A dissertation submitted towards the degree
Doctor of Engineering (Dr.-Ing.) of the Faculty
of Mathematics and Computer Science of
Saarland University

Understanding noise correlations in generative models for graphics content generation and editing

submitted by
Xingchang Huang
Saarbrücken, 2025

Day of Colloquium	24 November, 2025
Dean of the Faculty	Prof. Dr. Roland Speicher
Chair of the Committee	Prof. Dr. Thorsten Herfet
Reviewers	Dr. Gurprit Singh
	Prof. Dr. Hans-Peter Seidel
	Prof. Dr. Petr Kellnhofer
Academic Assistant	Dr. Easa AliAbbasi

Acknowledgments

I began my PhD during the challenging period of the Covid. Fortunately, I received invaluable support from many individuals, without whom this journey would not have been as rewarding. First and foremost, I would like to express my deepest gratitude to Gurprit Singh, who has guided me throughout the past five years. His thoughtful feedback on my research and presentations, his encouragement to engage in collaborations, and his patience through every obstacle I encountered have been truly instrumental. His expertise and vision have continually inspired me to broaden my perspective and think beyond conventional boundaries. I am also sincerely thankful to Hans-Peter Seidel for providing me with the opportunity and funding to pursue research at MPI Informatics. This support allowed me to fully immerse myself in developing the skills and experience necessary for academic research. I especially appreciate the interactive and supportive research environment he creates here, including the stimulating discussions during CG lunches and the constructive feedback on my projects during rehearsals and the qualifying exam.

I have been fortunate to work and learn alongside many brilliant collaborators throughout my PhD journey. I would like to thank Pooran Memari for guiding me in gaining deeper insights into geometry-related problems, for helping me develop better project management skills, and for offering me the opportunity to visit their lab and learn more about their research. Your support and encouragement have been a great source of motivation through both the highs and lows. To Tobias Ritschel, thank you for continuously pushing me to strive for higher standards and better results. Your insights, experience, and writing have been invaluable in shaping the direction and quality of our projects. I am also deeply grateful to Cengiz Öztireli and Cristina Vasconcelos for sharing their expertise in generative models, diffusion models, noise analysis, and image generation. Your guidance consistently pointed me in the right direction when navigating challenging research topics. Many thanks to Christian Theobalt for the opportunity to present at the new Saarbrücken VIA-Center during meetings and demo days, and for enabling collaborations with outstanding researchers from Google. I would also like to acknowledge Vahid Babaei, Paul Strohmeier, Easa AliAbbasi, Gabriela Vega, Thomas Leimkühler, Karol Myszkowski, Iliyan Georgiev, Niloy Mitra, Ashish Kumar Singh, Florian Dubost, Sakar Khattar, and Liang Shi for the insightful discussions and contributions from various research areas during our collaborations and group meetings. Finally, special thanks to Petr Kellnhofer for kindly serving on my thesis committee.

In addition, I would like to extend my thanks to everyone who has supported me with administrative matters. Special thanks to Gereon Fox, Sabine Budde, Ellen Fries, Sabine Zimmer, Anne Gesellchen, Petra Schaaf, Bin Chen, Ruth Schneppen-Christmann, Felix Mujkanovic, and the IST team for their invaluable help with visa arrangements, reimbursements, residence permit extensions, technical support, and many other administrative tasks that I often found overwhelming. I am especially grateful for your patience and understanding—no matter how unusual or unexpected my requests may have been.

I would like to sincerely thank my colleagues and friends—Corentin Salaün, Martin Bálint, Daniel Jimenez Navarro, Haolin Lu, Adarsh Djeacoumar, Bin Chen, Lingyan Ruan, Jente Vandersanden, Vishesh Gupta, Felix Mujkanovic, Sascha Holl, Emiliano Luci, Ugur Cogalan, Ntumba Elie Nsampi, Chao Wang, Xingang Pan, Wenxin Liu, Heming Zhu, Haoran Wang, Jianchun Chen, Guoxing Sun—and many others too numerous to name individually. Thank you for all the random yet inspiring conversations that somehow became part of my personal guide to surviving a PhD. I've learned so much from our everyday chats. From hanging out, having lunch and dinner, playing sports, and watching movies, to attending conferences and even traveling together, every moment with you made this journey more enjoyable and meaningful. I am also grateful to all the participants in my user studies, without whose involvement the results presented in my papers would not have been possible. A special thanks to Corentin Salaün, who has not only been a collaborator and colleague but also my office mate. Sharing the space where I spent most of my PhD time, you helped create a warm, productive, and comfortable environment that I will always remember fondly.

Last but certainly not least, my heartfelt thanks go to my family for their unconditional love and support. To my parents—thank you for always believing in me and encouraging me to follow my dreams. I couldn't have done this without you.

Abstract

Graphics content plays an essential role in our daily lives, spanning diverse media forms such as images, videos, textures, point patterns, and 3D objects. Despite their ubiquity, limited research has investigated how the concept of noise correlation across these varied representations can have an impact on controllable graphics generation and editing aligned with user intent.

This thesis explores how noise correlations can inform and simplify the process of generating and editing visual content. We focus in particular on the synthesis and editing of point patterns, images, and stereo videos. Our primary contributions lie in developing novel pipelines that leverage noise correlations to improve both fidelity and controllability in these domains. Specifically, we propose: (1) a training-free holistic feature extraction pipeline inspired by noise correlation for point pattern synthesis, (2) a decoupled representation and correlation embedding space for user-friendly point pattern editing, (3) a new perspective on integrating blue noise correlations into the training and sampling of diffusion models, and (4) a unified pipeline with noisy degraded data augmentation for fine-tuning diffusion models to enable simultaneous stereo video generation and restoration.

