universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Optimizing Point Distribution in a Unit Square Using Geometric Transformations

verfasst von | submitted by

Samaneh Hojati

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Computational Science
Betreut von | Supervisor:	Univ.-Prof. Torsten Möller PhD

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Torsten Möller, for his support, feedback, and patience in this process. I would also like to thank Aleksandar Doknic for his invaluable guidance, support, and encouragement throughout my research. A special thanks goes out to my family for their support and motivation during this journey.

Abstract

Distributing points within a defined space is an important problem across many fields, including geometry, simulations, and optimization. In particular, placing N points evenly within a multi-dimensional sampling space is useful for tasks such as numerical integration, machine learning, and computer graphics. It helps ensure uniform coverage of the space, reduces bias, and improves the accuracy of simulations and models [8].

Many existing methods struggle to distribute a random number of points while maintaining equal spacing. Some approaches use predefined templates, or random distributions, or rely on iterative adjustment methods to position points [6]. However, these methods often fail to distribute points evenly for any value of N, especially when N does not align with the method's specific limitations. Therefore, they are not always effective in achieving uniform sampling in all values of N.

Here, we show how to solve this problem in 2D by ensuring an even distribution of points within the sampling area. The approach begins by placing points on a Cartesian grid, with each point at an equal distance from its four closest non-diagonal neighbors. Then, to adjust it to any given value of N, we apply a combination of transformations such as rotation, scaling, and translation. These transformations allow the points to fit within the sampling space while maintaining uniform spacing for any desired N.

A key part of this work is not only the design of transformations that ensure coverage of any point distribution but also the development of a dashboard for visualization. The interactive dashboard allows users to enter different values of N and observe how the transformations modify the grid. We found patterns in the distributions that suggest ways to extend the approach to higher dimensions.

These findings suggest that transformation-based techniques can provide a reliable and flexible way to evenly distribute points in a constrained space. The ability to adjust and visualize point arrangements makes this method useful for applications where a precise point distribution is critical, such as computational modeling, optimization, and visualization. Future research will expand this approach to three- and higher-dimensional spaces to explore its broader applications.

Zusammenfassung

Das Verteilen von Punkten innerhalb eines definierten Raums ist ein wichtiges Problem in vielen Bereichen, wie Geometrie, Simulationen, und Optimierung. Insbesondere ist das gleichmäßige Platzieren von N Punkten in einem mehrdimensionalen Stichprobenraum nützlich für Aufgaben wie numerische Integration, maschinelles Lernen und Computergrafik. Es trägt dazu bei, eine gleichmäßige Abdeckung des Raums zu gewährleisten, Verzerrungen zu reduzieren und die Genauigkeit von Simulationen und Modellen zu verbessern [8].

Viele bestehende Methoden haben Schwierigkeiten, eine zufällige Anzahl von Punkten zu verteilen und dabei den Abstand gleich zu halten. Einige Ansätze verwenden vordefinierte Vorlagen oder zufällige Verteilungen, oder verlassen sich auf iterative Anpassungsmethoden, um Punkte zu positionieren [6]. Diese Methoden scheitern jedoch oft daran, die Punkte gleichmäßig für jeden Wert von N zu verteilen, insbesondere wenn N nicht den spezifischen Einschränkungen der Methode entspricht. Daher sind sie nicht immer effektiv, um eine gleichmäßige Stichprobe für alle Werte von N zu erreichen.

Hier zeigen wir, wie man dieses Problem lösen kann, indem man eine gleichmäßige Verteilung einer beliebigen Anzahl von Punkten innerhalb des Stichprobenbereichs sicherstellt. Der Ansatz beginnt damit, Punkte auf einem kartesischen Gitter zu positionieren, wobei jeder Punkt in gleichem Abstand zu seinen vier nächsten nicht-diagonalen Nachbarn platziert wird. Um es dann an einen beliebigen Wert von N anzupassen, wenden wir eine Kombination von Transformationen wie Rotation, Skalierung und Translation an. Diese Transformationen ermöglichen es, die Punkte innerhalb des Stichprobenraums zu platzieren, während der gleichmäßige Abstand für jedes gewünschte N beibehalten wird.

Ein wesentlicher Teil dieser Arbeit ist nicht nur das Design von Transformationen, die die Abdeckung jeder Punktverteilung gewährleisten, sondern auch die Entwicklung eines Dashboards zur Visualisierung. Das interaktive Dashboard ermöglicht es den Benutzern, verschiedene Werte von N einzugeben, und zu beobachten, wie die Transformationen das Gitter verändern. Wir haben Muster in den Verteilungen gefunden, die darauf hindeuten, dass der Ansatz auf höhere Dimensionen ausgeweitet werden kann.

Diese Ergebnisse legen nahe, dass transformationsbasierte Techniken eine zuverlässige und flexible Möglichkeit bieten können, Punkte gleichmäßig in einem begrenzten Raum zu verteilen. Die Fähigkeit, Punktanordnungen anzupassen und zu visualisieren, macht diese Methode nützlich für Anwendungen, bei denen eine präzise Punktverteilung entscheidend ist, wie z. B. bei der computergestützten Modellierung, Optimierungsproblemen, und Visualisierungen. Zukünftige Forschung wird sich darauf konzentrieren, diesen Ansatz auf dreidimensionale und höherdimensionale Räume auszudehnen, um seine breiteren Anwendungen zu erkunden.

Contents

List of Tables

List of Figures

1 Motivation

One of the main concepts in the field of Computational Science is sampling, the process of selecting a subset of data points from an area that is representative of the entire data set or space.

From numerical models to optimization problems, it is an important part of many different tasks [14]. In the same way, good sampling in data visualization is important for representing large datasets and ensuring that the visualizations are meaningful and understandable. Visualizations often have limited space, and sampling ensures that the displayed data fit within the screen while preserving patterns and relationships by removing noise or redundant information and reducing computational cost. In geographical analysis, achieving an ideal distribution of points improves the accuracy of modeling, sampling, and interpolation, thus reducing potential mistakes and biases [4]. An ideal distribution aims to place points so that they are evenly spaced throughout the sampling space to avoid gaps. This ensures that every region is well represented, improving the accuracy and reliability of numerical models and visualizations.

The problem addressed in this thesis is the distribution of N number of points that are equidistant within a 2D sampling space. In other words, the points must be evenly distributed across the sampling space, ensuring that no gaps remain in any region of the distribution. Gaps refer to cases where the distance between points near the edges and the sides of the sampling space is greater than the distance between points within the sampling space. In our study, the sampling space will be a unit square (a 1x1 square with coordinates from 0 to 1 on both sides). We chose this simple shape because:

- It gives us clear boundaries to work with (everything stays between 0 and 1)
- It makes calculations easier since we don't have to worry about different sizes
- Many other researchers use this same square, so we can compare our results [7]
- It's a good starting point before studying more complex shapes

This square is like a basic testing ground - if our methods work well here, they'll likely work for more complicated cases too.

At first glance, this problem may seem simple. However, when attempting to distribute N desired points, finding a solution for all possible cases becomes increasingly challenging. Although a straightforward solution exists for a complete square(N^2) number of points, different approaches are required for other values of N.

Existing point distribution methods, including uniform grid sampling and Latin hypercube sampling, often fail to yield straightforward solutions for equidistant point arrangements within constrained geometries. Other methods, such as *optimal covering points and curves* by Justin Tzou and Brian Wetton [16], do not generalize well for all N because they require specific computations and are computationally complex.

Naive approaches, such as fixed grids or random distributions, struggle to adapt to different sample sizes (N). Grids lead to uneven distributions when N does not fit the grid structure, causing gaps or overlaps [9]. For example, placing 8 points evenly in a square is harder than

placing 9 points, where a simple grid works perfectly. A 3×3 grid fits 9 points perfectly, but there is no even split for 8 points.

The issue of equidistant point distribution remains unsolved due to the limitations of current methodologies. Other methods also lack the capability to allow users to explore and modify configurations in real time, thus restricting their practical use.

This study seeks to fill the existing gaps by presenting a new methodology and interactive visualization. This study systematically explores geometric transformations, specifically rotation, scaling, and translation, to identify patterns for desire-point arrangements.

To solve this problem, our approach uses a Cartesian coordinate system to generate an initial grid in the unit square. It solves for some points, but since it has all possible sampling numbers, it applies a series of geometric transformations to adapt the grid based on the required number of points distributed in the unit square (Figure 1.1). Finding the combinations between different transformations is a difficult problem.

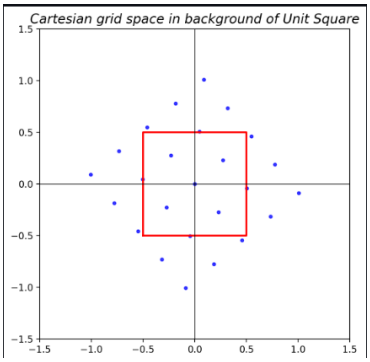


Figure 1.1: The Cartesian grid system is in the background of the Unit Square, and it shows as blue points

This research improves the field in several important ways:

- **Systematic Analysis of Geometric Transformations:** The application of rotation, scaling, and translation combinations in the Cartesian grid space to fit the desired point distributions in the unit square.
- **Building an interactive dashboard:** A simple tool has been made to show how changing transformation factors change objects and to let users see how setups work in real time. This dashboard makes it easier to show theoretical ideas with real-world applications.
- **Understanding limitations:** The research highlights instances where achieving satisfactory configurations is difficult, offering modification in step size.

The suggested technique presents considerable benefits; however, it has some limitations. This will be explained in the last chapter (Section 6.3).

In the next chapter, we will see the methodological approach, the methods used, and the steps applied to get the final solution.

2 Related Works

Recent studies have investigated methods for optimally distributing points within constrained spaces. The challenge of optimal point distribution has inspired diverse sampling approaches, each with distinct advantages. Various methods are used for point distributions, such as uniform grid sampling, Halton sequences, Sobol sequences, and Latin hypercube sampling. The mentioned studies suggest that Sobol-based methods are often the most effective for sampling in high-dimensional spaces, with LHS showing advantages in certain contexts. A detailed analysis and comparison of these methods is presented in the following sections.

2.1 Uniform Grid Sampling

Uniform Grid Sampling is a method that involves partitioning the sample space into uniformly distributed intervals in each dimension. Each point on the grid is selected at the intersection of the specified intervals, resulting in a uniform distribution of points throughout the sampling space. For uniform grid sampling, selecting a perfect square number of points (e.g., 16) aligns perfectly with the grid structure, ensuring optimal uniformity. However, here we deliberately choose 15 points to illustrate the limitations of uniform grid sampling when the number of points is not a perfect square. As shown in Figure 2.1, this causes a visible gap in the upper right corner, demonstrating uneven coverage.

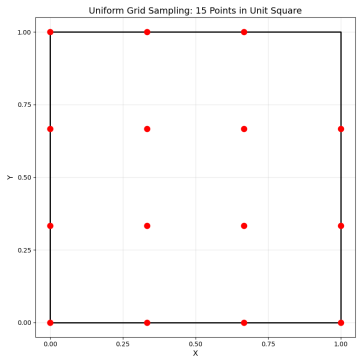


Figure 2.1: Uniform Grid Sampling with 15 Points in a Unit Square.

