < h < R, \quad \hat{\kappa}_1^R(Z,\mathbf{x},h) := \frac{6}{\pi R^6}(\hat{\lambda}_2 - 3\hat{\lambda}_1) + \frac{8}{5R}, \tag{B.3}$$

$$\hat{\kappa}_2^R(Z,\mathbf{x},h) := \frac{6}{\pi R^6}(\hat{\lambda}_1 - 3\hat{\lambda}_2) + \frac{8}{5R}, \tag{B.4}$$

where $\hat{\lambda}_1$ and $\hat{\lambda}_2$ are the two greatest eigenvalues of the covariance matrix of $B_{R/h}(\mathbf{x}/h) \cap Z$.

The covariance matrix needs to compute digital moments of order 0, 1 and 2 (see Equation 20 of [Coe+14] for more details). These estimators are proven convergent in $O\left(h^{\frac{1}{3}}\right)$ when setting the ball radius h in $R = kh^{\frac{1}{3}}$, where k is a constant related to the maximal curvature of the shape, on convex shapes with at least C^3-boundary and bounded curvature [Coe+14]. Additionally, eigenvectors associated to $\hat{\lambda}_1$ and $\hat{\lambda}_2$ of the covariance matrix are principal curvature direction estimators $\hat{\mathbf{w}}_1^R$ and $\hat{\mathbf{w}}_2^R$. The smallest eigenvector corresponds to the normal direction $\hat{\mathbf{n}}^R$ at $\mathbf{x}$. Convergence results can be found in [Lac+16].

It has been shown that the radius of the ball depends on the geometry of the underlying shape. In [Lev+14], a proposal was made for a parameter-free estimation of the radius of the ball by analyzing the shape *w.r.t.* the local shape geometry using maximal digital straight segments of the digital boundary. In [Lev+15], these estimators have been analyzed in scale-space (for a range of radii) for a given digital shape. This allows to detect features of the shape thanks to the behavior of estimators on singularities. As a consequence, for all these integral invariant based approaches, we need to consider different ball radius which could be time consuming when implemented on CPU. We propose here a fully parallel implementation on GPU allowing us to change the radius and thus update the estimated quantities in real-time.

B.4 Isosurface Extraction on GPU

In this section, we detail the adaptive isosurface extraction algorithm. The proposed approach uses an octree representation of the input object on which an adaptive Marching Cube builds the isosurface efficiently. Such hierarchical representation of the object allows us to handle large datasets and to locally adapt the level of details *w.r.t.* the geometry or camera position. We first present the octree representation and then the isosurface extraction.

B.4.1 Linear Octree Representation

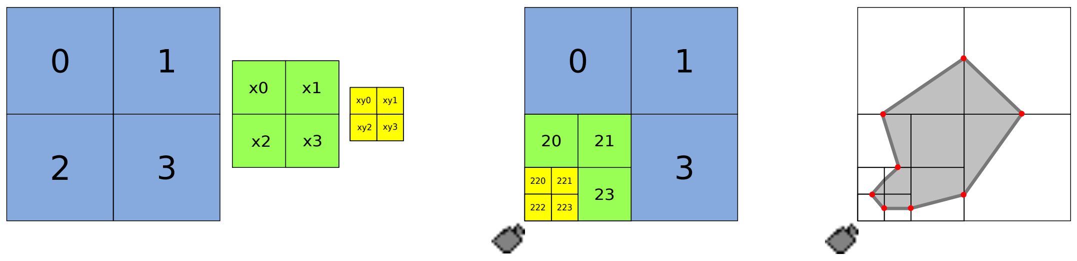


Fig. B.2.: Morton codes associated to cells of a linear quatree. Each morton code of a child cell is obtained by adding a suffix to its parent code *(left)*. The adaptive representation consists of quadtree cells whose depth is view point dependent *(middle)*. Finally, adaptive Marching Cubes is used to generate the triangulation *(right)*.

Representing a hierarchical structure on GPU is usually challenging since such a data parallel component is unable to handle recursivity. Efficient spatial tree encoding can be achieved using pointerless structures such as linear quadtrees or octrees Gargantini [Gar82]. This structure indexes each cell by a *Morton code*: the code of children cells are defined by the code of the parent suffixed by two bits (in dimension 2, three bits in dimension 3) (see Figure B.2-*left*). A cell's code encodes its position *w.r.t.* its parent cell and its complete path to the tree root. Hence, the tree is fully represented as a linear vector of its leaves. Furthermore, cell operations such as subdivision, merging can be efficiently implemented using bitwise operations on the

Morton code. In the following, we use the GPU friendly implementation proposed by Dupuy *et al.* [Dup+14].

B.4.2 Data parallel and adaptive mesh generation

Using this spatial data structure, a triangulated mesh can be constructed using Marching Cubes [LC87] (MC for short): the triangulation is generated from local triangle patches computed on local cell configurations. Such approach is fully parallel and easy to implement on the GPU. However, since adjacent cells may not have the same depth in the octree, original Lorensen and Cline's rules need to be updated (see Figure B.2 on the previous page-*right*). Many authors have addressed this problem both for primal and dual meshes [Shu+95; SW04; LO10; Lew+10; Lob+14].

In the following, we use the algorithm proposed by Lengyel *et al.* [LO10]. First, this approach forces the octree structure to make sure that the depth difference between any two adjacent cells is at most one. Then, Lengyel *et al.* introduce the concept of transition cells. Those cells are defined to be inserted between two neighboring octree cells of different depth. With specific MC configurations for their triangulation, a crack free mesh can be extracted.

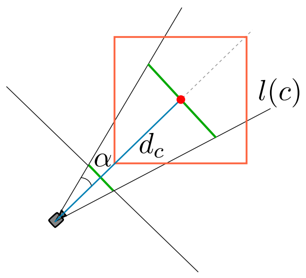
Similarly to original MC algorithm, this approach is well suited to a GPU implementation: given a set of cells (a vector of morton codes), each triangle patch can be extracted in parallel for both regular cells and transition cells.

B.4.3 Level of Details Criteria and Temporal Updates

As illustrated in Figure B.2 on the preceding page-*middle*, we propose a viewpoint dependent criterion to decide if a cell needs to be refined: the closer we are to the camera, the finer the cells are. Such a criterion is also well suited to GPU since it can be evaluated independently on every cell. Figure B.3 on the next page illustrates our level of details (LoD for short) criterion. In dimension 2, if α denotes the viewing angle, an object at a distance d from the camera has a projected size on screen of $2 \cdot d \cdot \tan(\alpha)$. (see Figure B.3 on the facing page-*left*). Our distance criterion is based on the ratio (*visibility ratio* in the following) between the cell diameter $l(c)$ (power of 2 depending on the depth), and its projected size. For a given cell c, split and merge decision are based on this visibility ratio:

- c is split if its children cells have a visibility ratio greater than constant k ;
- c and its sibling cells are merged if their parent cell c' has a visibility ratio lower than k ;
- otherwise, the cell c stays for the next frame.

Using such a criterion, split and merge decisions are computed in parallel from the morton codes of all cells.



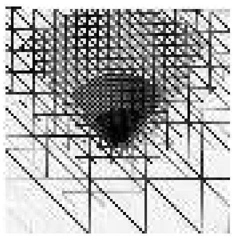


Fig. B.3.: Notations *(left)* and adaptive meshing in dimension 2 using the LoD distance and angular criterion *(right)*.