Our methods demonstrate the capability of outperforming existing state-of-the-art techniques and highlight the untapped potential of noise structure as a guiding principle in visual synthesis. These findings open new avenues for controllable and high-quality graphics content generation and editing.

Zusammenfassung

Grafische Inhalte spielen eine zentrale Rolle in unserem Alltag und erscheinen in vielfältigen Medienformen wie Bildern, Videos, Texturen, Punktmustern und 3D-Objekten. Trotz ihrer Allgegenwärtigkeit wurde bisher kaum erforscht, wie das Konzept der Rauschkorrelationen über diese unterschiedlichen Repräsentationen hinweg die kontrollierbare Generierung und Bearbeitung grafischer Inhalte im Einklang mit Benutzerabsichten beeinflussen kann.

Diese Dissertation untersucht, wie Rauschkorrelationen den Prozess der Generierung und Bearbeitung visueller Inhalte vereinfachen und verbessern können. Im Fokus stehen dabei insbesondere die Synthese und Bearbeitung von Punktmustern, Bildern und Stereovideos. Der zentrale Beitrag liegt in der Entwicklung neuartiger Pipelines, die Rauschkorrelationen nutzen, um sowohl die Qualität als auch die Steuerbarkeit in diesen Bereichen zu erhöhen. Im Einzelnen schlagen wir vor: (1) eine trainingsfreie Pipeline zur holistischen Merkmalsextraktion, inspiriert von Rauschkorrelationen, für die Synthese von Punktmustern, (2) eine entkoppelte Repräsentation und einen Korrelations-Embedding-Raum für benutzerfreundliche Punktmusterbearbeitung, (3) eine neue Perspektive zur Integration von Blue-Noise-Korrelationen in das Training und Sampling von Diffusionsmodellen sowie (4) eine einheitliche Pipeline mit augmentierten, verrauschten Daten zur Feinabstimmung von Diffusionsmodellen für die gleichzeitige Generierung und Restaurierung von Stereovideos.

Unsere Methoden zeigen eine Überlegenheit gegenüber aktuellen State-of-the-Art-Techniken und unterstreichen das bisher ungenutzte Potenzial von Rauschstrukturen als richtungsweisendes Prinzip in der visuellen Synthese. Diese Erkenntnisse eröffnen neue Perspektiven für die kontrollierbare und qualitativ hochwertige Generierung und Bearbeitung grafischer Inhalte.

Contents

Chapter 1
Introduction

1.1 Motivation

Noise correlations are ubiquitous in both the real world and computer graphics. We can observe them in many forms, from the spatial arrangement of trees in a forest, people in a crowd, to dots in stippling art. As illustrated in Figure 1.1, the first three examples can be seen as **point-based** forms of correlated noise.

In graphics, noise correlations also appear in **pixel-based** representations such as textures, which exhibit structured patterns similar to those in Figure 1.1. These textures can be used to drive point distribution, synthesize materials, or generate content with controllable structure. They serve as fundamental building blocks for representing, generating, and editing visual content across various modalities.

Despite their ubiquity, there is still no unified framework to understand how noise correlations influence traditional graphics generation and editing. Meanwhile, with the rise of modern generative models, especially diffusion-based ones, images and videos can now be created with minimal user effort. In such models, noise plays a central role, yet the correlations within noise have rarely been considered.

This gap motivates us to systematically study the impact of noise correlation on graphics content generation and editing, from both **point-based** and **pixel-based** perspectives.

1.2 Noise correlations

In this thesis, we define noise as a combination of density and correlation, realized either as points or pixels. The noise density describes how frequently samples occur across different regions. For example, Uniform and Gaussian are two common probability density functions: the Uniform distribution places samples with equal probability, whereas the Gaussian distribution concentrates them around the mean.

The correlation of the noise characterizes how samples relate to one another. For instance,

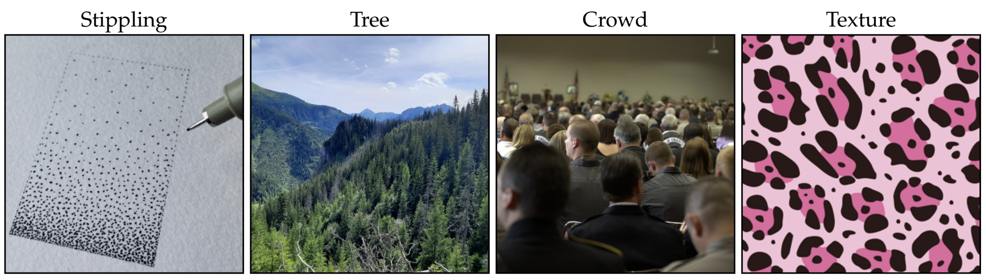


Figure 1.1: *Noise correlations as point patterns/distributions in the real world. Image sources from left to right are from Accurso [2021], Kluczek et al. [2023], Wang et al. [2020], free-vector [2024].*

whether nearby samples are independent, repulsive, or clustered. Even under the same density function, different correlations can lead to dramatically different sample arrangements.