This straightforward approach requires minimal computational resources, making it suitable for low-dimensional contexts. However, Uniform Grid Sampling has several disadvantages. First, it suffers from the curse of dimensionality: as the number of dimensions increases, the number of grid points needed grows exponentially, leading to high computational costs and memory usage. Second, it lacks flexibility, as uniform grids may not adapt well to irregular or nonuniform distributions within the space, potentially missing important features or variations. Lastly, edge effects can occur when points on the grid boundary are less well represented than those in the interior, affecting the accuracy of analyses near the edges.

2.2 Halton Sequences

Halton sequences are widely utilized in quasi-Monte Carlo methods for multidimensional integration and simulation [2]. They are straightforward to implement and facilitate incremental sampling. However, the original Halton sequence exhibits correlations among dimensions, leading to suboptimal distribution in higher dimensions. To address this, researchers have proposed various modifications, including randomized or scrambled versions [2] and generalized Halton sequences incorporating permutations [5]. These modifications improve the uniformity and performance of the sequence, with comparative studies showing better results than the classical version in integration tasks [5].

Mathematically, the Halton sequence is a low-discrepancy sequence defined for dimension d using coprime bases $b_1, b_2, \ldots, b_d$ (usually prime numbers). The n-th point $\mathbf{x}_n = (x_{n,1}, \ldots, x_{n,d})$ is given by the radical inverse function $\varphi_{b_j}(n)$ in base b_j:

$$x_{n,j} = \varphi_{b_j}(n) = \sum_{k=0}^{\infty} a_k b_j^{-(k+1)}$$

where

$$n = \sum_{k=0}^{\infty} a_k b_j^k$$

is the base-b_j expansion of n.

The low discrepancy property of Halton sequences ensures that points are distributed evenly throughout the sampling space, which is crucial for producing datasets that are spatially representative. A dataset is spatially representative if its sample points effectively capture the spatial variability of the domain. This can be quantified by low discrepancy D_N, as with Halton sequences, or by satisfying a covering radius criterion:

$$r_{\text{cover}} = \sup_{\mathbf{y} \in [0,1]^d} \min_{1 \le i \le N} \|\mathbf{y} - \mathbf{x}_i\|$$

A small covering radius r_{cover} indicates that no point in the domain lies far from a sample point, reflecting good spatial representativeness.

Figure 2.2 demonstrates this characteristic spatial distribution for 15 points in a unit square.

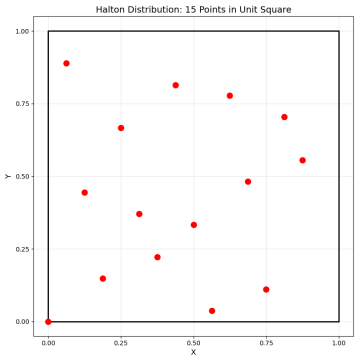


Figure 2.2: Halton Sequence Sampling with 15 points in a Unit Square.

Halton Sampling offers several advantages over Uniform Grid Sampling. First, Halton sequences can be easily extended to higher dimensions without suffering from the exponential

growth in points required by uniform grid sampling, making them more efficient in high-dimensional spaces. Second, unlike a uniform grid that requires a fixed resolution across the entire space, Halton sequences can be sampled incrementally. As a result, Halton sampling is better suited for irregular distributions and varying densities, allowing it to capture important features more accurately than rigid grids, which can either over-sample or under-sample specific regions.

However, there are also disadvantages. Halton sequences can exhibit correlation patterns, especially in lower dimensions or when starting from the beginning of the sequence, which may affect uniformity in certain applications. Implementing Halton sequences can be more complex than setting up a simple uniform grid, requiring careful handling to ensure the desired distribution properties are maintained. Additionally, while Halton sequences are efficient in covering space, they may introduce computational overhead compared to the straightforward calculation of grid points, especially if not optimized.

2.3 Sobol Sequences

Sobol sequences represent a low-discrepancy sampling method used in Monte Carlo techniques to perform uncertainty and sensitivity analyses [1]. A low-discrepancy sequence is a deterministic sequence of points designed to cover a space (e.g., a unit square) more uniformly than random sampling while avoiding the inflexible structure of a grid. These techniques generate optimally distributed points across the hypercube, offering significant advantages for parameter space exploration over traditional methods.

Mathematically, the Sobol sequence is constructed using direction numbers and bitwise operations to fill the space more uniformly than purely random points. Formally, the n-th point $\mathbf{x}_n = (x_{n,1}, x_{n,2}, \ldots, x_{n,d})$ in dimension d is generated by:

$$x_{n,j} = \bigoplus_{k=1}^{m} a_{j,k} v_k$$

where $\oplus$ is the bitwise XOR operator, $a_{j,k}$ are bits of n, and v_k are direction numbers specific to dimension j.

The sequence has low star-discrepancy, meaning it fills the space evenly and is useful for numerical integration.

Figure 2.3 demonstrates the Sobol sequence method applied to a unit square with 15 points. This method ensures a uniform distribution of points across the space, which is useful for numerical methods requiring low-discrepancy samples.

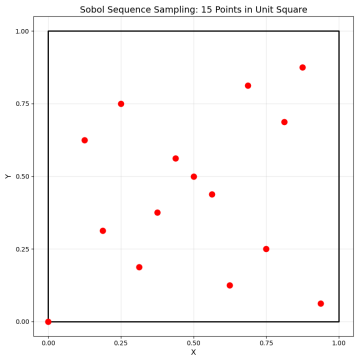


Figure 2.3: Sobol Sequence Sampling with 15 Points in a Unit Square.

Sobol sequences demonstrate superior performance compared to other sampling techniques in building simulation applications [1]. However, their effectiveness may be limited by the number of variables, as Sobol sampling performs optimally with up to 100 variables [3]. Hybrid approaches, such as Latin Hypercube Sampling (LHS) combined with Sobol sequences (LHS-SOBOL), have been developed to address limitations in large-scale optimization problems [3]. Additionally, scrambling techniques[1] are used to enhance Sobol sequences, and can be used to generate unbiased estimates and minimize variance in integration tasks [13].

2.4 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is a stratified sampling method[2] that is used to improve sample distribution. It efficiently covers the multidimensional sampling space of complex models [10]. Improves simple random sampling by ensuring a more uniform distribution of sampled points across the input domain [12]. LHS is particularly useful for computationally expensive simulations with numerous input variables, as it can provide valuable information even when only a few inputs dominate certain responses [10]. The method has been shown to have lower variance and normal distribution compared to simple random sampling, the degree of variance reduction depending on the additivity of the integrated function [12]. LHS can be adapted to handle statistically dependent input variables [12] and can be conditioned on additional information to maximize the stratification of multivariate distributions [11]. In general, LHS offers a more effective approach to replicating variable distributions compared to other sampling methods [11].

Key Steps

1. **Dividing the Space:** The range of each input parameter is partitioned into equal intervals (strata) along its axis.

2. **Sampling Points:** Within each dimension interval, a random point is selected, ensuring that each interval is sampled exactly once.

[1] Scrambling introduces controlled randomness into the sequence while preserving its low-discrepancy nature. This helps reduce systematic errors and improve convergence rates in high-dimensional spaces.

[2] "Stratified" refers to a method of sampling where the input space is divided into distinct, non-overlapping subregions or "strata." Each stratum is sampled independently, ensuring that the samples are spread throughout the range of each input variable.

3. **Combining Dimensions:** The sampled points across different dimensions are randomly combined to produce the final set of multidimensional samples.

LHS is widely used in fields such as optimization, sensitivity analysis, and uncertainty quantification because it produces quasi-random, yet structured, samples that provide comprehensive coverage of the input space [15].

Figure 2.4 demonstrates Latin Hypercube Sampling (LHS) applied to a unit square with 15 points. This method ensures that the points are evenly distributed across the parameter space.

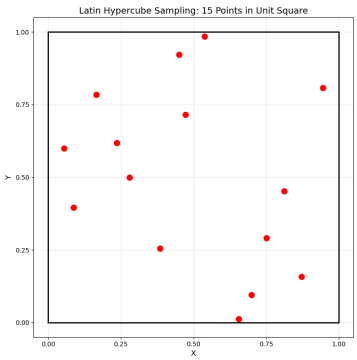


Figure 2.4: Latin Hypercube Sampling (LHS) with 15 Points in a Unit Square.

2.5 Gaps in Existing Methods

Established sampling methods, such as **Uniform Grid Sampling**, **Halton Sequences**, **Sobol Sequences**, and **Latin Hypercube Sampling**, are common; nevertheless, they include significant restrictions that limit their applicability in certain situations.

Uniform grid sampling, although straightforward and computationally efficient, experiences inadequate coverage when the number of points (N) does not correspond effectively to the grid configuration. For example, in Figure 2.1, trying to allocate 15 points within a square unit results in vacant areas in the upper right corner, which undermines regularity. These inconsistencies make this method less reliable for sampling in high-dimensional or irregular spaces. This issue arises not only for this method but also for all other methods in our list. When trying to place 15 points within the unit square, for example, it results in vacant areas, which disrupts the uniformity of the distribution.

Halton sequences also seek to distribute points quasi-randomly; however, the spatial intervals between points are frequently irregular. This irregularity is particularly evident in higher dimensions, where the approach may generate clusters or leave vacant regions within the sampling space. Figure 2.4 shows that although the points seem pseudorandom, some areas are conspicuously sparse, reducing coverage efficiency.

Sobol sequences, a prominent quasirandom technique, offer better uniformity than Halton sequences, particularly in lower dimensions. In low to moderate dimensions (e.g., 2 to 10), Sobol points fill the unit square evenly with minimal clustering or gaps. However, as dimensionality increases significantly (e.g., 50 or more), maintaining uniform coverage becomes more challenging without a sufficiently large number of sampled points. High-dimensional spaces require exponentially more points for even coverage, so limited samples may lead to underrepresented regions and reduced uniformity. Additionally, when the total number of points is small relative

to dimension, the sequence may fail to fully capture the complexity of the space, resulting in uneven coverage.

The limitations revealed in current methodologies underscore the necessity for a sampling technique that ensures equidistant points regardless of sample size and mitigates clustering or vacant areas. This study presents an enhanced interactive visualization tool for sampling. Our approach guarantees a uniform distribution with equidistant spacing of points throughout the domain, despite of the selected sample size (N), thereby addressing the empty-region and clustering problems associated with uniform grid sampling and quasi-random sequences.

The proposed sampling tool improves uniformity and consistency, thus boosting performance in applications such as optimization, sensitivity analysis, and uncertainty quantification.

2.6 Summary

Previous studies exhibit a variety of sampling methodologies, each possessing distinct advantages and disadvantages. However, these methods often fail to address issues such as irregular point distribution, vacant areas, and voids within the sampling region. This study expands upon these principles by highlighting uniform point distribution and facilitating the visual examination of transformation-based sampling methods, providing more detailed coverage of the sampled area.

This work also presents a dashboard application that enables users to visualize results immediately. The dashboard offers different configurations, allowing users to dynamically choose the number of points (N) and see how transformations modify the grid to encompass the selected N within the unit square. This interactive element provides a clear understanding of the sampling process and demonstrates the adaptability and accuracy of the proposed method, fulfilling an essential gap in previous methodologies.

In the next chapter, we present our methodological approach for optimal point distribution using Cartesian grid transformations (rotation, scaling, translation) while maintaining:

- Equidistant point spacing
- Complete unit square coverage
- Proper boundary alignment

Visual and mathematical analysis demonstrates how these operations achieve optimal configurations.