Once decisions have been made, a new set of cells is sent to the mesh generation step described above. Finally, before constructing the triangulation from remaining cells and transition cells, geometrical culling is performed in order to skip the triangle patch construction for cells that are not visible. Figure B.4 illustrates the overall fully data parallel pipeline.

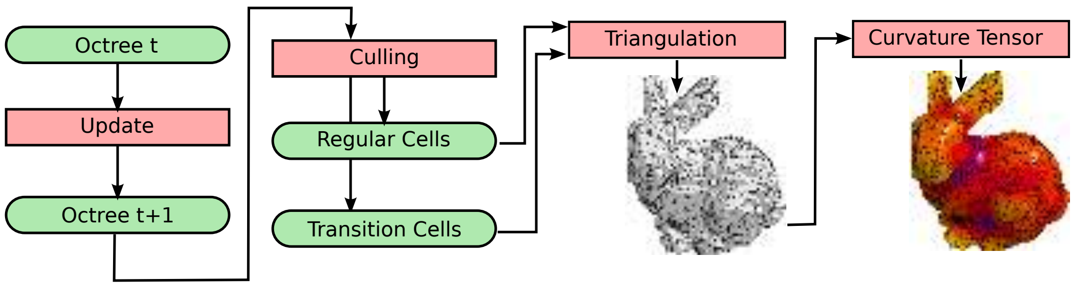


Fig. B.4.: GPU pipeline summary. Data buffers are represented in green and computations in red. Each computation retrieves data from a buffer and fills a new one.

B.5 Interactive Curvature Computation on GPU

We first present design principles for the GPU implementation and then present our approaches. The general idea is to perform an Integral Invariant computation on GPU at each vertex of the generated triangulated mesh. Since GPU have massively parallel architectures, we can do all those computations in parallel to obtain a very efficient implementation. Please note that when the LoD criterion is removed, each MC vertex is exactly centered at a surfel center. At different depth of the octree, MC vertices still correspond to surfel centers of subsampled versions of the input object. Hence, Integral Invariant framework defined in Section B.3 on page 175 is consistent with the triangulated surface obtained on GPU: triangles are used for visualization purposes but all computations are performed on the digital object $\mathtt{G}_h(X)$ for estimators (B.1 on page 176), (B.2 on page 176), (B.3 on page 176) and (B.4 on page 176).

To implement the integration on $B_R(\mathbf{x}) \cap X$ (Fig. B.1 on page 176), several strategies have been evaluated.

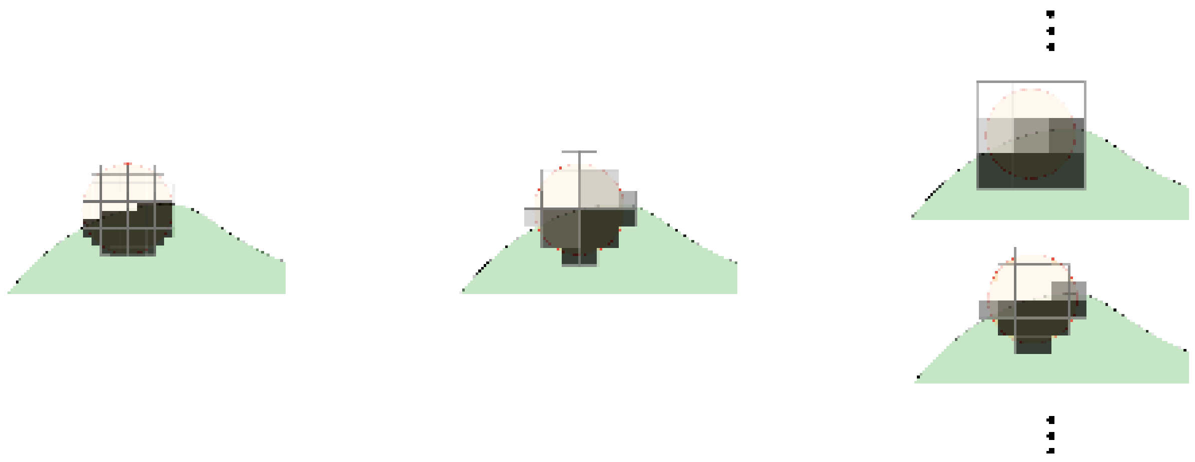


Fig. B.5.: Here are the three approaches we used to evaluate the integrand. (*left*) is the naive approach, where we sum all the digital points that are inside the integrand domain (*middle*) shows the hierarchical decomposition of the ball, thus limiting the number of texture probes to do. (*right*) illustrates the dynamic approach, we probe the textures at a higher resolution and we refine it at each frame until we reach the finer level.

B.5.1 Per Vertex Real-time Computation on GPU

Naive Approach. A first simple solution consists in scanning all digital points, at a given resolution, lying inside the integration domain (Fig. B.5-*left*) and then estimating the geometrical moment as the sum of the geometrical moments of each elementary cubes lying in the intersection (see Eq. (B.1 on page 176) to (B.4 on page 176)). This is exactly similar to what is done on the CPU. On the GPU, we can exploit *mipmap* textures to obtain multi-resolution information. If the input binary object is stored in a 3D texture, GPU hardware constructs a multi-resolution pyramid (mipmap) such that 8 neighboring voxel intensities at level l are averaged to define the voxel value at level $l+1$. If the level 0 corresponds to the binary input object, at a given level l, a texture probe at a point (x,y,z) returns the fraction of $\mathtt{G}_h(X)$ belonging to the cube of center (x,y,z) and edge length 2^l. As a consequence, we can approximate the volume of $B_R(\mathbf{x}) \cap X$ by considering mipmap values at a given resolution l (Fig. B.5-*right*). In this case, errors only occurs for cells lying inside X (with density 1) not entirely covered by $B_R(\mathbf{x})$. Furthermore, the *mipmap* texture can be used to design, using a single texture probe, a fast inclusion test of a given cell c at level l into the shapes: we say that c is in X if its density (retrieved from the texture probe at level l) is greater than 1/2. The idea here is to mimic a kind of adaptive Gauss digitization process.

Hierarchical Decomposition. Using the hierarchical nature of the mipmap texture, we could also consider hierarchical decompositions of $B_R(\mathbf{x})$. The idea is to decompose the ball into mipmap cells of different resolution in order to limit the number of texture access (important bottleneck on GPU hardwares) and to get better approximated quantities. On the GPU, computing a hierarchical decomposition of $B_R(\mathbf{x})$ at each point $\mathbf{x}$ is highly inefficient since the hardware optimizes the parallelism only when the micro-program described in the shader has a predictive execution flow. Hence, implementing the recursive algorithm (or its de-recursified version) requires a lot of branches (conditional *if* tests) in the flow. We have thus considered a fast multi-resolution approximation as illustrated in Fig. B.5-*middle*: for a given radius R, we precompute a hierarchical decomposition of B_R. Such decomposition is made of octree cells at different resolutions. At a given MC vertex $\mathbf{x}$, the algorithm becomes simple since we just scan the ball cells and estimate the area (or other moments) from the mipmap values associated to this cell. Note that these precomputed cells in the B_R decomposition may not be aligned with mipmap cells. However, we can use GPU which interpolates mipmap values if we probe at non-discrete positions. For a given radius R, the expected number of cells in B_R is in $O(\log R)$. Implementation details on the hierarchical octree representation can be found in the supplementary material (see Section B.7 on page 183).