Together, density and correlation define the underlying structure of randomness that drives many graphics generation processes. In this thesis, we focus on understanding how to characterize and synthesize noise correlations from both point-based and pixel-based perspectives. This aspect remains underexplored in generative modeling for graphics.

Noise correlation can be visualized as different colors of noise, such as blue, green, and pink noise, as shown in Fig. 1.2. These point patterns have the same number of points, but with different correlations, named as different colors of noise. The "color" terminology refers to the frequency characteristics of their Fourier power spectra. More specifically, the blue noise pattern does not have low-frequency components in its frequency power spectrum, while green and pink noise do not have mid- and high-frequency components. White noise spans all the frequencies and is often called random noise. Details about how to compute the Fourier power spectrum will be explained in Chapter 2.

1.3 Graphics content representation, generation and editing

1.3.1 Representations

In the realm of computer graphics, there exist diverse graphics representations depending on the applications. **Images**, the most common form, capture and convey visual information in a two-dimensional format, utilized everywhere from web content to print media. **Videos** extend this by adding the temporal dimension, creating moving visuals that are integral to entertainment, education, and storytelling, and can also serve critical purposes in fields such as medical imaging, where temporal dynamics convey diagnostic or functional information.

Point patterns, also called point-based textures, offer a more abstract representation than images, where sets of individual points are used to define textures or patterns, crucial in digital art and procedural texturing. 3D objects, constructed from complex meshes or point clouds, are pivotal in virtual reality, gaming, and simulation, providing immersive experiences that mimic the real world. Each of these representations employs different

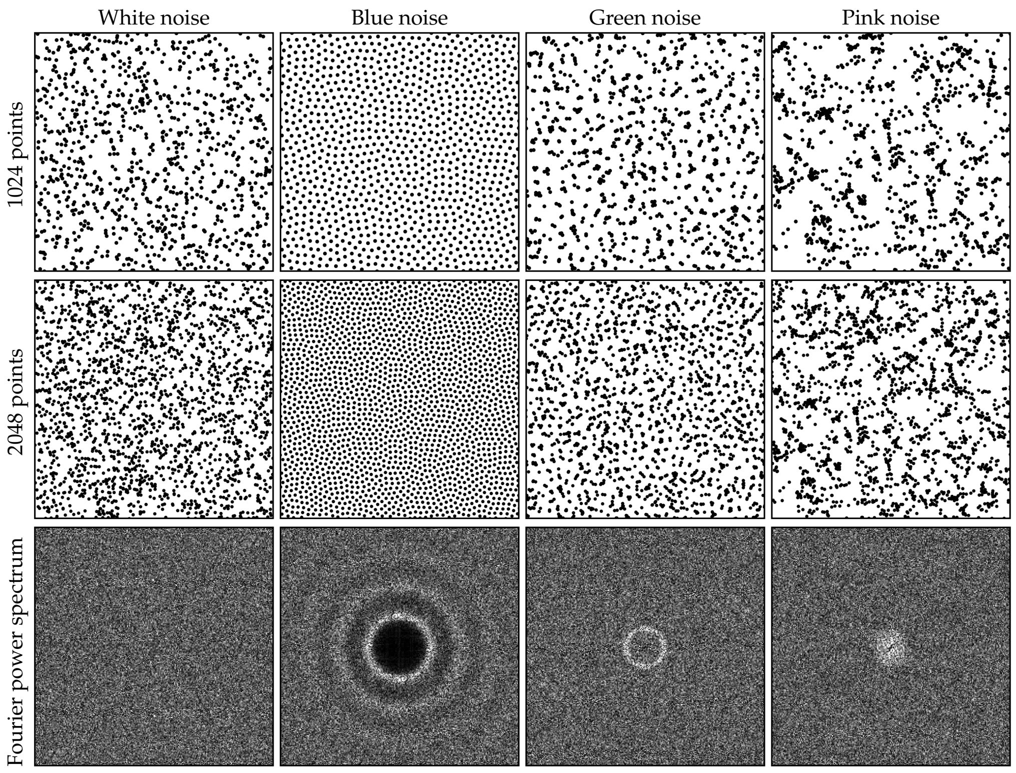


Figure 1.2: *Noise correlations are illustrated as white, blue, green, and pink noise, each generated with different point counts. The "color" terminology refers to the frequency characteristics of their Fourier power spectra, shown in the third row for the 1024-point examples.*

techniques for creation and manipulation, reflecting the vast scope of graphics in modern technology.

1.3.2 Generative modeling for graphics

Generative modeling becomes increasingly important in graphics as it can simplify the traditional generation process of designing 3D objects, lighting, and rendering to generate every pixel from neural networks. There are various ways that generative modeling can be used for graphics generation and editing. For example, texture synthesis algorithms based on optimizing random pixels to user-defined texture patterns are one of the earliest ways for graphics content generation [Efros and Leung, 1999; Efros and Freeman, 2023; Wei and Levoy, 2000]. More recently, generative adversarial nets and diffusion models are getting more and more attention. These models are powerful tools to generate images that can match the distribution of real images. In this thesis, we have also used these tools for graphics generation and editing.

1.3.3 The role of noise correlations

We have mentioned different types of graphics representations and tools, such as generative models for graphics generation and editing.