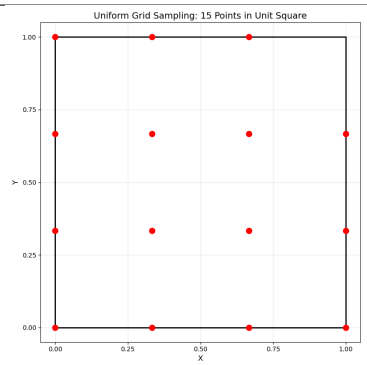
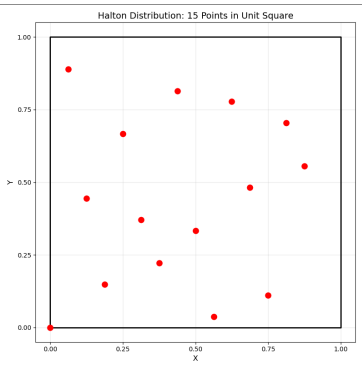
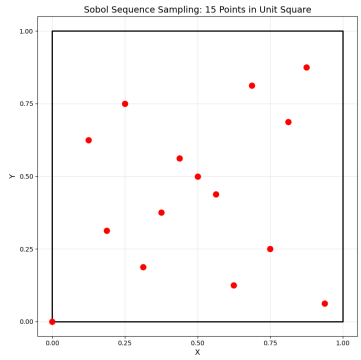
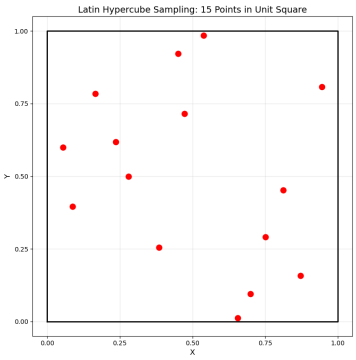
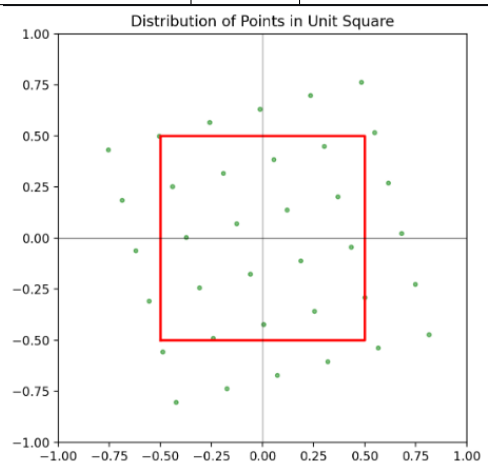
Comparison of distribution methods with plots			
Method	**Visualization**	**Method**	**Visualization**
Uniform Grid Sampling		Halton Sequences	
Sobol Sequences		LHS	
My Method			

Table 2.1: Comparison of Distribution Methods with Plots

3 Methodological Approach

In this work, we constrain ourselves to the 2D case and sample N points within the sampling space (specifically, a 2D unit square, as explained in previous chapters) by arranging them using a Cartesian grid coordinate system. This grid forms a larger square that shares the same center as the unit square and is large enough to always cover it. The number of grid cells can be adjusted based on the desired distribution of sampling points within the unit square.

The grid can rotate around this center; rotation alters the grid's orientation but does not change the distance between points. With each degree of rotation, the number of points within the unit square changes.

Scaling is another transformation applied to the grid. It allows the grid to expand (but not contract, as it must remain larger than the sampling space to avoid empty corners). As the scale factor increases, the distance between the points also increases. During scaling, the Cartesian grid expands while maintaining its center.

Translation is the final transformation. It involves shifting the Cartesian grid in any direction, provided it remains within the unit square.

In our study, we initially define the Cartesian grid with zero degrees of rotation, a scaling factor of 1, and zero translations. From this starting point, we apply all transformations together to follow the rule below:

1. **Equidistance:** The distance between any point and its direct adjacent points, the points in its right, left, above, or below, remains fixed throughout the square. That is why we derived the points from a Cartesian grid system. Diagonal adjacent points, located at the corners relative to a point, are also part of the configuration, but their distances are $\sqrt{2}$ times the distances to the direct adjacent points.

2. **Full Coverage:** The points must always cover the whole area of the unit square, leaving no empty regions or gaps in the corners, unlike the example shown in Figure 3.1. Transformations on the Cartesian grid are predefined in a table and tested before application. These transformations always cover the unit square, leaving no uncovered corners.

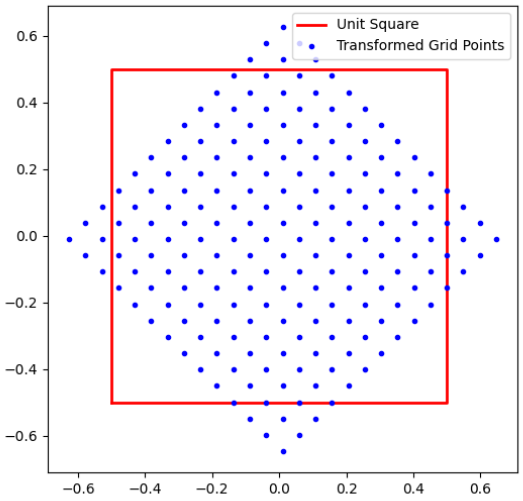


Figure 3.1: Uncovered space in the corners of the unit square is not permitted.

3. **Boundary Alignment:** The distance between points and the borders of the unit square must be less than or equal to the inter-point distance. This ensures that the distance of the points from the edges of the unit square is less than or equal to the distance between the diagonal points inside the square.

To achieve these conditions, the Cartesian grid is subject to the following transformations:

- **Rotation:** The grid can rotate around the center of the unit square to align with its boundaries, while ensuring full coverage. When the grid rotates around the center of the unit square, the scaling factor must be adjusted accordingly. This ensures full coverage of the unit square, since rotating the grid without scaling would leave empty spaces in the corners.

  Since the Cartesian grid in the background of our work is subject to transformations and its dimensions must be modified to maintain coverage of the unit square, we defined a limited construct for it. Therefore, both the scaling and the rotation must be modified together to always guarantee that the unit square is fully covered (Figure 3.2).

- **Scaling:** The grid can scale from a unit square (scale factor 1) and expand to fulfill the desired number of points, ensuring that no corners are uncovered. Scaling down (less than 1) is not allowed, as it would cause a loss of coverage at the square's corners.

- **Translation:** The grid can shift horizontally or vertically, allowing for finer adjustments to achieve an optimal configuration.

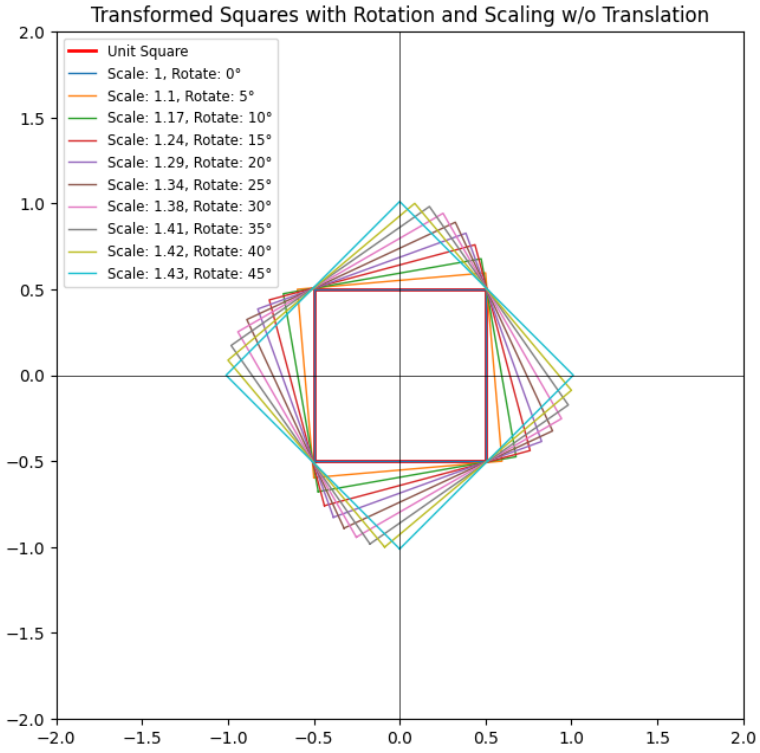


Figure 3.2: A Cartesian grid system is represented by a square that surrounds the unit square and can be transformed. The borders of this square define the grid's boundaries, within which the grid points are distributed.

- **Size of the grids in the Cartesian grid system:** For each given N number of points to be distributed within the unit square, a starting point (with N number of grids in the Cartesian grid) is considered. It should be the first complete square point. The grid will undergo all transformations defined in the step size. If the desired result does not appear with the current step size, the step size will be reduced, and the process will proceed to the next level of transformations.

In the next chapters, we will discuss in greater detail the algorithms and the implementation of this methodology within the dashboard. This will include an overview of the computational techniques employed and how they interact within the dashboard environment to facilitate user engagement and data analysis.

4 Algorithm Development and Code Implementation

This chapter provides algorithms for evenly distributing the desired number of points within the unit square and for scaling, rotating, and translating the Cartesian grid so that it is centered on the unit square.

The numerical approach developed in this work focuses on the mathematical foundations for positioning a Cartesian grid system to cover a unit square.

In the following sections, the mathematical foundations and computational techniques are explained.

4.1 Algorithm Description

The algorithm distributes N points within a unit square by iteratively refining a Cartesian grid system (Figure 4.1). The steps are as follows:

1. **Initialization:**
   - Define all possible transformations (rotation, scaling, translation).
   - Compute the smallest perfect square greater than N, $n' \geq N$.
2. **Grid Generation:**
   - Generate a Cartesian grid with n' points.
   - Apply all pre generated transformations (Section 4.2) to the grid.
3. **Point Counting:**
   - For each transformed grid, count the points within the unit square.
   - If the count matches N, accept the solution and terminate.
4. **Iterative Refinement:**
   - If N is not met, increment n' to the next perfect square.
   - Repeat steps 2–3 until N is achieved or 10 iterations elapse.

Iteration Limit

The algorithm includes a limit of **10 iterations** to ensure finite execution and improve computational performance.

For example $N = 5$, after 10 iterations the grid reaches $12 \times 12 = 144$ points. If no solution is found by this point, the desired N is considered less possible to achieve.

The limit of 10 iterations is a safety mechanism that ensures the algorithm terminates quickly while still searching through a sufficiently large range of grid sizes to find solutions if they exist.