Dynamic Refinement. In this last approach, we want to optimize the interactivity and the execution flow or parallelism on GPU. The integration is simply computed using a regular grid at different resolution l (Fig. B.5 on the preceding page-*right*). The shader code becomes trivial (simple triple loop) and a lot of texture fetches are involved, but interactivity can be easily obtained. Indeed, we consider the following multi-pass approach:

1. When the surface geometry has been computed, we set the current level l at an high level $l := l_{\max}$ of the mipmap pyramid.

2. We compute the integrals (and the curvature tensor) at the mipmap level l and send the estimated quantities to the visualization stage.

3. If the user changes the camera position or the computation parameters, we return to step 1.

4. If the current framerate is above a given threshold, we decrease l and return to step 2 if the final l_0 level has not been reached yet.

A consequence of the multi-pass approach is that when there is no interaction (camera settings, parameters), the GPU automatically refines the estimation. Even if step 2 is quite expensive, in $O\left(\left(\frac{R}{2^l}\right)^3\right)$, the interactive control in step 3 considerably improves the interactive exploration of the tensor with fast preliminary approximations which are quickly refined.

Compared to the hierarchical approach, no precomputation is required for a given radius R. As a consequence, we could even locally adapt the ball radius to the geometry (for instance following the octree depth of the current MC vertex). In the next section, we evaluate the performances of both approaches.

B.6 Experiments

B.6.1 Full resolution experiment

We first evaluate curvature estimations on a full-resolution geometry obtained by disabling the LoD criterion at the mesh generation step. Figure B.6 on the following page-top shows

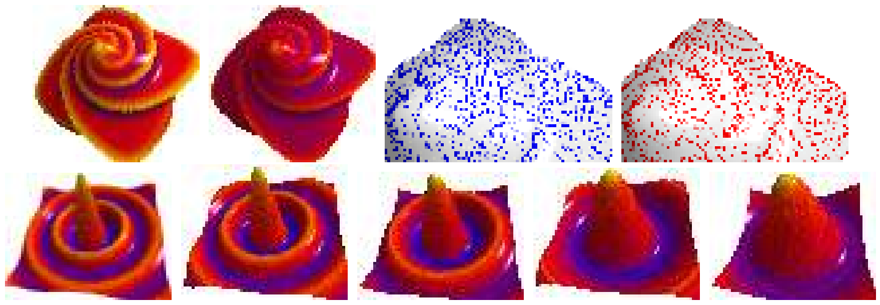


Fig. B.6.: *First row*: Mean and Gaussian curvature estimation, (zoom of) first and second principal directions estimation on “OctaFlower” with a digital domain of 130^3. *Second row:* Mean curvature computed in real-time (around 20 FPS) on a dynamic object.

results of curvature tensor estimation (mean, Gaussian, first and second principal directions) on “OctaFlower” at level l_0 (considered as our ground truth in the following). Figure B.6-bottom shows mean curvature estimation in real-time on a dynamic object. For this case, we simply evaluate the implicit expression at each vertex of a cell to decide if such cell is included or not into $B_R(\mathbf{x}) \cap X$ instead of updating the *mipmap texture* of densities at each time step. For all illustrations, we use directly normal vectors computed by algorithm discussed in Section B.3 on page 175.

Then, we compare the approximations made by computing curvature from level $l \geq l_0$ in Figure B.7 on the next page. We can see that results using approximation seems to quickly converge to ground truth results. Table B.1 on the facing page shows numerical results of L_∞ error (maximal absolute difference for all vertices) and L_2 error (mean squared errors of all vertices), as well the number of *mipmap texture* fetches. We can also see that the number of texel fetch, required to compute the curvature, reduces drastically when computing approximations. However, at higher levels, approximation errors become more important. Those levels are thus never used in practice, they are presented here to illustrate how the refining process converges to the l_0 level.

We also compare them with the hierarchical algorithm (as discussed in Section B.5.1 on page 180). This method introduces a higher error when compared with l_0. This is due to the precomputation of the subdivision that no longer ensures to fetch data at the center of a mipmap cell. The GPU interpolation reduces the bias, but it does not remove it.

B.6.2 Fully adaptive evaluation

When dealing with large datasets, we cannot expect real-time curvature tensor estimation if we consider the full resolution geometry, due to the huge amount of data to process. By computing a dynamically refined approximate curvature tensor estimation joined with an adaptive triangulation (as discussed in Section B.5.1 on page 180), we manage to maintain a real-time framerate by giving control over the amount of data to process at each frame. In Figure B.8 on page 184, we compare timings (in logarithmic scale) for a triangulation that is dynamically refined according to its distance to the camera. We measured those timings using the multiresolution regular and the hierarchical algorithm.

		H	l_4	l_3	l_2	l_1	l_0
Number of texture fetches	$R = 8$	1468		2	28	260	2104
	$R = 16$	5706	2	28	90	2120	17080
L_∞ error ($w.r.t.\ l_0$)	$R = 8$	0.051		0.306	0.085	0.047	0
	$R = 16$	0.053	0.146	0.041	0.017	0.005	0
L_2 error ($w.r.t.\ l_0$)	$R = 8$	3.51e-05		2.67e-04	7.57e-05	2.02e-05	0
	$R = 16$	4.43e-05	1.39e-04	4.16e-05	1.57e-05	2.07e-06	0

Tab. B.1.: Comparison of number of texture fetches, L_2 and L_∞ error obtained on "OctaFlower" with a digital domain of 130^3 when computing mean curvature with two radii: 8 and 16, with hierarchical algorithm (H) and $l \geq l_0$ approximation algorithms. The object is triangulated at full resolution with a regular grid and contains 282,396 vertices.

First we can note that the required time to compute the ground truth curvature for an object is usually as high as the time required to extract its geometry and to compute all of the above levels. Using approximations is thus mandatory. Another advantage with approximations is that it allows us to get a visualization of our object as soon as we run the application. This allows for real-time interactions with the object, required in order to change the visualized quantity, curvature radius, etc.

It is also visible in Figure B.8 on the following page that a hierarchical decomposition greatly reduces the curvature computation time, especially with big radii. However, due to the precomputation and the current hierarchical structure, this algorithm is biased and creates an error (presented in Table B.1) that needs to be considered. Figure B.9 shows curvature computation

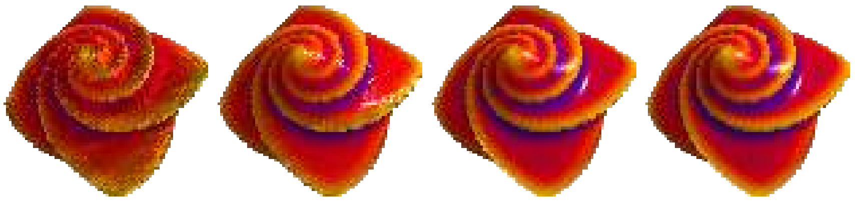


Fig. B.7.: Illustration of mean curvature computation on "OctaFlower" (digital domain of 130^3) using mipmap approximation with different levels: l_3, l_2, l_1 and l_0 (*i.e.* no approximation).

and exploration in real-time on large datasets: "XYZ-Dragon" (with digital domain of 512^3) and "Snow microstructures" (233^3).