In this thesis, we take a step forward to understand how noise correlations can impact the generation process of these methods. To achieve this, we need a unified framework to characterize different forms of noise correlations and adapt it to different applications. For example, diffusion models rely heavily on adding noise and denoising given a set of images as training data. Noise correlations can intuitively play an important role in these types of models.

1.4 Our contributions

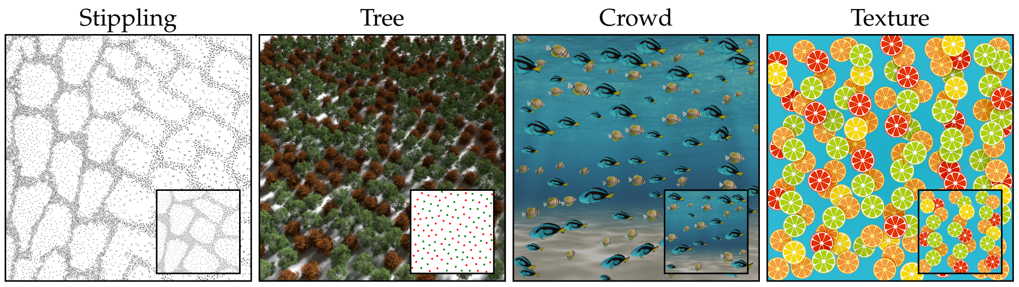


Figure 1.3: *Our proposed methods can be used for placing objects in a virtual environment, creating virtual worlds, generating digital stippling, and texture design. The tree, crowd, and texture results are obtained using our method proposed in Huang et al. [2022], while the stippling result is obtained using our method proposed in Huang et al. [2023]. Input exemplars to create these outputs are shown in the bottom-right inset as either points or rendered elements. This represents a step forward in recreating the distributions observed in the real world, shown in Fig. 1.1, in a more controllable manner compared to previous methods [Roveri et al., 2017; Tu et al., 2019].*

We contribute in two ways. First, we design noise correlation aware filters and editing tools for point pattern authoring, as shown in Fig. 1.3 and Fig. 1.4. Our methods provide different levels of control for generating point patterns, including an example-based approach (Sec. 4.1) and a framework for detailed control over density and correlation via image editing (Sec. 4.2).

Second, we propose first explorations on how to fundamentally change the way of adding and removing noise for training diffusion models (Sec. 5.1). Fig. 1.5 shows our proposed time-varying blue noise model for training diffusion-based image generation. The generated results can benefit from higher-frequency details compared to using random Gaussian noise. Besides designing noise for the diffusion process, we also observe that noise plays an important role in diffusion-based restoration tasks. In particular, we investigate its impact in stereo video generation, a highly challenging problem due to the strict requirements on temporal and view consistency, especially under VR viewing conditions, where visual artifacts are much more perceptible to human observers. We propose to use noisy degradations during the training of diffusion-based stereo video generation, which is useful for video input with varying levels of degradations and achieves simultaneous stereo video generation and restoration (Sec. 5.2).

Our contributions have been published as conference papers, journal papers, or in submission:

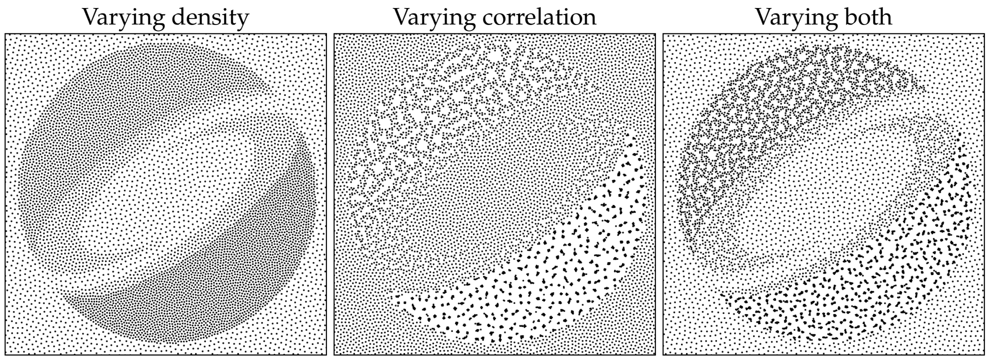


Figure 1.4: *Examples of point patterns with varying density, varying correlation, and varying both density and correlation, generated using our method proposed in Patternshop [Huang et al., 2023]. Compared to the point patterns with the same correlations in Fig. 1.2, our contribution is a framework with spatially-varying density and correlation control based on user preference.*

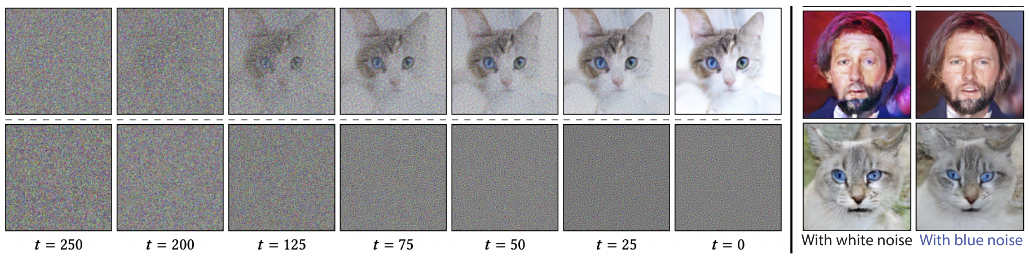


Figure 1.5: *The top row shows the time-varying blue noise correlations introduced by Huang et al. [2024] for diffusion models. The bottom row shows the noise masks that vary from white noise to blue noise. The rightmost two columns show the impact on the generated samples using their proposed noise.*