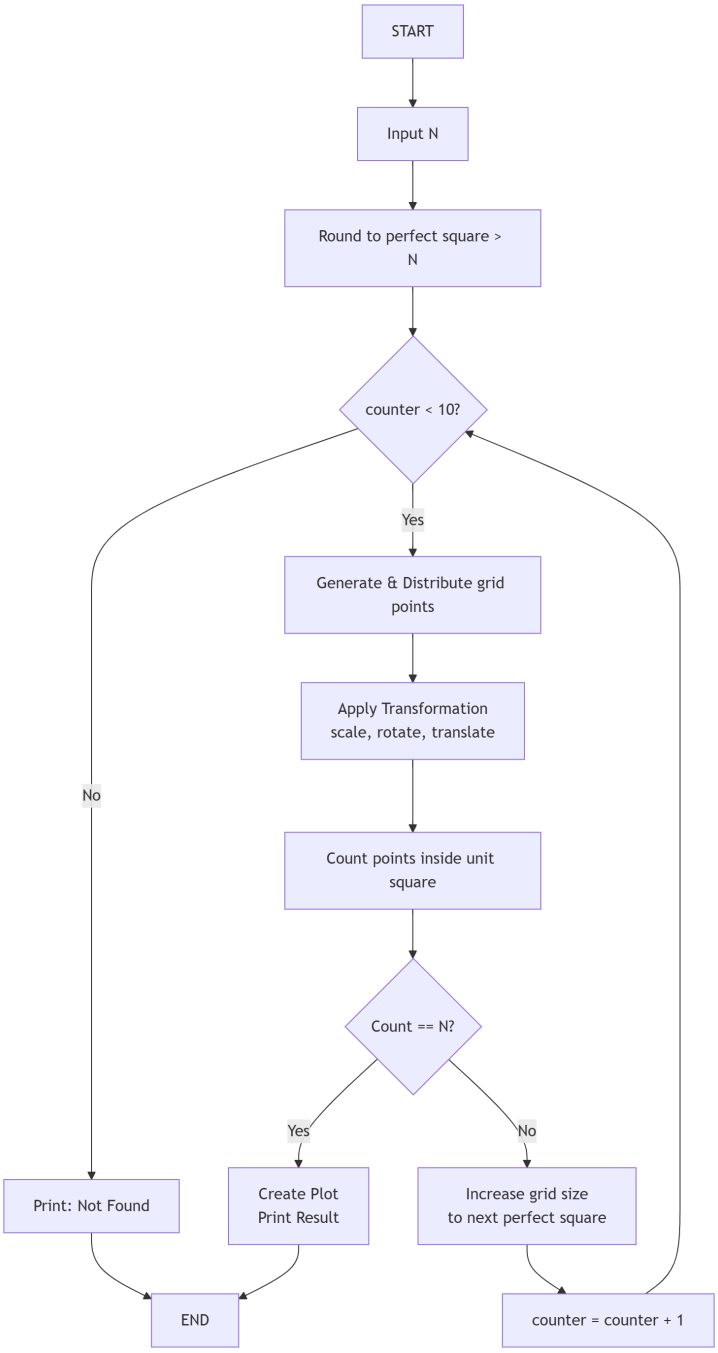


Figure 4.1: Flowchart of the point distribution algorithm.

4.2 Transformations and Coverage Criteria

The Cartesian grid system, with applied transformations to a square, is designed to:

1. Have a **limited size** such that it fully covers the unit square without exceeding its boundaries.

2. Share the **same center** as the unit square, allowing it to rotate around this central point.

3. Begin with a **rotation degree of 0** and increase in different step sizes.

4. Begin with a **scaling factor of 1** and increase in size as needed.

5. Begin with a **translation vector of 0** and can decrease or increase. The x and y translations are independent and not necessarily equal. This shifts the Cartesian grid square in any direction, and the rule is to avoid cutting the unit square.

6. Allow **translations** in the up, down, left, and right directions.

The proposed algorithm guarantees that the Cartesian grid system undergoes transformations based on geometric elements while satisfying specific requirements, as shown in Figure 4.2.

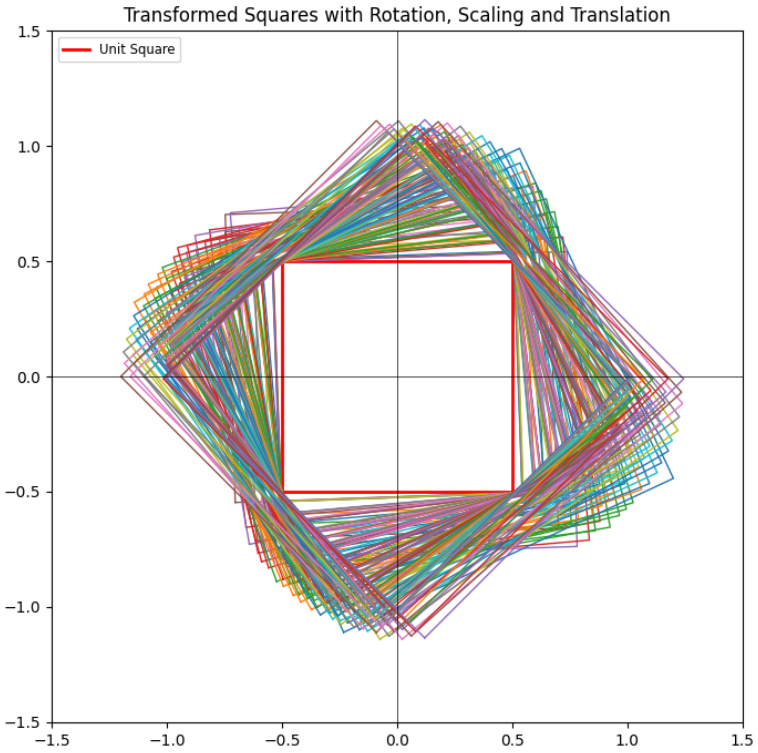


Figure 4.2: Cartesian grid squares with different transformations are located around the unit square

The combination of these transformations should ensure that the grid fully covers the unit square, avoiding cutting or exceeding its sides. These transformations enable the grid to dynamically adjust its alignment, size, and position while maintaining its structural integrity. Importantly, the unit square remains **fixed** throughout the process, and all transformations are applied exclusively to the Cartesian grid system.

4.3 Equation of Transformation

The Cartesian grid system is a two-dimensional coordinate system in which points are defined by their (x, y) coordinates. The grid transformations include:

1. **Rotation:** The grid can be rotated at an angle θ (in degrees) using a rotation matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

2. **Scaling:** To adjust the size of the grid, we apply a scaling factor s:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s \begin{pmatrix} x \\ y \end{pmatrix}.$$

3. **Translation:** The grid can be translated (shifted) to align within the unit square by adding offsets t_x and t_y:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}.$$

To apply the rotation, scaling, and translation transformations together, we combine them into a single transformation matrix using homogeneous coordinates. This allows us to perform all transformations in a single step via matrix multiplication, which is convenient for computation. The combined transformation matrix is given by:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Here, the parameter s represents the scaling factor, θ is the rotation angle, and t_x, t_y are the translation offsets. The use of homogeneous coordinates enables us to represent all transformations in a unified matrix form, making it easier to apply sequential operations.

The parameters are generated for different intervals and used within the main code. The intervals are defined on the basis of the rotation degree, and other elements are determined accordingly. We will refer to these transformations as pregenerated transformations from now on and deep dive into them in the following.

To ensure that the points remain within the unit square, the following constraints are applied:

$$-0.5 \leq x' \leq 0.5, \quad -0.5 \leq y' \leq 0.5.$$

The analytical solution involves determining the values of θ, s, t_x, and t_y that satisfy these constraints while ensuring that the required number of points remains within the unit square.

4.3.1 Handling Missing Solutions

During execution of the algorithm, it was observed that there are no solutions for certain values of N. For example, as shown in Table 4.1, a rotation degree increase of 5° produced rotation angles $5°, 10°, 15°, \ldots, 45°$, together with the corresponding adjustments in other transformations. This led to gaps in the results for some distributions of points in the unit square. To address this issue, we adjusted the transformation step size.

Scaling Factor	Rotation Degree	Translation X	Translation Y
1.1	5	0	0
1.18	10	0	0
1.25	15	0	0
1.31	20	0	0
1.34	25	0	0
1.4	30	0	0
1.42	35	0	0
1.43	40	0	0
1.43	45	0	0

Table 4.1: Transformations with Scaling and Rotation (No Translations)

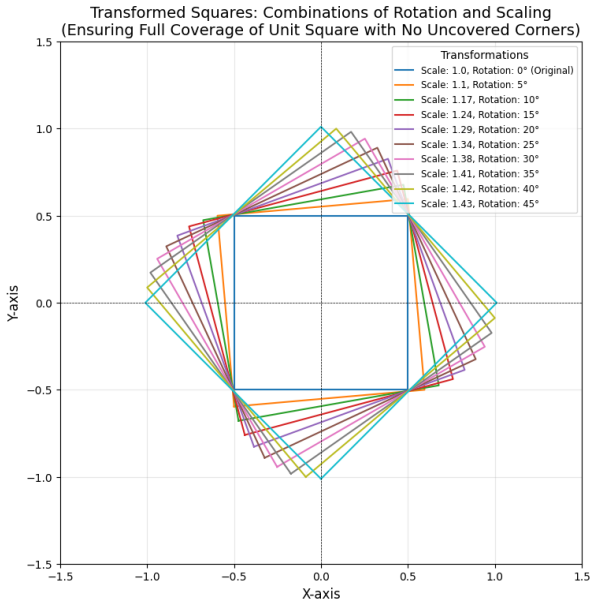


Figure 4.3: Transformations with Scaling and Rotation

To address this, finer increments were introduced between the existing rotation degrees, increasing the granularity of the rotational transformations. By adding intermediate rotation degrees, additional transformations were generated, allowing solutions to previously unsolvable values of N. The complete set of all transformations used in this work to fit the required points inside the unit square is provided in the Appendix (Chapter 7).

When the range of transformations becomes smaller, the probability of achieving the desired number of points within the unit square increases.

However, this adaptive strategy introduces a trade-off. Although a smaller transformation range improves the probability of success, it also increases computational time, especially for

larger point sets. The algorithm takes longer to converge as the number of points increases. For example, in the case of $N = 3$, no solution was found initially, necessitating the addition of supplementary transformations.

The proposed algorithm, supported by pregenerated transformations, provides a complete solution to distribute a specified number of points within the unit square. By applying transformations such as rotation, scaling, and translation, the Cartesian grid system adapts dynamically while preserving its structural integrity. The inclusion of finer rotational increments further enhances the algorithm's capabilities; however, it slows down the runtime.

This methodology can be extended to other geometrical configurations, offering a versatile tool for advanced computational and analytical studies.

4.4 Functions and Code Development

This section outlines the step-by-step numerical procedure. How the functions are defined and the order in which they are used to generate the final result. We only mentioned the mathematical functions here; the visualization parts should be explained in the future sections related to dashboard development.

1. **Create Unit Square:** The `create_square` function generates the coordinates of the vertices of a square given a size. The default size is set to 1. For example, a square of size 1 has corners at coordinates:

$$(-0.5, -0.5), (-0.5, 0.5), (0.5, 0.5), (0.5, -0.5)$$

These points represent the vertices of the square, centered at the origin. The first step is to define the unit square.

Algorithm 1 Create Unit Square

Input: size (default = 1)
Output: list of coordinates of square vertices
Set the four corners of the square as $(-0.5, -0.5), (-0.5, 0.5), (0.5, 0.5), (0.5, -0.5)$
if size $\neq 1$ **then**
    Scale the coordinates by the given size.
end if

return List of transformed coordinates

2. **Input value and round it to a perfect square:**

The `round_to_perfect_square` function rounds n to the nearest perfect square greater than or equal to the input number. For example, given $n = 10$, the function returns 16, since $4^2 = 16$ is the smallest perfect square greater than or equal to 10.

The aim is to distribute this square number within the Cartesian grid system, which is the main system for generating points, and then we can apply transformations to it.

The second step is to round the input (given by the user) to the nearest perfect square larger than the input number.