B.7 Conclusion and Discussion

In this article, we have proposed a fully data parallel framework on GPU hardware which combines an adaptive isosurface construction from digital data with a curvature tensor estimation at each vertex. Using this approach, we can explore in real-time different curvature measurements (mean, Gaussian, principal directions) with different ball radii on potentially large dynamic dataset. Our proposal relies on both a linear octree representation with Morton codes and an efficient integral computation on GPU. The source code and additional material (video, ...) are available on the project website (https://github.com/dcoeurjo/ICTV).

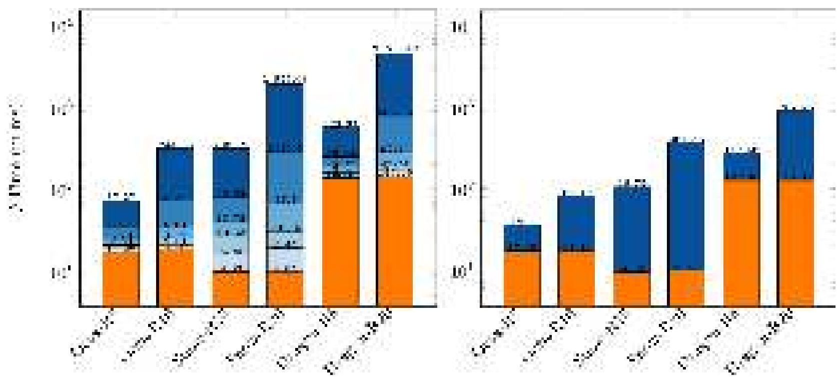


Fig. B.8.: Timings in milliseconds (in logscale) obtained while visualizing an adaptive triangulation on three objects – “OctaFlower” (digital domain of 130^3), “Snow microstructures” (233^3) and “XYZ-Dragon” (510^3) – by computing the curvature with a regular grid (*left*) and with hierarchical algorithm (*right*), with two different radii for each object. In *orange color*: time required to extract the triangulation. In *blue color*: time required to compute the curvature tensor at different levels: from l_4 (light blue) to l_0 (dark blue). Timings are given using a NVIDIA GeForce GTX 850M GPU.

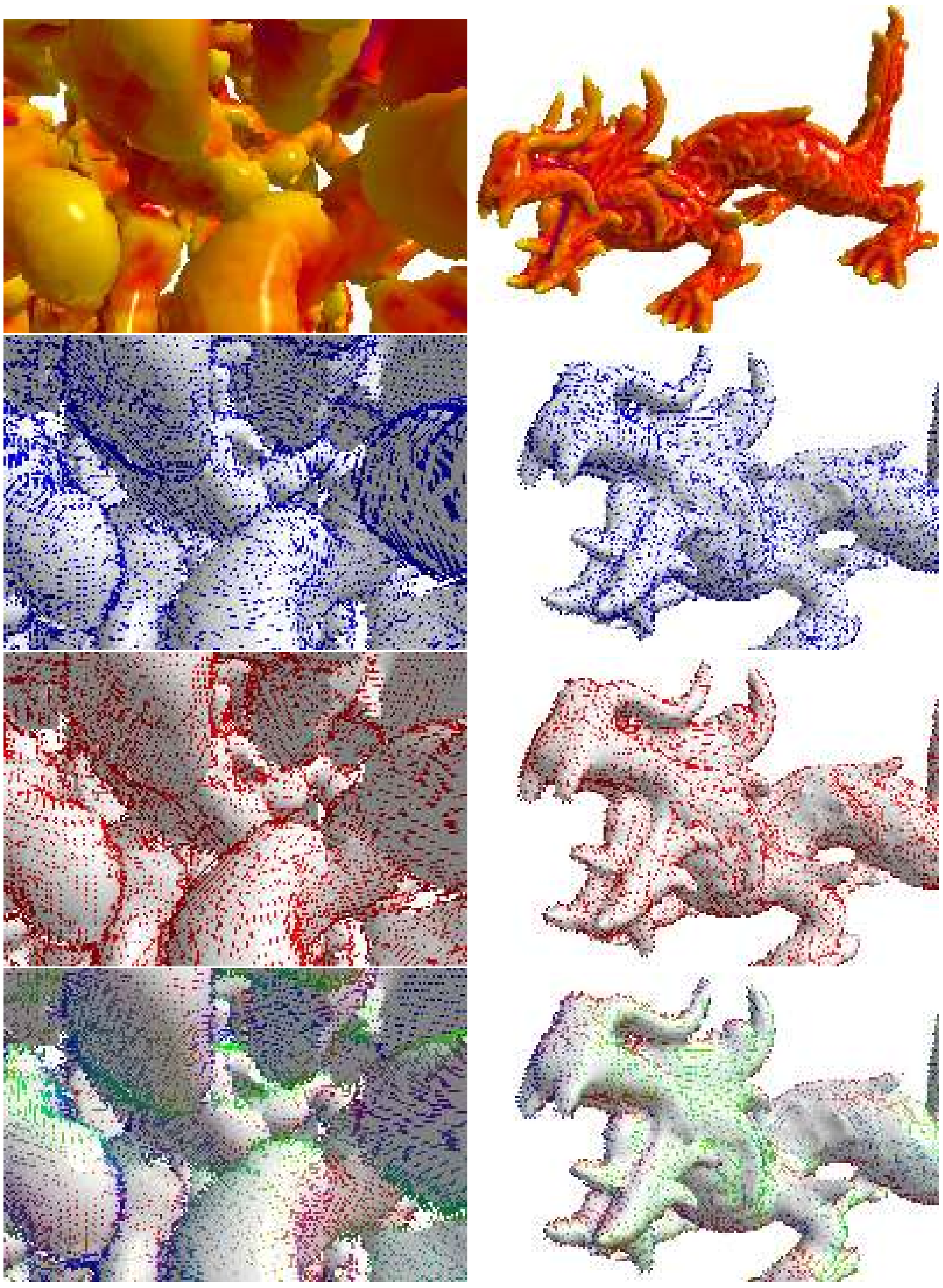


Fig. B.9.: *Left column*: Mean curvature, first and second principal directions and normal vector field estimation on “Snow microstructures” (233^3 and $R = 8$). *Right column:* Mean curvature, first and second principal directions and normal vector field estimation on “XYZ-Dragon” (510^3 and $R = 8$). Normal vectors are colored with a mapping of their component to RGB color space.

List of Figures

List of Tables

Bibliography

[Ahm+15] Abdalla GM Ahmed, Hui Huang, and Oliver Deussen. „AA patterns for point sets with controlled spectral properties“. In: *ACM Transactions on Graphics (TOG)* 34.6 (2015), p. 212 (cit. on pp. 51, 52, 155).

[Ahm+16a] Abdalla GM Ahmed, Jianwei Guo, Dong-Ming Yan, et al. „A simple push-pull algorithm for blue-noise sampling“. In: *IEEE transactions on visualization and computer graphics* (2016) (cit. on p. 48).

[Ahm+16b] Abdalla GM Ahmed, Hélene Perrier, David Coeurjolly, et al. „Low-discrepancy blue noise sampling“. In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), p. 247 (cit. on pp. 93, 103).

[Ahm+17] Abdalla Ahmed, Till Nies, Hui Huang, and Oliver Deussen. „An Adaptive Point Sampler on a Regular Lattice“. In: *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017* 36.4 (2017), 138:1–138:13 (cit. on p. 52).