- Restereo: Diffusion stereo video generation and restoration
  Xingchang Huang, Ashish Kumar Singh, Florian Dubost, Cristina Nader Vasconcelos, Sakar Khattar, Liang Shi, Christian Theobalt, Cengiz Oztireli, Gurprit Singh
  In submission 2025

- Blue noise for diffusion models
  Xingchang Huang, Corentin Salaün, Cristina Vasconcelos, Christian Theobalt, Cengiz Öztireli, Gurprit Singh
  SIGGRAPH (Conference Proceedings) 2024

- Patternshop: Editing Point Patterns by Image Manipulation
  Xingchang Huang, Tobias Ritschel, Hans-Peter Seidel, Pooran Memari, Gurprit Singh
  SIGGRAPH (ACM Transactions on Graphics) 2023

- Point-Pattern Synthesis using Gabor and Random Filters
  Xingchang Huang, Pooran Memari, Hans-Peter Seidel, Gurprit Singh
  Eurographics Symposium on Rendering (Computer Graphics Forum) 2022

Note that we also have follow-ups or related works that are relevant to the above papers.

We will not discuss them in detail in this thesis. The works include:

- Demystifying noise: The role of randomness in generative AI
  Gurprit Singh, **Xingchang Huang**, Jente Vandersanden, Cengiz Öztireli, Niloy Mitra
  Eurographics tutorial 2025 / SIGGRAPH Course 2025
- Online Importance Sampling for Stochastic Gradient Optimization
  Corentin Salaün, **Xingchang Huang**, Iliyan Georgiev, Niloy Mitra, Gurprit Singh
  ICPRAM (full paper) 2025
- Multiple Importance Sampling for Stochastic Gradient Estimation
  Corentin Salaün, **Xingchang Huang**, Iliyan Georgiev, Niloy Mitra, Gurprit Singh
  ICPRAM (short paper) 2025
- Edge-preserving noise for diffusion models
  Jente Vandersanden, Sascha Holl, **Xingchang Huang**, Gurprit Singh
  ICLR (DeLTa workshop) 2025
- TextureShop: Describing Tactile Textures with Pair Correlation Function
  Easa AliAbbasi*, Gabriela Vega*, **Xingchang Huang**, Emiliano Luci, Vahid Babaei, Gurprit Singh, Paul Strohmeier
  In submission 2025

1.5 Thesis outline

This thesis is structured as follows:

- Chapter 1 introduces the research topic, stating the problem and outlining the objectives and the contributions of our work.
- Chapter 2 reviews relevant background knowledge, providing a technical foundation for the following chapters.
- Chapter 3 reviews previous work from point pattern synthesis and editing to image/video generation that is related to noise-aware design.
- Chapter 4 describes our proposed noise-aware methods for point pattern synthesis and editing, including using training-free correlations and embedding point correlations for point pattern synthesis and editing.
- Chapter 5 presents our novel ideas to design noise for diffusion models, including time-varying blue noise for improving diffusion-based image generation, as well as using noise-degraded videos for diffusion-based simultaneous stereo video generation and restoration.
- Finally, Chapter 6 concludes with a summary of the findings, the contributions to the field, and potential directions for future research.
- Appendices and references follow, providing supplementary material and documented sources for each of our works.

Note that in the following chapters, the same figures from our own works will be used, including our works [Huang et al., 2022] [Huang et al., 2023] [Huang et al., 2024] and our work under review.

Chapter 2
Background

In this chapter, we will explain the background knowledge that is required for the following thesis, including different ways of characterizing noise correlations and the basics of generative modeling.

2.1 Characterization of noise correlations

2.1.1 Fourier power spectrum

One well-developed tool to characterize noise correlations is the Fourier transform. This is widely used in signal processing-related tasks where we need to look at the spatial data in the frequency domain to understand the structure of the data.

As shown in Fig. 2.1 (first row), points are sampled and distributed over the 2D space. Each sampling pattern has unique characteristics that affect the distribution of points. For instance, white noise, also known as independent random sampling, involves placing each point in the space independently of the positions of other points. The formulation of sampling in the spatial domain is as follows:

$$\bar{x} = \frac{1}{N} \sum_{k=1}^{N} \delta(\tilde{x} - \tilde{x}_k) \tag{2.1}$$

where $\tilde{x}_k$ are the sampled points, $\bar{x}$ are arbitrary positions over the 2D space, and δ is the Dirac pulse function.