Algorithm 2 Round to Perfect Square

Input: n
Output: nearest perfect square
Find the square root of n
Round the square root to the nearest integer k

return k^2

3. **Grid Generation:**

   The `generate_grid_points` function generates a grid of points within the square based on the range of x and$y$ coordinates. The size of the grid is determined by n, where n is the number of points along each axis.

   An initial grid is generated for the unit square based on N. The total number of points is n^2, and the points are evenly spaced across the square.

Algorithm 3 Generate Grid Points

Input: n (number of points along each axis)
Output: list of grid points within the square
Generate the step size: step = $\frac{1}{n-1}$
Initialize an empty list for grid points
for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $n - 1$ **do**
    $x \leftarrow -0.5 + i \cdot \text{step}$
    $y \leftarrow -0.5 + j \cdot \text{step}$
    Append point (x, y) to the grid points list
  end for
end for

return list of grid points

   Transformations: Each grid is subject to a set of transformations, defined by a combination of scaling factors, rotation degrees, and translation values. A part of the code includes a hard-coded set of precomputed transformations, each represented as a tuple (θ, s, t_x, t_y), where θ is the rotation angle (in degrees), s is the scaling factor, and (t_x, t_y) are the translation components. These transformations are generated by a separate program that lists all possible valid combinations while ensuring that each transformation fully covers the unit square, as defined in the methodological approach chapter (see Figure 4.2). The complete list of these transformations is provided in Table 7.1 in the Appendix.

   The reason we needed this table was to accelerate the code's response time.

4. **Rotation Transformation** The rotation algorithm rotates a square around the origin at a specified angle. This transformation preserves the square's shape and size while changing its orientation. The implementation uses a standard 2D rotation matrix that operates on each vertex individually.

Algorithm 4 Rotate Square

Input: Square vertices $V = \{(x_1, y_1), ..., (x_4, y_4)\}$, angle θ (degrees)
Output: Rotated square vertices V'
Convert angle to radians:

$$\theta_r \leftarrow \text{radians}(\theta)$$

Compute rotation matrix:

$$R \leftarrow \begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) \\ \sin(\theta_r) & \cos(\theta_r) \end{bmatrix}$$

for each vertex $(x, y) \in V$ **do**
    Apply rotation:

$$(x', y') \leftarrow (x, y) \times R^\top$$

    $V' \leftarrow V' \cup \{(x', y')\}$
end for

return V'

5. **Scaling Transformation** The scaling algorithm uniformly resizes a square about the origin by a specified factor. This transformation changes the square's size while maintaining its shape and orientation. Scaling factors greater than 1 enlarge the square, while factors between 0 and 1 reduce it proportionally.

Algorithm 5 Scale Square

Input: Square vertices $V = \{(x_1, y_1), ..., (x_4, y_4)\}$, scale factor s
Output: Scaled square vertices V'
for each vertex $(x, y) \in V$ **do**
    Apply scaling:

$$(x', y') \leftarrow (s \cdot x, s \cdot y)$$

    $V' \leftarrow V' \cup \{(x', y')\}$
end for

return V'

6. **Translation Transformation** The translation algorithm displaces a square by a specified vector. This transformation changes the square's position without affecting its size, shape, or orientation. All vertices are shifted equally by the translation components in both x and y directions.

Algorithm 6 Translate Square

Input: Square vertices $V = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$, translation vector (t_x, t_y)
Output: Translated square vertices V'
Initialize $V' \leftarrow \emptyset$ % Initialize the output set
for each vertex $(x, y) \in V$ **do**
    Apply translation:
        $(x', y') \leftarrow (x + t_x, y + t_y)$
    $V' \leftarrow V' \cup \{(x', y')\}$
end for

return V'

7. **Evaluation:** The `points_inside_square` function checks how many points from a given set of points lie inside the unit square. The unit square is defined by the range $[-0.5, 0.5]$ on both axes. Therefore, any point (x, y) that satisfies the conditions:

$$-0.5 \leq x \leq 0.5 \quad \text{and} \quad -0.5 \leq y \leq 0.5$$

   is considered within the square.

   For each transformation, the number of points that remain inside the boundaries of the unit square after the transformation is calculated.

Algorithm 7 Checkpoints after Transformation

Input: scale_factor, rotation_degree, x_translation, y_translation, n (number of grid points)
Output: count of points inside the unit square after transformation
Generate grid points using n
Apply scaling, rotation, and translation transformations
Count how many points lie inside the unit square

return count

8. **Visualize the final result:** In the last step, the final transformation will print in the dashboard, and a scatter plot will show the distribution of points inside the unit square.

Algorithm 8 Visualize The Final Result

```
Input: perfect_square_list (grid sizes), transformations (scaling, rotation, translation)
Output: plot of grid sizes and points inside unit square
for each grid size in perfect_square_list do
    for each transformation in transformations do
        Apply the transformation to the grid points
        Count the number of points inside the unit square
        if desired condition met then
            Stop and plot results
        end if
    end for
end for

return plot of results
```

In the next chapter, we will discuss dashboard development and how it helps visualize the algorithms.

5 Dashboard Development

This chapter outlines the dashboard's development process, emphasizing its purpose, requirements, and prototype. Designed as a visualization tool, the dashboard aims to enhance understanding and exploration of the problem and the algorithm. It illustrates how transformations are applied to the points and promotes an even distribution of points within the sampling space.

The results can be explored using the interactive dashboard available at https://app-app-geabb2tv7ruxudw8pgtvmk.streamlit.app/.

5.1 Purpose of the Dashboard

The purpose of the dashboard is to facilitate a better understanding of the theoretical algorithms and methodology. Users can observe transformations affecting the grid's geometry in real time. This interactive visualization shows the distribution of points N within a unit square (our sampling space), which is a static square centered at the origin $(0, 0)$ with side length 1. The dashboard allows users to specify how many points they want to distribute within the unit square. Once entered, it immediately shows:

- The final arrangement of points in the unit square
- The transformations (scaling, rotation, translation) applied to achieve it.

Through these visualizations, users can observe how even a slight adjustment to the transformations can significantly alter their final count within the space.

5.2 Requirements

This section defines the essential features, functionalities, and technical specifications of the dashboard. The primary objectives are to enable user interaction and provide clear visual insight into different transformations. The dashboard must support real-time adjustments, accurately represent points distributed in the unit square, and maintain computational efficiency to handle various grid sizes and transformations.

5.2.1 Platform: Streamlit

The dashboard is built with Streamlit, an open-source Python framework for building interactive web applications. Streamlit is particularly useful for data visualization applications, as it allows the development and deployment of applications directly from Python scripts. It also provides real-time responsiveness to user input, while Python handles the computational backend, which is the feature required for this project.

5.2.2 Data Sources

The dashboard relies on two primary data sources to generate and display the results. The first is used to visualize the N points in the unit square, while the second reads data from a precalculated table to show other transformation options in the 4D scatterplot matrix. These sources serve different purposes and work together to provide users with both real-time insights and precomputed information.

Real-Time Calculations

The first data source is based on real-time calculations triggered whenever the user enters a value in the dashboard's input fields. Upon input, the system dynamically calculates the transformations and displays the resulting visuals. The back-end script processes the input data, applies the required transformations using the algorithm defined in the previous chapter, and dynamically updates both the visualizations and the results section. This allows users to interactively explore how changes in input values influence the distribution of points within the unit square.

Precalculated Table

The second data source is a precomputed table generated in advance and used to provide additional transformation options that produce the same number of points within the unit square as selected by the user.

Before presenting the precomputed table, it is important to clarify how multiple solutions for the same N arise. The primary algorithm is designed to stop as soon as it finds the first valid transformation for a given input N. This is done for efficiency, as a single working transformation is sufficient to demonstrate that N points can be placed inside the unit square. However, if the search continues beyond the first solution — by exploring different initial grid points and/or different transformations — additional results can be found. For the precomputed table, I intentionally continued the search and collected all such distinct transformations for each N.

You can see in the following table 5.1 an example for N = 14. It is created for all numbers between 1 and 1400.

This approach minimizes processing time and enables a responsive experience, as the dashboard can access transformation data without recomputing values for each interaction.

This table includes calculations for N values ranging from 1 to 1000. For each value N, the table lists how many unique transformations can successfully distribute that number of points inside the unit square. In the dashboard, this information is presented in a 4×4 scatter plot.

The scatter plot also provides access to alternative transformations that achieve the same result, enabling users to explore configurations that yield the desired distribution. This feature ensures comprehensive analysis and flexibility in understanding the possible arrangements for distributing a given number of inside points.

5.3 Prototypes

In this section, we will present the dashboard design. It is clean and easy to use, with charts, data tables, and key metrics. The layout is simple, so users can quickly find what they need. We will explain how it works and why we designed it this way.

Table 5.1: Precomputed transformation parameters for 14 points

Inside Points	Scaling Factor	Rotation Degree	Translation X	Translation Y
14	1.32	5	-0.01	0.11
14	1.30	7	-0.09	0.07
14	1.33	10	0.08	-0.07
14	1.32	10	-0.07	0.05
14	1.34	13	-0.07	0.045
14	1.32	15	0.05	-0.03
14	1.33	15	-0.04	0.02
14	1.30	17	0.03	-0.014
14	1.33	17	0.05	-0.025
14	1.33	17	-0.04	0.02
14	1.58	37	0.12	-0.02
14	1.55	37	-0.09	0.01
14	1.59	40	0.12	-0.02
14	1.47	40	-0.02	0
14	1.58	42	0.12	-0.01
14	1.46	42	-0.02	0

5.3.1 Interaction and Outputs

The dashboard offers multiple functionalities. Figure 5.1 provides an overview of its design and structure. This section explains the rationale behind the organization of its elements and describes the associated workflow.

- **User Input:** The user specifies the desired number of points N to be contained within the unit square. It can be typed or done using the increase/decrease buttons next to the input box, or the number can be inserted by typing it.
- **Output:** The applied transformations to achieve N points within the unit square, the dashboard presents the results below the input box.
  - The **scale factor**,
  - The **rotation angle** in degrees,
  - The **translation values** along the x- and y-axes.
  - The **Cartesian grid size** The number of grids in the Cartesian grid system (blue points in the Figure 5.2.
- **Plots:** The system generates the distribution of points within the unit square, visualized in two plots, which will be explained in the next section. Additionally, it displays a 4D scatterplot matrix based on a precalculated table for the input value, revealing alternative transformation possibilities for the given input.