[All+05] Pierre Alliez, Éric Colin De Verdière, Olivier Devillers, and Martin Isenburg. „Centroidal Voronoi diagrams for isotropic surface remeshing“. In: *Graphical models* 67.3 (2005), pp. 204–231 (cit. on p. 45).

[Bal+09] Michael Balzer, Thomas Schlömer, and Oliver Deussen. *Capacity-constrained point distributions: a variant of Lloyd's method*. Vol. 28. 3. ACM, 2009 (cit. on pp. 45, 46, 54).

[BB86] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York, 1986 (cit. on pp. 20, 170).

[BN12] Eric Bruneton and Fabrice Neyret. „A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading“. In: *IEEE Transactions on Visualization and Computer Graphics* 18.2 (Feb. 2012), pp. 242–260 (cit. on p. 3).

[Bow+10] 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)*. Vol. 29. 6. ACM. 2010, p. 166 (cit. on p. 43).

[Bri07] Robert Bridson. „Fast Poisson disk sampling in arbitrary dimensions.“ In: *SIGGRAPH sketches*. 2007, p. 22 (cit. on p. 43).

[CG12] Renjie Chen and Craig Gotsman. „Parallel Blue-noise Sampling by Constrained Farthest Point Optimization“. In: *Computer Graphics Forum*. Vol. 31. 5. Wiley Online Library. 2012, pp. 1775–1785 (cit. on p. 47).

[Cga] *The Computational Geometry Algorithms Library*. https://liris.cnrs.fr/ hperrier/utk/. Accessed: 2017-11-30 (cit. on p. 151).

[Che+12] Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. „Variational blue noise sampling“. In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1784–1796 (cit. on pp. 46, 134, 143, 155).

[Chi+94] Kenneth Chiu, Peter Shirley, and Changyaw Wang. „Multi-jittered sampling“. In: *Graphics gems IV* 4 (1994), p. 370 (cit. on p. 38).

[CM67] RR Coveyou and Robert D MacPherson. „Fourier analysis of uniform random number generators“. In: *Journal of the ACM (JACM)* 14.1 (1967), pp. 100–119 (cit. on p. 14).

[Coe+13] David Coeurjolly, Jacques-Olivier Lachaud, and Jérémy Levallois. „Integral based curvature estimators in digital geometry“. In: *Discrete Geometry for Computer Imagery.* Springer. 2013, pp. 215–227 (cit. on p. 176).

[Coe+14] David Coeurjolly, Jacques-Olivier Lachaud, and Jérémy Levallois. „Multigrid convergent principal curvature estimators in digital geometry“. In: *Computer Vision and Image Understanding* 129 (2014), pp. 27–41 (cit. on pp. 175, 176).

[Coh+03] Michael F Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. „Wang tiles for image and texture generation“. In: 22.3 (2003) (cit. on p. 50).

[Coo86] Robert L Cook. „Stochastic sampling in computer graphics“. In: *ACM Transactions on Graphics (TOG)* 5.1 (1986), pp. 51–72 (cit. on p. 40).

[Cor+17] Daniel Cornel, Robert F Tobler, Hiroyuki Sakai, Christian Luksch, and Michael Wimmer. „Forced Random Sampling: fast generation of importance-guided blue-noise samples“. In: *The Visual Computer: International Journal of Computer Graphics* 33.6-8 (2017), pp. 833–843 (cit. on pp. 52, 53, 155).

[Cor35] *Verteilungsfunktionen.* 1935 (cit. on p. 64).

[CP76] Roy Cranley and Thomas NL Patterson. „Randomization of number theoretic methods for multiple integration“. In: *SIAM Journal on Numerical Analysis* 13.6 (1976), pp. 904–914 (cit. on p. 75).

[Dcp] *Automatic differentiation in C++, Infinite differentiability of conditionals, loops, recursion and all things C++.* https://github.com/ZigaSajovic/dCpp. Accessed: 2017-12-08 (cit. on p. 83).

[DG+12] Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. „Blue noise through optimal transport“. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 171 (cit. on pp. 9, 47, 48, 71, 72, 80, 83, 93, 100, 103, 134, 155).

[DH06] Daniel Dunbar and Greg Humphreys. „A spatial data structure for fast Poisson-disk sample generation“. In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), pp. 503–508 (cit. on pp. 43, 44, 155).

[DK08] Sabrina Dammertz and Alexander Keller. „Image synthesis by rank-1 lattices“. In: *Monte Carlo and Quasi-Monte Carlo Methods 2006* (2008), pp. 217–236 (cit. on pp. 58, 156).

[DK17] Ken Dahm and Alexander Keller. „Learning Light Transport the Reinforced Way“. In: *arXiv preprint arXiv:1701.07403* (2017) (cit. on p. 166).

[Dob+96] David P. Dobkin, David Eppstein, and Don P. Mitchell. „Computing the Discrepancy with Applications to Supersampling Patterns“. In: *ACM Trans. Graph.* 15.4 (Oct. 1996), pp. 354–376 (cit. on pp. 19, 20).

[Du+99] Qiang Du, Vance Faber, and Max Gunzburger. „Centroidal Voronoi tessellations: Applications and algorithms“. In: *SIAM review* 41.4 (1999), pp. 637–676 (cit. on pp. 44, 45, 134, 143, 155).

[Dur11] Fredo Durand. „A frequency analysis of Monte-Carlo and other numerical integration schemes“. In: (2011) (cit. on pp. 4, 30).

[DW85] Mark A. Z. Dippé and Erling Henry Wold. „Antialiasing Through Stochastic Sampling“. In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 69–78 (cit. on p. 6).

[Ebe+11] Mohamed S Ebeida, Andrew A Davidson, Anjul Patney, et al. „Efficient maximal Poisson-disk sampling“. In: *ACM Transactions on Graphics (TOG).* Vol. 30. 4. ACM. 2011, p. 49 (cit. on p. 43).

[Ebe+12] 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.* Vol. 31. 2pt4. Wiley Online Library. 2012, pp. 785–794 (cit. on p. 43).

[Eld+97] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. „The farthest point strategy for progressive image sampling“. In: *IEEE Transactions on Image Processing* 6.9 (1997), pp. 1305–1315 (cit. on p. 46).

[Fat11] Raanan Fattal. „Blue-noise point sampling using kernel density model“. In: *ACM Transactions on Graphics (TOG).* Vol. 30. 4. ACM. 2011, p. 48 (cit. on pp. 47, 49, 134).

[Fau82] Henri Faure. „Discrépance de suites associées à un système de numération (en dimension s)“. In: *Acta Arithmetica* 41.4 (1982), pp. 337–351 (cit. on pp. 67, 134, 138, 156).

[Gar82] Irene Gargantini. „An effective way to represent quadtrees“. In: *Communications of the ACM* 25.12 (1982), pp. 905–910 (cit. on p. 177).

[GL07] Hardeep S Gill and Christiane Lemieux. „A search for extensible Korobov rules“. In: *Journal of Complexity* 23.4-6 (2007), pp. 603–613 (cit. on p. 58).

[Gla95] Andrew S Glassner. *Principles of digital image synthesis: Vol. 1.* Vol. 1. Elsevier, 1995 (cit. on pp. 3–5, 168).