Besides analysis in the spatial domain, there exist other tools for characterizing point sets. The power spectrum of a point set provides a frequency domain representation of the point distribution, revealing the energy or power distribution among various frequency components. It is computed as follows in 2D:

$$E\left[\left|\frac{1}{\sqrt{N}} \sum_{k=1}^{N} e^{-2\pi i(\omega \cdot \tilde{x}_k)}\right|^2\right] \tag{2.2}$$

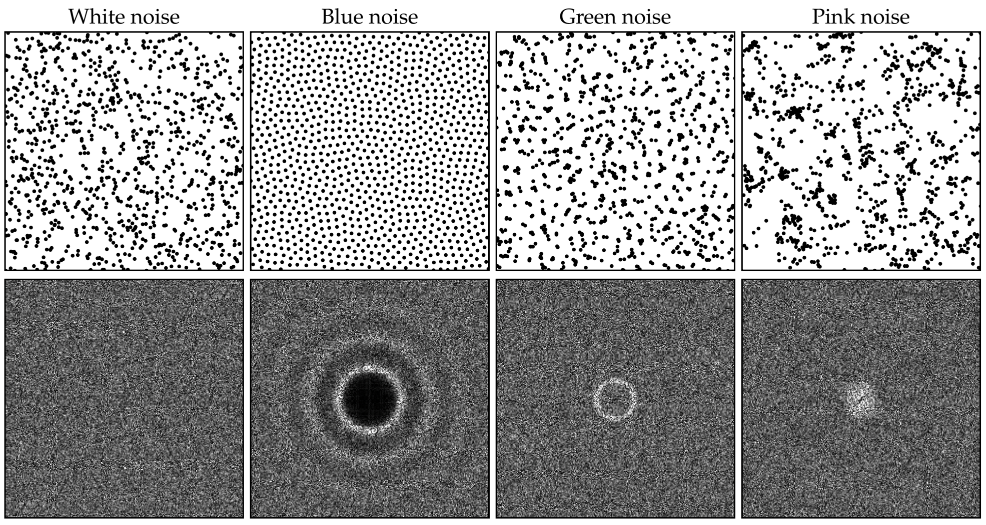


Figure 2.1: *Noise correlations as white, blue, green, and pink noise, and their Fourier power spectrum.*

where $\tilde{x}_k$ are the sampled points, ω are frequencies, and N is the number of points.

For a completely random point set (white noise), the power spectrum is expected to be flat because there is no preferential scaling or repetition of structures at any particular frequency, as shown in Fig. 2.1 (first column). While for other point sets, such as blue noise, their power spectrum shows no energy in the low-frequency components. Therefore, the Fourier power spectrum is a useful mathematical tool to distinguish different noise characteristics of point sets.

2.1.2 Pair correlation function

The key idea of the pair correlation function (PCF) is to count neighbors within rings of increasing radius around each point [Ecormier-Nocca et al., 2019]. In other words, PCF requires computing the distance between points and mapping the distances to predefined histograms. The following two operations are required to generate density-invariant, continuous PCF curves:

First, distances are normalized according to overall exemplar density. Given an input exemplar with n points that over a 2D unit square, distances are divided by a value r_{max} based on how far points would be separated if they were maximally spread to occupy the whole input domain. r_{max} is given by (see Lagae and Dutré [2006]):

$$r_{\text{max}} = 2\sqrt{\frac{1}{2\sqrt{3}n}} \tag{2.3}$$

Second, to ensure the function is continuous and robust to noise, the influence of a neighbouring point is spread using a Gaussian Kernel $k_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-x^2/2\sigma^2}$, centered at the distance r from the reference point. In the work of Ecormier-Nocca et al. [2019], they use the following version for point patterns over a 2D unit square:

$$\text{PCF}(r) = \frac{1}{A_{r,n}^2}\sum_{i \neq j} k_\sigma(r - d_{ij}) \tag{2.4}$$

where $A_{r,n}$ is the area of the ring with inner radius $r - 1/2$ and outer radius $r + 1/2$, and d_{ij} is the distance between reference point P_i and neighbour P_j.

Eq. (2.4) often represents "mean PCF", which is different from "individual PCF" associated to each point as shown in the following:

$$\mathrm{PCF}(P_i, r) = \frac{1}{A_{r,n}} \sum_{j \neq i} k_\sigma(r - d_{ij}) \tag{2.5}$$

Later in Sec. 4.2, we will reformulate PCF computation in Eq. (2.5) to consider an additional guiding function. If the guiding function is a constant, our formulation falls back to the above equation (Eq. (2.5)). In this case, the area term is basically considered as a normalization term that takes the boundary handling into account.

2.1.3 Gram and correlation matrices

Gram and correlation features [Gatys et al., 2015a] based on VGG [Simonyan and Zisserman, 2014] are mostly used for image-based texture synthesis and stylization.

In image-based texture synthesis, Gatys et al. [2015a] propose to use the pre-trained VGG-19 [Simonyan and Zisserman, 2014] feature maps to extract local and non-local correlations. These extracted feature maps are then optimized for image expansion or stylization during synthesis using the following losses:

Gram matrix. Gram matrix [Gatys et al., 2015a] is defined as the following:

$$G^l_{t_1,t_2} = \frac{1}{n^l W^l H^l} f^l_{t_1} \cdot f^l_{t_2} \tag{2.6}$$

where n^l, W^l, H^l are the number, width and height of $2D$ feature maps at layer l, respectively. $G^l_{t_1,t_2}$ is the gram matrix computed between t_1^{th} and t_2^{th} feature maps at layer l and $f^l_{t_1}, f^l_{t_2}$ are t_1^{th} and t_2^{th} feature maps at layer l.

Gram loss. Gram loss L_{gram} can be defined as the square error between the output and input Gram matrices:

$$L^l_{gram} = \sum_{t_1,t_2} (\widetilde{G}^l_{t_1,t_2} - G^l_{t_1,t_2})^2 \tag{2.7}$$

where $\widetilde{G}$, G represent Gram matrices of output and input patterns, respectively.