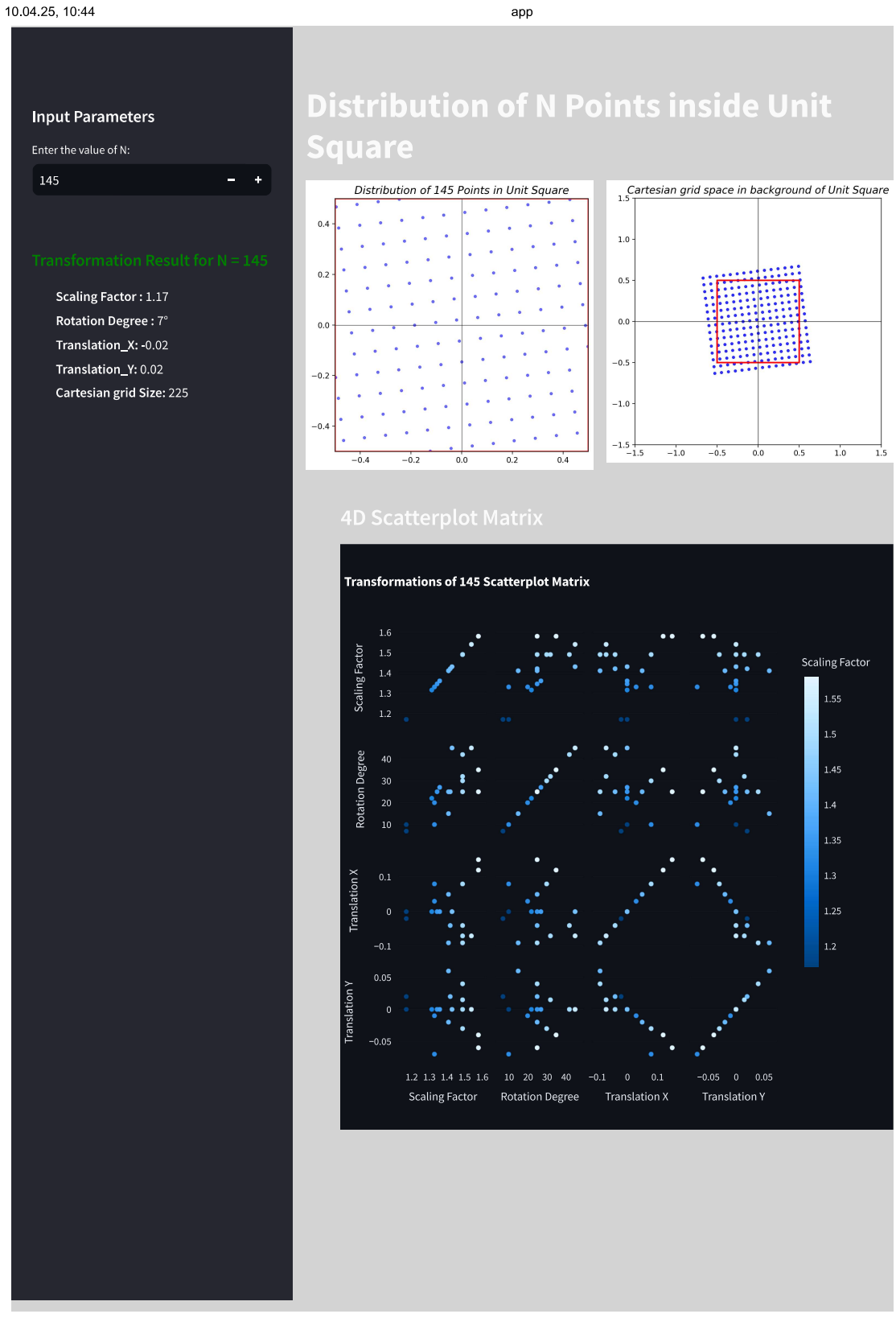


Figure 5.1: Dashboard overview: (1) user input section, (2) print computed results, (3) visualization of point distribution, and (4) a 4D scatterplot of alternative transformations.

5.3.2 Plots

The first part of the visualization contains two main components:

Red unit square: A fixed red square centered on $(0, 0)$. This represents the unit square where the program evaluates how many points in the transformed grid fall inside.

Transformed points (blue dots): Transformed square grid points are plotted as blue dots. These represent the new positions of the grid points after scaling, rotation, and translation transformations are applied.

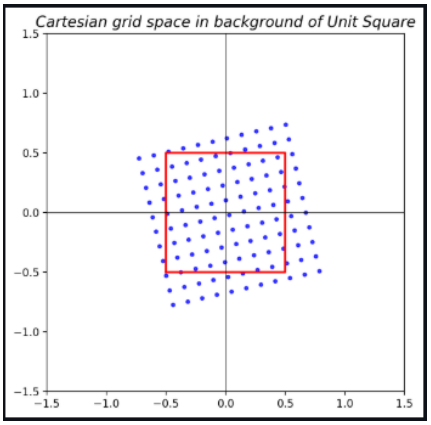


Figure 5.2: The red unit square represents the sampling space, while the blue points are transformed to achieve the desired distribution within the unit square.

Key Insights:

- The visualization shows how transformations affect the spatial distribution of points within the unit square.
- Points that fall within the unit square are plotted, whereas those outside the boundaries are excluded from the display.

The other plot (4D scatter plot matrix): The other plot provides a broader view, showing the Cartesian coordinate grid around the unit square. This visualization includes transformations such as rotation, scaling, and translation, with grid points distributed accordingly. The unit square, along with the desired N points inside it, remains visible within this transformed space. Additional visualizations are available, including a 4D scatter plot that illustrates other transformations that yield the same number of points within the unit square.

This dashboard serves as a tool for the mathematical and visual exploration of geometric transformations. Even without visualization, the numerical output (scale factor, rotation, translation) provides precise information about the transformations required to achieve the desired point distribution.

Example Result

For example, if the user inputs $N = 145$, the system might produce the following result:

- Scale Factor: 1.22,

- Rotation Angle: 10°,
- Translation: $(x = -0.02, y = 0.1)$.

This indicates that, after scaling the grid by 1.17, rotating it by 7°, and translating it by -0.02 units along the x axis and 0.02 units along the x axis, exactly the 145 grid points fall inside the unit square.

Points Achievable with Scaling and Rotation

We tried to find the distribution of points by rotating the Cartesian grid, but there were some numbers of points for which no transformation exists to distribute them inside the unit square. Then we added scaling, and the results were promising. In the next step, we sought a rule for each combination of scaling factor and rotation degree. To do that, the table is generated. It contains all the numbers of distributed points in the unit square obtained by combining these 2 transformations. This lists the values of N (the number of points) that can be achieved using only scaling and rotation, together with their corresponding transformation parameters. See Table 7.2 in Appendix 7 for more details.

Figure 5.3 illustrates the relationship between geometric transformations (rotation and scaling) and the resulting point distributions within the unit square. The plot shows the following.

- **Y-axis**: Pairs of rotation-scaling parameters, formatted as:

  Rotation Angle (°)Scaling Factor (SF: $x.xx$)

  where each entry represents a unique combination tested in our methodology.
- **X-axis**: Generated point counts (N) that achieve equidistant distribution when the specified transformations are applied to the base Cartesian grid.

Key observations from the visualization include the following.

1. **Perfect Square Pattern (0° rotation)**: The baseline configuration (0° rotation, SF:1.00) exclusively generates perfect squares ($N = 1, 4, 9, \ldots, 961$), confirming the theoretical expectation for an untransformed grid.
2. **Rotation-Induced Complexity**: As rotation increases (5°-45°), the scaling factor grows non-linearly (1.10–1.43), and the producible point counts become more irregular. Higher rotations require greater scaling to maintain unit square coverage, as evidenced by the following.
   - The 45° configuration (SF:1.43) generates only seven viable N values
   - Intermediate rotations (15°–35°) yield the most diverse point distributions

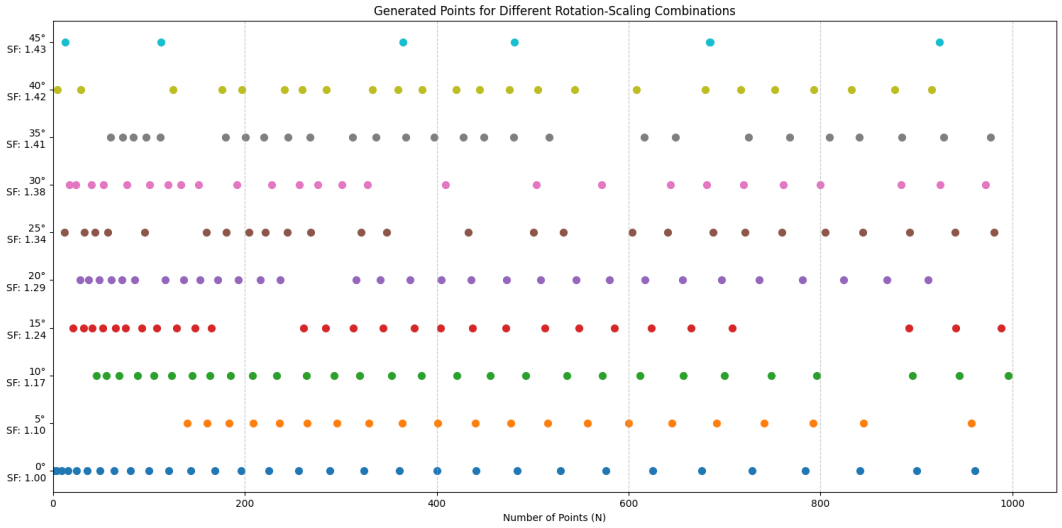


Figure 5.3: Visualization of point distribution patterns showing the relationship between geometric transformations and generated points. The y-axis displays rotation-scaling parameter pairs, while the x-axis shows the number of points (N) that achieve equidistant distribution under each transformation.

Figure 5.4 reveals some patterns. For each rotation angle, the achievable point counts N increase with the sequence index, but some integer values were not achieved. As the rotation angle increases from 0° to 45°, the points become sparser on the plot, showing that rotation reduces the number of achievable values of N. At 45°, only seven isolated points appear, showing how extreme rotation severely restricts the possible values. Despite these differences, all rotation angles eventually reach a similar maximum N (around 1000) when the scaling factor becomes large enough. In short, rotation introduces gaps and sparsity in the achievable point counts, with 45° being the most restrictive case.

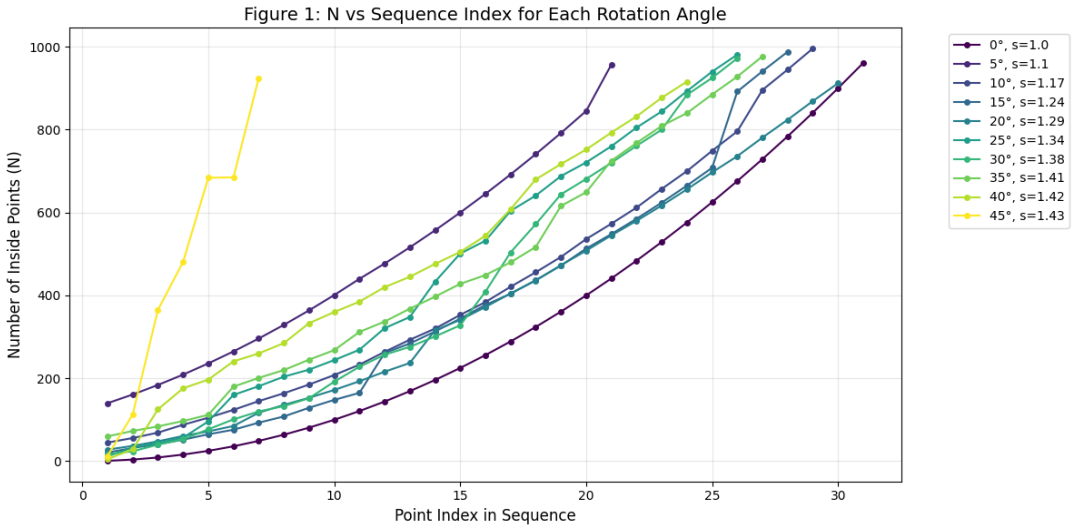


Figure 5.4: Achievable point counts N vs. sequence index per rotation angle. Higher rotations introduce increasing gaps and sparsity, with 45° being the most restrictive case (only seven points).