[GM09] Manuel N Gamito and Steve C Maddock. „Accurate multidimensional Poisson-disk sampling“. In: *ACM Transactions on Graphics (TOG)* 29.1 (2009), p. 8 (cit. on pp. 27, 43).

[Gne+08] Michael Gnewuch, Anand Srivastav, and Carola Winzen. „Finding optimal volume subintervals with k points and computing the star discrepancy are NPhard“. In: *Journal of Complexity.* Citeseer. 2008 (cit. on p. 19).

[Gne+11] Michael Gnewuch, Magnus Wahlström, and Carola Winzen. „A randomized algorithm based on threshold accepting to approximate the star discrepancy“. In: *arXiv preprint arXiv:1103.2102* (2011) (cit. on pp. 18, 19).

[Grü+08] Leonhard Grünschloß, Johannes Hanika, Ronnie Schwede, and Alexander Keller. „(t, m, s)-Nets and Maximized Minimum Distance“. In: *Monte Carlo and Quasi-Monte Carlo Methods 2006* (2008), pp. 397–412 (cit. on pp. 84, 87, 88).

[Grü+12] Leonhard Grünschloß, Matthias Raab, and Alexander Keller. „Enumerating quasi-monte carlo point sequences in elementary intervals“. In: *Monte Carlo and Quasi-Monte Carlo Methods 2010.* Springer, 2012, pp. 399–408 (cit. on p. 78).

[Hal64] John H Halton. „Algorithm 247: Radical-inverse quasi-random point sequence“. In: *Communications of the ACM* 7.12 (1964), pp. 701–702 (cit. on pp. 64, 71, 72, 156).

[Ham60] John M Hammersley. „Monte Carlo methods for solving multivariable problems“. In: *Annals of the New York Academy of Sciences* 86.1 (1960), pp. 844–874 (cit. on pp. 68, 69, 156).

[Hec+13] Daniel Heck, Thomas Schlömer, and Oliver Deussen. „Blue noise sampling with controlled aliasing“. In: *ACM Transactions on Graphics (TOG)* 32.3 (2013), p. 25 (cit. on pp. 56, 93, 100, 103).

[Hes+10] Kerstin Hesse, Ian H Sloan, and Robert S Womersley. „Numerical integration on the sphere“. In: *Handbook of Geomathematics*. Springer, 2010, pp. 1185–1219 (cit. on p. 30).

[HH64] J. M. Hammersley and D. C. Handscomb. „Monte Carlo Methods.“ In: *Methuen & Co., London* (1964) (cit. on pp. 3, 5).

[HH97] Fred J Hickernell and Hee Sun Hong. „Computing multivariate normal probabilities using rank-1 lattice sequences“. In: *Proceedings of the Workshop on Scientific Computing (Hong Kong)*. 1997, pp. 209–215 (cit. on p. 58).

[Hic+00] Fred J Hickernell, Hee Sun Hong, Pierre L'Écuyer, and Christiane Lemieux. „Extensible lattice sequences for quasi-Monte Carlo quadrature“. In: *SIAM Journal on Scientific Computing* 22.3 (2000), pp. 1117–1138 (cit. on p. 58).

[Hic98] Fred Hickernell. „A generalized discrepancy and quadrature error bound“. In: *Mathematics of Computation of the American Mathematical Society* 67.221 (1998), pp. 299–322 (cit. on p. 20).

[Hil+01] Stefan Hiller, Oliver Deussen, and Alexander Keller. „Tiled blue noise samples“. In: *Vision, Modeling, and Visualization (VMV)*. 2001 (cit. on p. 50).

[Hla61] Edmund Hlawka. „Funktionen von beschränkter variatiou in der theorie der gleichverteilung“. In: *Annali di Matematica Pura ed Applicata* 54.1 (1961), pp. 325–333 (cit. on pp. 28, 57).

[Ill+08] Janine Illian, Antti Penttinen, Helga Stoyan, and Dietrich Stoyan. *Statistical analysis and modelling of spatial point patterns*. Vol. 70. John Wiley & Sons, 2008 (cit. on pp. 9, 26, 112).

[Jia+15] Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. „Blue noise sampling using an SPH-based method“. In: *ACM Transactions on Graphics (TOG)* 34.6 (2015), p. 211 (cit. on pp. 47, 48).

[JK08] Stephen Joe and Frances Y Kuo. „Notes on generating Sobol sequences“. In: *ACM Transactions on Mathematical Software (TOMS), 29 (1), 49* 57 (2008) (cit. on p. 65).

[Jon06] Thouis R Jones. „Efficient generation of Poisson-disk sampling patterns“. In: *Journal of Graphics Tools* 11.2 (2006), pp. 27–36 (cit. on p. 43).

[Kai+16] Bhavya Kailkhura, Jayaraman J Thiagarajan, Peer-Timo Bremer, and Pramod K Varshney. „Stair blue noise sampling“. In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), p. 248 (cit. on pp. 28, 56).

[Kan+11] Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita. „Deterministic blue noise sampling by solving largest empty circle problems“. In: *The Journal of the Institute of Image Electronics Engineers of Japan* 40.1 (2011), pp. 6–13 (cit. on p. 47).

[Kel06] Alexander Keller. „Myths of Computer Graphics“. In: *Monte Carlo and Quasi-Monte Carlo Methods 2004*. Ed. by Harald Niederreiter and Denis Talay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 217–243 (cit. on p. 29).

[Ken13] Andrew Kensler. „Correlated multi-jittered sampling“. In: *Pixal Technical Memo* 7 (2013), pp. 86–112 (cit. on pp. 38–40, 157).

[KN74] Lauwerens Kuipers and Harald Niederreiter. *Uniform Distribution of Sequences*. Pure and applied mathematics. New York: John Wiley & Sons, 1974 (cit. on pp. 17, 75, 99, 102).

[Kok42] JF Koksma. „Een algemeene stelling uit de theorie der gelijkmatige verdeeling modulo 1“. In: *Mathematica B (Zutphen)* 11.7-11 (1942), p. 43 (cit. on p. 28).

[Kop+06] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. „Recursive Wang tiles for real-time blue noise“. In: 25.3 (2006) (cit. on p. 50).

[Kor59] NM Korobov. „The approximate computation of multiple integrals“. In: *Dokl. Akad. Nauk SSSR*. Vol. 124. 6. 1959, pp. 1207–1210 (cit. on p. 58).

[Lac+16] Jacques-Olivier Lachaud, David Coeurjolly, and Jérémy Levallois. „Robust and Convergent Curvature and Normal Estimators with Digital Integral Invariants“. In: *Modern Approaches to Discrete Curvature*. Lecture Notes in Mathematics. Springer International Publishing, 2016, forthcoming (cit. on p. 177).

[LC87] William E Lorensen and Harvey E Cline. „Marching Cubes: A High Resolution 3D Surface Construction Algorithm“. In: *ACM Computer Graphics* 21.4 (1987) (cit. on pp. 175, 178).

[LD05] Ares Lagae and Philip Dutré. „A procedural object distribution function“. In: *ACM Transactions on Graphics (TOG)* 24.4 (2005), pp. 1442–1461 (cit. on p. 50).

[Lem09] Christiane Lemieux. *Monte Carlo and Quasi Monte Carlo Sampling*. Springer-Verlag New York, 2009 (cit. on pp. 14, 18, 29, 36, 57, 59, 62, 66, 76, 121).