Correlation matrix. To better capture the correlations, Sendik and Cohen-Or [2017] compute a correlation matrix from the feature maps:

$$C^{t,l}_{a,b} = \frac{f^{t,l}_{ij} f^{t,l}_{i-a,j-b}}{[(W^l - |a|)(H^l - |b|)]} \tag{2.8}$$

where $a \in [-W^l, W^l], b \in [-H^l, H^l]$. We use zero-padding when $i - a, j - b$ are outside the boundary. t denotes the index of feature maps, meaning that the deep correlation matrix is computed at each channel of feature maps independently.

Correlation loss. Based on this, deep correlation loss L_{corr} is defined as:

$$L^l_{corr} = \sum_{a,b,t} (\widetilde{C}^{t,l}_{a,b} - C^{t,l}_{a,b})^2 \tag{2.9}$$

where $\widetilde{C}, C$ represent correlation matrices of output and input patterns, respectively.

Tu et al. [2019] proposed to use the idea of image-based texture synthesis for point pattern synthesis. To bridge the gap between image-based representation and point-based representation, they propose a differentiable point splatting module with Gaussian kernels. In this way, they can use both the Gram and Correlation loss to perform point pattern synthesis and expansion. Similarly, Reddy et al. [2020] proposed to use the Gram matrix to perform pattern expansion. Unlike Tu et al., the input for Reddy et al. is an image with corresponding elements. Our approach [Huang et al., 2022] also uses Gram and Correlation loss. Different from Tu et al. [2019], we propose a different representation including Gabor transform with multi-scale random features, to better extract local and non-local correlations from an input point pattern in a more efficient manner (Sec. 4.1).

2.1.4 Correlated Gaussian noise for modeling pixel correlations

For some specific type of noise, such as Gaussian noise, there is a unique way to characterize noise correlation using the covariance matrix.

Definition. A random vector $\mathbf{X} \in \mathbb{R}^d$ is said to follow a *multivariate normal distribution*, denoted $\mathbf{X} \sim N_d(\boldsymbol{\mu}, \Sigma)$, if **every** linear combination $\sum_{i=1}^d a_i X_i$ is univariate normal for all $(a_1, \dots, a_d) \in \mathbb{R}^d$.

Parameters.

- *Mean vector* $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_d \end{bmatrix} \in \mathbb{R}^d$.
- *Covariance matrix* $\Sigma = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_{dd} \end{bmatrix} \succ 0$, which is symmetric and **strictly positive-definite** ($\mathbf{v}^\top \Sigma \, \mathbf{v} > 0$ for all non-zero $\mathbf{v} \in \mathbb{R}^d$). Diagonal entries are variances, off-diagonals are covariances.

Probability density function. The probability density function is as follows:

$$\frac{1}{(2\pi)^{d/2} \, |\Sigma|^{1/2}} \exp\Big\{-\tfrac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\Big\}, \qquad \mathbf{x} \in \mathbb{R}^d.$$

Sampling. To draw $\mathbf{X} \sim N_d(\boldsymbol{\mu}, \Sigma)$:

1. Compute a *Cholesky decomposition* $\Sigma = LL^\top$ with L lower-triangular.
   (If numerical conditioning is poor, replace Σ by $\Sigma + \varepsilon I$ for a small $\varepsilon > 0$.)
2. Generate a standard normal vector $\mathbf{Z} \sim N_d(\mathbf{0}, I_d)$.

3. Set
$$\boxed{\mathbf{X} = \boldsymbol{\mu} + L\mathbf{Z}}$$
Then $\mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}$ and $\mathrm{Cov}[\mathbf{X}] = LI_dL^\top = \Sigma$.

The sampling procedure is particularly useful for generating correlated Gaussian noise, with user-defined mean and covariance matrices. In our work [Huang et al., 2024], we use this sampling procedure to sample our proposed time-varying Gaussian blue noise in an efficient manner. You may find more details and properties on correlated Gaussian noise from Orduz [2019].

2.2 Generative modeling

2.2.1 Neural texture synthesis

Neural texture synthesis is a form of generative modeling, as it usually learns a probability distribution over input textures and provides a mechanism for sampling new, previously unseen realizations from that distribution. Note that we will discuss both image-based and point-based texture synthesis in the following. Different from traditional patch-based texture synthesis techniques, which typically operate by copying and arranging small patches of an input texture, neural texture synthesis methods often better capture global structures.

Neural image-based texture synthesis. For image textures, neural texture synthesis utilizes deep neural networks to model the complex statistical relationships inherent in natural textures. More specifically, these methods learn an underlying representation of textures from a given image. Then, a loss function is designed to minimize the difference between the statistical features of the synthesized image and those of a target texture. In this way, we can obtain results with better local and global structures similar to the exemplar input.

A common framework involves using a pre-trained convolutional neural network (CNN), such as VGG19 [Simonyan and Zisserman, 2015], to extract feature representations at various layers. The textural information is captured by matching the Gram matrices [Gatys et al., 2015a] of the synthesized image and the target texture at selected layers, as discussed in Sec. 2.1.3. This process ensures that the synthesized texture retains the perceptual qualities of the original image at multiple scales.

Extension to point patterns. Extending neural texture synthesis to point patterns presents unique challenges, primarily due to the unstructured nature of point data. Tu et al. [2019] is one of the first to address this by differentiably rasterizing a point pattern into an image, which is then turned into a stack of 2D feature maps via the VGG-19 network. Subsequent steps for optimization with Gram and Correlation losses are the same as image-based methods.