5.3.3 Rules

Rule for Transformation with Rotation 5° and Scaling 1.1

For the transformation with rotation $\theta = 5°$ and scaling factor $s = 1.10$, the sequence of achievable point counts N (sorted in increasing order) follows a quadratic function of the index i:

$$\boxed{N_i = p \cdot i^2 + q \cdot i + r}$$

where:

- N_i is the i-th value in the sequence ($i = 1, 2, 3, \ldots$)
- p, q, and r are constants determined by fitting the data

From the actual data in Table 7.2 for $\theta = 5°$, $s = 1.10$:

$$N = [140, 161, 184, 209, 236, 265, 296, 329, 364, 401,$$
$$440, 477, 516, 557, 600, 645, 692, 741, 792, 845, 957]$$

Fitting a quadratic $N_i = pi^2 + qi + r$ to these values yields:

$$p \approx 1.17, \quad q \approx 18.2, \quad r \approx 120.5$$

Note: The coefficient $p \approx 1.17$ is a numerical result of the fit. It is **not** the scaling factor (which is $s = 1.10$). The scaling factor influences p, but they are different quantities. For other rotation angles, the quadratic coefficient p will differ.

Verification: Figure 5.5 shows the actual sequence values (blue circles) alongside the fitted quadratic model (red dashed line). The residuals (bottom panel) are small and randomly distributed, confirming that a quadratic model accurately describes the sequence.

Rule for Transformation with Rotation 15° and Scaling 1.24

For the transformation with rotation $\theta = 15°$ and scaling factor $s = 1.24$, the sequence of achievable point counts N (sorted in increasing order) follows a quadratic function of the index i:

$$\boxed{N_i = p \cdot i^2 + q \cdot i + r}$$

where:

- N_i is the i-th value in the sequence ($i = 1, 2, 3, \ldots$)
- p, q, and r are constants determined by fitting the data

From the actual data in Table 7.2 for $\theta = 15°$, $s = 1.24$:

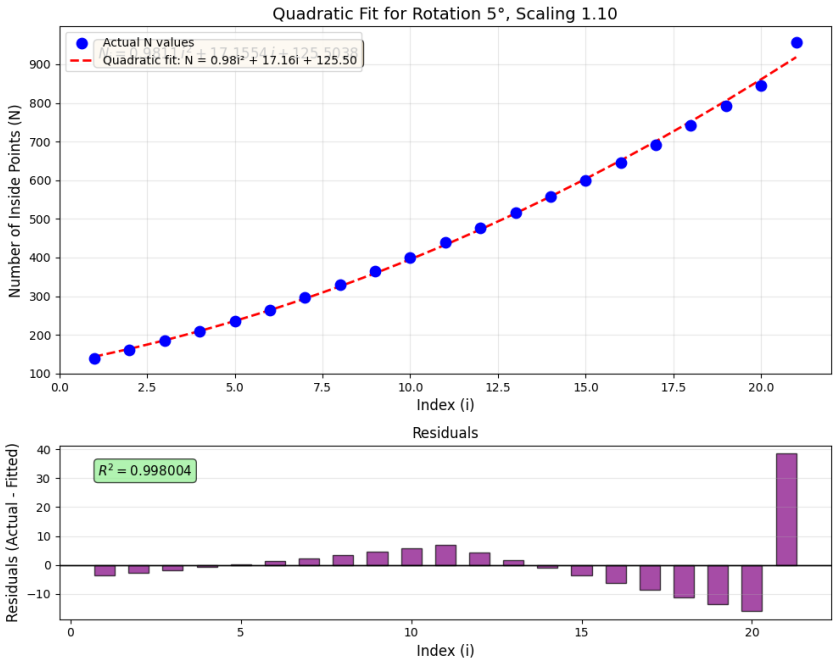


Figure 5.5: Quadratic fit to the N sequence $\theta = 5°$ and $s = 1.10$

$$N = [21, 32, 41, 52, 65, 76, 93, 108, 129, 148, 165, 261, 284, 313, 344, 377,$$
$$404, 437, 472, 513, 548, 585, 624, 665, 708, 892, 941, 988]$$

Fitting a quadratic $N_i = pi^2 + qi + r$ to these values yields:

$$p \approx 1.23, \quad q \approx 4.88, \quad r \approx 15.00$$

Note: The quadratic coefficient $p \approx 1.23$ is very close to the scaling factor $s = 1.24$. This suggests that, for this rotation angle, the quadratic coefficient is approximately equal to the scaling factor. The linear coefficient $q \approx 4.88$ and constant term $r \approx 15.00$ adjust for the specific geometry of the 15° rotation.

Proposed Simplified Formula: Given the proximity of p to $s = 1.24$, we propose the following simplified rule for this transformation:

$$\boxed{N(i) \approx 1.24 \cdot i^2 + 5 \cdot i + 15}$$

where the quadratic term is scaled directly by the transformation's scaling factor, the linear term is approximately 5, and the constant is 15. This rule captures the accelerated growth of the sequence, with larger increments observed at higher indices.

Verification: Figure 5.6 compares the actual sequence values with both the best-fit quadratic and the proposed simplified formula. The key observations are:

- The best-fit quadratic (red line) matches the data almost perfectly.
- The proposed formula $N(i) = 1.24i^2 + 5i + 15$ (green dashed line) is visually very close to the best fit.

- Residuals for both fits are small (bottom panels), with RMS errors of approximately 2.3 and 3.1 respectively.
- Both fits achieve $R^2 > 0.999$, confirming that a quadratic model accurately describes the sequence.

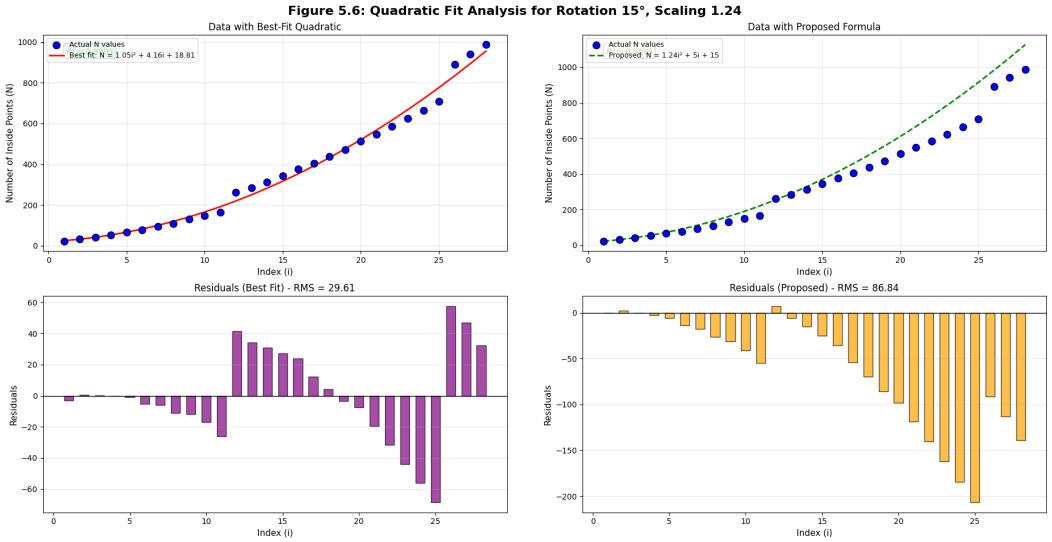


Figure 5.6: Quadratic fit analysis for rotation 15° and scaling factor 1.24. Top-left: Actual data (blue circles) with best-fit quadratic (red line). Top-right: Actual data with proposed formula $N(i) = 1.24i^2 + 5i + 15$ (green dashed line). Bottom panels show residuals. The close agreement validates the quadratic rule.

Connection to the Scaling Factor: The fact that the quadratic coefficient $p \approx 1.23$ is nearly equal to the scaling factor $s = 1.24$ reveals an important pattern:

$$p \approx s \quad \text{for} \quad \theta = 15°$$

This suggests that the scaling factor directly influences the quadratic growth rate of the sequence. For other rotation angles, the relationship between p and s may differ, but the quadratic form remains valid. This empirical observation provides a practical rule for predicting N values without performing geometric transformations repeatedly.

5.4 Summary

This chapter has outlined the requirements and prototype development, and presented rules for the number of points with specified rotation degrees and scaling factors. In the next chapter, we will see that the pseudocode is used to implement this dashboard.

6 Discussion

In this chapter, we discuss the contribution, limitations, and future work.

6.1 Contribution

The core contributions of this thesis are (i) finding the solution for distributing any desired points in a unit square while reducing step sizes for transformation, ensuring a reasonable running time; (ii) developing a user-friendly dashboard to visualize the solution in real time; and (iii) identifying possible patterns in the solution to expand the method to higher dimensions.

6.2 Summary of Key Findings

This thesis investigated the problem of equidistant point distribution in a 2D unit square for arbitrary N points, a critical challenge for computational sampling, visualization, and spatial analysis. Our primary contributions include

- A **methodology of geometric transformation** (rotation, scaling, translation) to adapt Cartesian grids for arbitrary N, addressing gaps in static grid and Latin hypercube approaches.
- An **interactive dashboard** enabling real-time exploration of point distributions, bridging theoretical and practical applications.
- Systematic identification of edge cases (e.g., nonsquare N) where traditional methods fail, with proposed adaptive solutions.

6.3 Limitations

Our approach encounters several key limitations when generating point distributions.

- **Step size sensitivity**: For certain values of N (particularly small prime numbers), the default step sizes fail to produce uniform distributions
- **Manual adjustments required**: Cases like $N = 3$ demand iterative step size reduction and positional adjustment (Figure 6.1).

Example Case ($N = 3$):
These observations reveal fundamental constraints in our method.

- Dependence on N's factorization properties
- Trade-offs between computation time and precision

- Geometric constraints of the unit square boundary

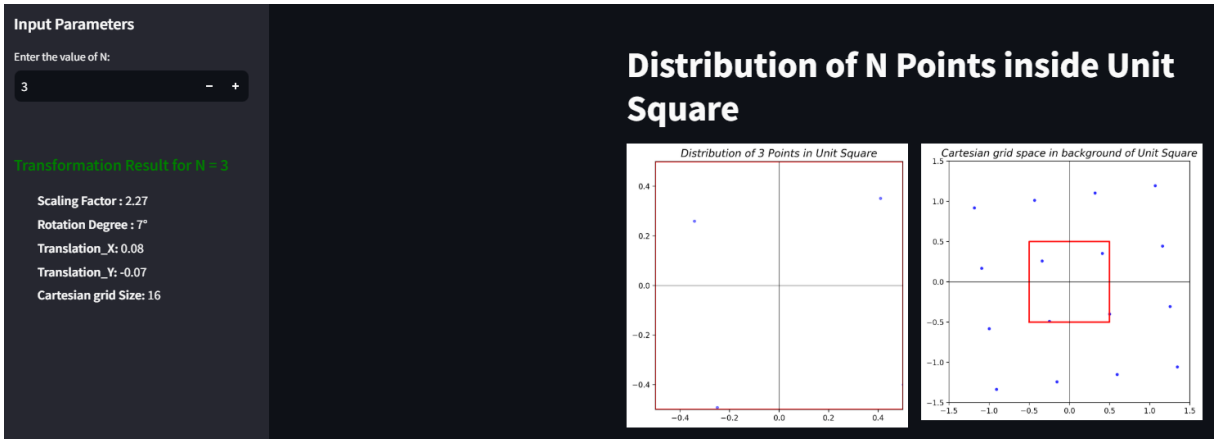


Figure 6.1: Visualization of the distribution for $N = 3$, showing the characteristic empty regions that emerge despite optimization attempts.