[Lev+14] Jérémy Levallois, David Coeurjolly, and Jacques-Olivier Lachaud. „Parameter-free and Multigrid Convergent Digital Curvature Estimators“. In: *Discrete Geometry for Computer Imagery*. Springer. 2014, pp. 162–175 (cit. on p. 177).

[Lev+15] Jérémy Levallois, David Coeurjolly, and Jacques-Olivier Lachaud. „Scale-space Feature Extraction on Digital Surfaces“. In: *Computers and Graphics* (2015), p. 12 (cit. on p. 177).

[Lew+10] Thomas Lewiner, Vinícius Mello, Adelailson Peixoto, Sinésio Pesco, and Hélio Lopes. „Fast Generation of Pointerless Octree Duals“. In: *Comput. Graph. Forum* 29.5 (2010), pp. 1661–1669 (cit. on pp. 175, 178).

[Li+10a] Hongwei Li, Li-Yi Wei, Pedro V Sander, and Chi-Wing Fu. „Anisotropic blue noise sampling“. In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 6. ACM. 2010, p. 167 (cit. on p. 54).

[Li+10b] Hongwei Li, Diego Nehab, Li-Yi Wei, Pedro V Sander, and Chi-Wing Fu. „Fast capacity constrained Voronoi tessellation“. In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM. 2010, p. 13 (cit. on p. 46).

[Llo82] Stuart Lloyd. „Least squares quantization in PCM“. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137 (cit. on p. 44).

[LO10] Eric Stephen Lengyel and John D Owens. *Voxel-based terrain for real-time virtual simulations*. University of California at Davis, 2010 (cit. on pp. 175, 178).

[Lob+14] Ricardo Uribe Lobello, Florent Dupont, and Florence Denis. „Out-of-core adaptive iso-surface extraction from binary volume data“. In: *Graphical Models* 76.6 (2014), pp. 593–608 (cit. on pp. 175, 178).

[LS90] Boris D Lubachevsky and Frank H Stillinger. „Geometric properties of random disk packings“. In: *Journal of statistical Physics* 60.5-6 (1990), pp. 561–583 (cit. on p. 42).

[Mat98] Jiří Matoušek. „On the L2-discrepancy for Anchored Boxes“. In: *J. Complex.* 14.4 (Dec. 1998), pp. 527–556 (cit. on p. 19).

[MF92] Michael McCool and Eugene Fiume. „Hierarchical Poisson disk sampling distributions". In: *Proceedings of the conference on Graphics interface*. Vol. 92. 1992, pp. 94–105 (cit. on p. 42).

[Mit91] Don P Mitchell. „Spectrally optimal sampling for distribution ray tracing". In: *ACM SIGGRAPH Computer Graphics*. Vol. 25. 4. ACM. 1991, pp. 157–164 (cit. on p. 42).

[MN88] Don P Mitchell and Arun N Netravali. „Reconstruction filters in computer-graphics". In: *ACM Siggraph Computer Graphics* 22.4 (1988), pp. 221–228 (cit. on pp. 152, 172).

[MP92] Theophano Mitsa and Kevin J Parker. „Digital halftoning technique using a blue-noise mask". In: *JOSA A* 9.11 (1992), pp. 1920–1929 (cit. on p. 52).

[NBJ97] Jeffrey L Newbern and V Michael Bove Jr. „Generation of blue noise arrays by genetic algorithm." In: *Human Vision and Electronic Imaging*. 1997, pp. 441–450 (cit. on p. 52).

[Nie72] H. Niederreiter. „Discrepancy and convex programming". In: *Annali di Matematica Pura ed Applicata* 93.1 (1972), pp. 89–97 (cit. on p. 18).

[Nie88] Harald Niederreiter. „Low-discrepancy and low-dispersion sequences". In: *Journal of number theory* 30.1 (1988), pp. 51–70 (cit. on pp. 62, 67, 134, 156).

[Nie92] Harald Niederreiter. *Random Number Generation and quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992 (cit. on pp. 15, 18, 59, 63, 68).

[NX95] Harald Niederreiter and Chaoping Xing. „Low-discrepancy sequences obtained from algebraic function fields over finite fields". In: *Acta Arithmetica* 72.3 (1995), pp. 281–298 (cit. on p. 67).

[Oei] *Online Encyclopedia of Integer Sequences*. https://oeis.org/A058947. Accessed: 2017-11-30 (cit. on p. 65).

[OG12] A. Cengiz Öztireli and Markus Gross. „Analysis and Synthesis of Point Distributions based on Pair Correlation". In: *ACM Trans. Graph. (Proc. of ACM SIGGRAPH ASIA)* 31.6 (2012), to appear (cit. on pp. 27, 55, 84, 112).

[Ost+04] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. „Fast hierarchical importance sampling with blue noise properties". In: *ACM Transactions on Graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 488–495 (cit. on pp. 50, 51).

[Ost07] Victor Ostromoukhov. „Sampling with polyominoes". In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 78 (cit. on pp. 50, 51).

[Owe95] Art B Owen. „Randomly permuted (t, m, s)-nets and (t, s)-sequences". In: *Monte Carlo and quasi-Monte Carlo methods in scientific computing*. Springer, 1995, pp. 299–317 (cit. on pp. 74, 76, 77, 79, 100, 109, 110, 131, 156).

[Pha+16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016 (cit. on pp. 2, 35, 36, 38, 154).

[Pil+15] Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. „Variance Analysis for Monte Carlo Integration". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 34.4 (2015), 124:1–124:14 (cit. on pp. 30, 32, 40, 44, 56, 140, 142, 143, 163).

[Pot+07] H. Pottmann, J. Wallner, Y. Yang, Y. Lai, and S. Hu. „Principal curvatures from the integral invariant viewpoint". In: *Computer Aided Geometric Design* 24.8-9 (2007), pp. 428–442 (cit. on p. 175).

[Pot+09] H. Pottmann, J. Wallner, Q. Huang, and Y. Yang. „Integral invariants for robust geometry processing“. In: *Computer Aided Geometric Design* 26.1 (2009), pp. 37–60 (cit. on p. 175).

[Pur+94] Werner Purgathofer, Robert F Tobler, and Manfred Geiler. „Forced random dithering: improved threshold matrices for ordered dithering“. In: *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*. Vol. 2. IEEE. 1994, pp. 1032–1035 (cit. on p. 52).

[Ram+12] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. „A theory of monte carlo visibility sampling“. In: *ACM Transactions on Graphics (TOG)* 31.5 (2012), p. 121 (cit. on p. 30).

[Rei+15] Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. „Projective Blue-Noise Sampling“. In: *Computer Graphics Forum* (2015) (cit. on p. 48).

[Ron+11] Guodong Rong, Miao Jin, Liang Shuai, and Xiaohu Guo. „Centroidal Voronoi tessellation in universal covering space of manifold surfaces“. In: *Computer Aided Geometric Design* 28.8 (2011), pp. 475–496 (cit. on p. 45).

[Rov+17] Riccardo Roveri, A Cengiz Öztireli, and Markus Gross. „General Point Sampling with Adaptive Density and Correlations“. In: *Computer Graphics Forum*. Vol. 36. 2. Wiley Online Library. 2017, pp. 107–117 (cit. on p. 55).

[Sch+11] 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*. ACM. 2011, pp. 135–142 (cit. on pp. 47, 155).