6.4 Future Work

To address these limitations, we propose the following directions:

- **Automated Parameter Tuning**: Integrate metaheuristics (e.g., genetic algorithms) to optimize transformation sequences without manual intervention.
- **3D Generalization**: Extend the method to volumetric sampling by incorporating z-axis transformations (e.g., spherical coordinate mappings).
- **Benchmarking**: Rigorous comparison against quasi-Monte Carlo and low-discrepancy sequences [14] to quantify the efficacy of gap reduction.
- **Adaptive Step Sizes**: Dynamically adjust rotation/translation increments based on N's factorization properties to reduce search space.

For future work, we plan to extend the distribution of equidistant points to higher dimensions, beginning with three dimensions and beyond. Furthermore, instead of limiting our study to the unit square, we will explore the distribution of equidistant points in various geometric shapes and spaces across different dimensions. Additionally, we will enhance the tool by allowing selection of step sizes for each transformation, providing greater flexibility within the system.

We also would like to investigate possible rules for the distribution of certain groups of numbers based on specific combinations of transformations on a broader scale. Understanding these patterns will deepen understanding of equidistance and even distribution across different spaces. Finally, these steps will not only enhance the theoretical understanding of the problem but also improve the algorithm's visualization capabilities, making it more usable across a wider range of practical scenarios.

6.5 Conclusion

By unifying geometric transformations with interactive tools, this work advances the state of the art in equidistant sampling. Although limitations persist in computational complexity and edge

cases, the framework's adaptability offers a foundation for future research in spatial computing and beyond. The interactive component, in particular, opens new avenues for democratizing advanced sampling techniques.

Bibliography

[1] Sebastian Burhenne, Dirk Jacob, and Gregor P Henze. Sampling based on sobol'sequences for monte carlo techniques applied to building simulations. In *Building Simulation 2011*, volume 12, pages 1816–1823. IBPSA, 2011.

[2] H. Chi, M. Mascagni, and T. Warnock. On the optimal halton sequence. *Mathematics and Computers in Simulation*, 70(1):9–21, 2005. ISSN 0378-4754. doi: https://doi.org/10.1016/j.matcom.2005.03.004. URL https://www.sciencedirect.com/science/article/pii/S037847540500087X.

[3] Nishant Dige and Urmila Diwekar. Efficient sampling algorithm for large-scale optimization under uncertainty problems. *Computers & Chemical Engineering*, 115:431–454, 2018. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2018.05.007. URL https://www.sciencedirect.com/science/article/pii/S009813541830437X.

[4] Qiang Du, Maria Emelianenko, Hyung-C. Lee, and Xiaoqiang Wang. Ideal point distributions, best mode selections and optimal spatial partitions via centroidal voronoi tessellations. In *Proceedings (unknown)*, 2005. URL https://api.semanticscholar.org/CorpusID:14057480.

[5] Henri Faure and Christiane Lemieux. Generalized halton sequences in 2008: A comparative study. *ACM Trans. Model. Comput. Simul.*, 19:15:1–15:31, 2009. URL https://api.semanticscholar.org/CorpusID:14128369.

[6] Jiří Filip and Radomír Vávra. Template-based sampling of anisotropic brdfs. In *Computer Graphics Forum*, volume 33, pages 91–99. Wiley Online Library, 2014.

[7] F Javier Gallego. *Sampling frames of square segments*. European Commission Luxembourg, 1995.

[8] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical algorithms*, 18(3):209–232, 1998.

[9] Holger Heitsch. Annual research report 2021. In *Proceedings (unknown)*, 2022. URL https://api.semanticscholar.org/CorpusID:259768421.

[10] Ronald L Iman. L atin hypercube sampling. *Wiley StatsRef: Statistics Reference Online*, 2014.

[11] Budiman Minasny and Alex B. McBratney. A conditioned latin hypercube method for sampling in the presence of ancillary information. *Computers & Geosciences*, 32(9):1378–1388, 2006. ISSN 0098-3004. doi: https://doi.org/10.1016/j.cageo.2005.12.009. URL https://www.sciencedirect.com/science/article/pii/S009830040500292X.

[12] Budiman Minasny and Alex B. McBratney. A conditioned latin hypercube method for sampling in the presence of ancillary information. *Comput. Geosci.*, 32:1378–1388, 2006. URL https://api.semanticscholar.org/CorpusID:15508922.

[13] Art B. Owen. Scrambling sobol' and niederreiter–xing points. *Journal of Complexity*, 14 (4):466–489, 1998. ISSN 0885-064X. doi: https://doi.org/10.1006/jcom.1998.0487. URL https://www.sciencedirect.com/science/article/pii/S0885064X98904873.

[14] Tofik Mussa Reshid. Sampling distribution and simulation in r. *International Journal*, 7 (2):154–163, 2020.

[15] Nathalie Saint-Geours, Jean-Stéphane Bailly, Christian Lavergne, and Frédéric Grelot. Latin Hypercube Sampling of Gaussian random field for Sobol' global sensitivity analysis of models with spatial inputs and scalar output. In *Proceedings of the ninth international symposium on spatial accuracy assessment in Natural Ressources and Environmental Sciences*, Proceedings of the ninth international symposium on spatial accuracy assessment in Natural Ressources and Environmental Sciences, pages 81–84, Leicester, United Kingdom, July 2010. Nicholas J Tate, Peter F Fisher. URL https://hal.science/hal-00470529. [Departement_IRSTEA]Territoires [TR1_IRSTEA]SYNERGIE
4 pages.

[16] Justin Tzou and Brian Wetton. Optimal covering points and curves. *AIMS Mathematics*, 4(6):1796–1804, 2019. ISSN 2473-6988. doi: 10.3934/math.2019.6.1796. URL https://www.aimspress.com/article/doi/10.3934/math.2019.6.1796.

7 Appendix

7.1 Transformations Table

Table 7.1: Transformations: Scaling Factor, Rotation Degree, and Translation factors

Scaling Factor	**Rotation Degree** (°)	**Translation** t_x	**Translation** t_y
1.1	5	0	0
1.14	5	0.03	-0.02
1.19	5	0.05	-0.04
1.25	5	0.08	-0.07
1.32	5	0.12	-0.1
1.32	5	-0.1	0.09
1.25	5	-0.07	0.06
1.19	5	-0.04	0.03
1.14	5	-0.02	0.1
1.18	10	0	0
1.22	10	0.03	-0.02
1.27	10	0.05	-0.04
1.33	10	0.08	-0.07
1.39	10	0.12	-0.09
1.36	10	-0.09	0.06
1.32	10	-0.07	0.05
1.27	10	-0.04	0.03
1.22	10	-0.02	0.1
1.25	15	0	0
1.28	15	0.03	-0.02
1.32	15	0.05	-0.03
1.37	15	0.08	-0.05
1.43	15	0.12	-0.07
1.41	15	-0.09	0.06
1.38	15	-0.07	0.05
1.33	15	-0.04	0.02
1.29	15	-0.02	0.1
1.31	20	0	0
1.33	20	0.03	-0.01
1.36	20	0.05	-0.02
1.42	20	0.08	-0.04
1.46	20	0.12	-0.05
1.45	20	-0.09	0.04
1.42	20	-0.07	0.03
1.38	20	-0.04	0.02

Continued on next page

Table 7.1 – Continued from previous page

Scaling Factor	Rotation Degree (°)	Translation t_x	Translation t_y
1.34	20	-0.02	0.1
1.345	25	0	0
1.39	25	0.03	-0.01
1.41	25	0.05	-0.02
1.46	25	0.08	-0.04
1.52	25	0.12	-0.05
1.49	25	-0.09	0.04
1.47	25	-0.07	0.03
1.42	25	-0.04	0.02
1.39	25	-0.02	0.1
1.4	30	0	0
1.42	30	0.03	-0.01
1.44	30	0.05	-0.02
1.49	30	0.08	-0.04
1.55	30	0.12	-0.04
1.54	30	-0.09	0.04
1.51	30	-0.07	0.03
1.47	30	-0.04	0.02
1.44	30	-0.02	0.1
1.42	35	0	0
1.45	35	0.03	-0.01
1.48	35	0.05	-0.02
1.53	35	0.08	-0.02
1.58	35	0.12	-0.04
1.54	35	-0.09	0.02
1.52	35	-0.07	0.02
1.48	35	-0.04	0.01
1.44	35	-0.02	0.01
1.43	40	0	0
1.46	40	0.03	-0.01
1.5	40	0.05	-0.02
1.54	40	0.08	-0.02
1.58	40	0.12	-0.02
1.56	40	-0.09	0.01
1.53	40	-0.07	0
1.49	40	-0.04	0.01
1.47	40	-0.02	0
1.43	45	0	0
1.47	45	0.03	-0.01
1.5	45	0.05	-0.01
1.55	45	0.08	-0.01
1.59	45	0.12	-0.01
1.57	45	-0.09	0
1.54	45	-0.07	0
1.5	45	-0.04	0

Continued on next page

Table 7.1 – Continued from previous page

Scaling Factor	Rotation Degree (°)	Translation t_x	Translation t_y
1.47	45	-0.02	0.1

7.2 Achievable points applying just scaling and rotating

Table 7.2: Achievable points applying just scaling and rotation

Rotation Degree (°)	Scaling Factor	*N* (Points)
0	1.00	1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961
5	1.10	140, 161, 184, 209, 236, 265, 296, 329, 364, 401, 440, 477, 516, 557, 600, 645, 692, 741, 792, 845, 957
10	1.17	45, 56, 69, 88, 105, 124, 145, 164, 185, 208, 233, 264, 293, 320, 353, 384, 421, 456, 493, 536, 573, 612, 657, 700, 749, 796, 896, 945, 996
15	1.24	21, 32, 41, 52, 65, 76, 93, 108, 129, 148, 165, 261, 284, 313, 344, 377, 404, 437, 472, 513, 548, 585, 624, 665, 708, 892, 941, 988
20	1.29	28, 37, 48, 61, 72, 85, 117, 136, 153, 172, 193, 216, 237, 316, 341, 372, 405, 436, 473, 508, 545, 580, 617, 656, 697, 736, 781, 824, 869, 912
25	1.34	12, 33, 44, 57, 96, 160, 181, 204, 221, 244, 269, 321, 348, 433, 501, 532, 604, 641, 688, 721, 760, 805, 844, 893, 940, 981
30	1.38	17, 24, 40, 53, 77, 101, 120, 133, 152, 192, 228, 257, 276, 301, 328, 409, 504, 572, 644, 681, 720, 761, 800, 884, 925, 972
35	1.41	60, 73, 84, 97, 112, 180, 201, 220, 245, 268, 312, 337, 368, 397, 428, 449, 480, 517, 616, 649, 725, 768, 809, 840, 885, 928, 977
40	1.42	5, 29, 125, 176, 197, 241, 260, 285, 333, 360, 385, 420, 445, 476, 505, 544, 608, 680, 717, 752, 793, 832, 877, 916
45	1.43	13, 113, 365, 481, 684, 685, 924