[Sch+17] Christoph Schied, Anton Kaplanyan, Chris Wyman, et al. „Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination“. In: *Proceedings of High Performance Graphics*. ACM. 2017, p. 2 (cit. on p. 166).

[Sch72] Peter Schmidt. „Irregularities of distribution“. In: *VII. Acta Arith.* Citeseer. 1972 (cit. on p. 18).

[Sec02] Adrian Secord. „Weighted voronoi stippling“. In: *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. ACM. 2002, pp. 37–43 (cit. on p. 46).

[Sha10] Manan Shah. „A genetic algorithm approach to estimate lower bounds of the star discrepancy“. In: *Monte Carlo Methods and Applications* 16.3-4 (2010), pp. 379–398 (cit. on p. 19).

[Shi+91] Peter Shirley et al. „Discrepancy as a quality measure for sample distributions“. In: *Proc. Eurographics*. Vol. 91. 1991, pp. 183–194 (cit. on pp. 38, 39, 157).

[Shu+95] Renben Shu, Chen Zhou, and Mohan S Kankanhalli. „Adaptive marching cubes“. In: *The Visual Computer* 11.4 (1995), pp. 202–217 (cit. on pp. 175, 178).

[SK13] Kartic Subr and Jan Kautz. „Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration“. In: *To appear in ACM TOG* 32 (2013), p. 4 (cit. on pp. 4, 30).

[Sob67] Il'ya Meerovich Sobol'. „On the distribution of points in a cube and the approximate evaluation of integrals“. In: *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 7.4 (1967), pp. 784–802 (cit. on pp. 9, 15, 65, 71, 72, 79, 117, 119, 129, 134, 138, 156).

[SW04] Scott Schaefer and Joe Warren. „Dual marching cubes: Primal contouring of dual grids“. In: *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*. IEEE. 2004, pp. 70–76 (cit. on pp. 175, 178).

[T+12] Ritschel T, Dachsbacher C, Grosch T, and Kautz J. „The State of the Art in Interactive Global Illumination“. In: *Computer Graphics Forum* 31.1 (2012) (cit. on p. 2).

[Tat+07] Natalya Tatarchuk, Jeremy Shopf, and Christopher DeCoro. „Real-time isosurface extraction using the GPU programmable geometry pipeline“. In: *ACM SIGGRAPH 2007 courses*. ACM. 2007, pp. 122–137 (cit. on p. 175).

[Tez95] Shu Tezuka. *Uniform random numbers: Theory and practice*. Kluwer Academic Publishers, 1995 (cit. on p. 67).

[Thi01] Eric Thiémard. „An algorithm to compute bounds for the star discrepancy“. In: *journal of complexity* 17.4 (2001), pp. 850–880 (cit. on p. 19).

[TO03] Antonio Torralba and Aude Oliva. „Statistics of natural image categories“. In: *Network: computation in neural systems* 14.3 (2003), pp. 391–412 (cit. on p. 145).

[Tor+06] S Torquato, OU Uche, and FH Stillinger. „Random sequential addition of hard spheres in high Euclidean dimensions“. In: *Physical Review E* 74.6 (2006), p. 061308 (cit. on p. 44).

[Uli87] Robert Ulichney. *Digital halftoning*. MIT press, 1987 (cit. on p. 52).

[Uli88] Robert A Ulichney. „Dithering with blue noise“. In: *Proceedings of the IEEE* 76.1 (1988), pp. 56–79 (cit. on p. 31).

[Utk] *UTK Sampling Tool Kit*. https://www.cgal.org/. Accessed: 2017-11-30 (cit. on p. 133).

[Val+08] Sebastien Valette, Jean Marc Chassery, and Remy Prost. „Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams“. In: *IEEE Transactions on Visualization and Computer Graphics* 14.2 (2008), pp. 369–381 (cit. on p. 45).

[Vea97] Eric Veach. *Robust monte carlo methods for light transport simulation*. 1610. Stanford University PhD thesis, 1997 (cit. on pp. 2, 76).

[Wac+14] Florent Wachtel, Adrien Pilleboue, David Coeurjolly, et al. „Fast tile-based adaptive sampling with user-specified Fourier spectra“. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 56 (cit. on p. 51).

[War73] Tony Turner Warnock. „Computational Investigations of Low-discrepancy Point-sets.“ AAI7310747. PhD thesis. 1973 (cit. on p. 19).

[Wei08] Li-Yi Wei. „Parallel Poisson disk sampling“. In: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, p. 20 (cit. on p. 43).

[Wei10] Li-Yi Wei. „Multi-class blue noise sampling“. In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), p. 79 (cit. on p. 54).

[Whi+07] Kenric B White, David Cline, and Parris K Egbert. „Poisson disk point sets by hierarchical dart throwing“. In: *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on*. IEEE. 2007, pp. 129–132 (cit. on p. 43).

[WW11] Li-Yi Wei and Rui Wang. „Differential domain analysis for non-uniform sampling“. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 50 (cit. on pp. 23, 55, 93).

[Xia+11] Ying Xiang, Shi-Qing Xin, Qian Sun, and Ying He. „Parallel and accurate Poisson disk sampling on arbitrary surfaces“. In: *SIGGRAPH Asia 2011 Sketches*. ACM. 2011, p. 18 (cit. on p. 43).

[Xu+11] Yin Xu, Ligang Liu, Craig Gotsman, and Steven J Gortler. „Capacity-constrained Delaunay triangulation for point distributions“. In: *Computers & Graphics* 35.3 (2011), pp. 510–516 (cit. on p. 46).

[Yan+09] Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. „Isotropic remeshing with fast and exact computation of restricted Voronoi diagram“. In: *Computer graphics forum*. Vol. 28. 5. Wiley Online Library. 2009, pp. 1445–1454 (cit. on p. 45).

[Yan+14a] Dong-Ming Yan, Jianwei Guo, Xiaohong Jia, Xiaopeng Zhang, and Peter Wonka. „Blue-Noise Remeshing with Farthest Point Optimization“. In: *Computer Graphics Forum*. Vol. 33. 5. Wiley Online Library. 2014, pp. 167–176 (cit. on p. 47).

[Yan+14b] Dong-Ming Yan, Guanbo Bao, Xiaopeng Zhang, and Peter Wonka. „Low-resolution remeshing using the localized restricted Voronoi diagram“. In: *IEEE transactions on visualization and computer graphics* 20.10 (2014), pp. 1418–1427 (cit. on p. 45).

[Yan+15] Dong-Ming Yan, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka. „A survey of blue-noise sampling and its applications“. In: *Journal of Computer Science and Technology* 30.3 (2015), pp. 439–452 (cit. on p. 40).

[Yel83] John I Yellott. „Spectral consequences of photoreceptor sampling in the rhesus retina“. In: *Science* 221.4608 (1983) (cit. on p. 30).

[Zho+12] Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. „Point sampling with general noise spectrum“. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 76 (cit. on p. 55).

[Dup+14] Jonathan Dupuy, Jean-Claude Iehl, and Pierre Poulin. „GPU Pro 5“. en. In: A K Peters/CRC Press, Mar. 2014. Chap. Quadtrees on the GPU (cit. on p. 178).

[Özt16] A Cengiz Öztireli. „Integration with stochastic point processes“. In: *ACM Transactions on Graphics (TOG)* 35.5 (2016), p. 160 (cit. on pp. 30, 31).