2.2.2 Generative adversarial networks

Generative Adversarial Networks (GANs) are a class of artificial neural networks introduced by Goodfellow et al. [2014]. They have revolutionized the field of generative modeling by enabling the generation of highly realistic synthetic data.

GANs consist of two neural networks—the generator and the discriminator—engaged in a zero-sum game. The core concept of GANs lies in their adversarial process: the discriminator's goal is to accurately classify data as real or fake, while the generator aims to produce data that is sufficiently convincing to be classified as real by the discriminator. This adversarial competition drives both networks to improve continually, resulting in the generation of high-quality data.

The interaction between the generator G and the discriminator D can be described by the following min-max game:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

where $\mathbf{x}$ is a real data instance, $\mathbf{z}$ is a latent sample typically drawn from a normal distribution, and p_{data} is the data distribution.

GANs have found a large variety of applications in computer vision and graphics, where they are used for tasks such as photo-realistic image generation, image-to-image translation, etc. Our work [Huang et al., 2023] (Sec. 4.2) also utilized the idea of GAN for high-quality image-to-image translation for analyzing the density and correlation given a point pattern as input. With the high-quality estimated density and correlation, not only can point patterns be re-synthesized to generate new realizations, but users can also have full control over editing density and correlation independently to achieve various artistic pattern creations.

2.2.3 Diffusion models

In recent years, diffusion models have represented a cutting-edge class of generative models that convert a simple noise distribution, typically Gaussian, into complex data distributions through a controlled denoising process. These models are renowned for their capability to generate high-quality, photorealistic samples, surpassing the results of models like GANs.

Diffusion models differ significantly from other generative models like GANs, which rely on adversarial training, and Variational Autoencoders (VAEs) [Kingma et al., 2013], which are based on variational inference. Unlike these models, diffusion models employ a non-adversarial training process and focus on modeling the data distribution through a sequence of generative and reverse steps. The conceptual framework for diffusion models is derived from the principles of thermodynamics in statistical physics, specifically the process of reversing a diffusion. The underlying principles have been adapted to model the transformation of distributions, as demonstrated in the pioneering work of Sohl-Dickstein et al. [2015].

The operation of diffusion models can be divided into two distinct processes:

- **Forward Process:** This involves gradually adding Gaussian noise to data over a series of steps, transitioning from an original data distribution to a noise distribution.
- **Backward Process:** This is essentially the reverse of the forward process, where a model, typically parameterized by neural networks, learns to reconstruct the original data from the noise.

Mathematically, these processes can be described by the equations [Ho et al., 2020]:

$$\text{Forward: } \mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1 - \alpha_t}\epsilon, \quad \epsilon \sim N(0, I) \tag{2.10}$$

$$\text{Backward: } \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon_\theta(\mathbf{x}_t, t)\right), \tag{2.11}$$

where α_t are constants that control the noise levels at each diffusion step, $\mathbf{x}_t$ are noisy images, and ϵ_θ represents the model's estimation of the noise. Similar to GANs, diffusion models have found a wide array of applications, for image generation, text-to-image generation, image editing, etc.

However, most existing diffusion models rely on Gaussian noise, and relatively little research has explored whether alternative noise types could also benefit diffusion model training and sampling. In our work [Huang et al., 2024], we take a first step in this direction by investigating the use of correlated noise, specifically blue noise, in diffusion processes. Our motivation comes from the rendering community, where blue noise has been shown to be effective for Monte Carlo denoising. Since diffusion models can be interpreted as iterative denoising procedures, we explore how incorporating blue noise during both the forward (noise addition) and backward (denoising) processes might influence performance. Our experiments demonstrate that using a time-varying blue noise schedule can lead to improved generation quality, particularly in preserving fine image details. More details are provided in Sec. 5.1.

Chapter 3
Related work

In this chapter, we review existing literature that is related to our work, ranging from point pattern synthesis and editing to generative diffusion models.

3.1 Example-based point pattern synthesis

By-example pattern synthesis is well-studied in the past decade, including image-based [Gatys et al., 2015a,b; Ustyuzhaninov et al., 2016], surface-based [Turk, 2001], discrete [Ma et al., 2011; Öztireli and Gross, 2012; Ma et al., 2013; Roveri et al., 2015, 2017; Tu et al., 2019], and continuous [Tu et al., 2020] texture synthesis. In general, these methods are designed for synthesizing a larger pattern given a small exemplar as input. Our approach is inspired by recent developments in point pattern [Tu et al., 2019] and texture [Ustyuzhaninov et al., 2016; Guehl et al., 2020] synthesis. In this section, we cover the most relevant works and direct interested readers to a recent survey [Gieseke et al., 2021] for an extensive study.

Image-based texture synthesis. Image-based texture synthesis considers synthesizing image texture from an exemplar in the pixel space. Heeger and Bergen [1995] propose to use image pyramid whereas Portilla and Simoncelli [2000] use wavelet-based descriptors for texture synthesis. We instead use a Gabor filter bank to extract exemplar features. More recently, Guehl et al. [2020] presented Point Process Texture Basis Functions (PPTBF), together with stringed Gabor functions to synthesize a large variety of binary textures. In Chapter 4, we show how our work is designed for by-example point-pattern synthesis.

By-example point pattern synthesis. Point pattern synthesis is typically defined as by-example synthesis: given a small point set as input, algorithms are designed to synthesize a larger one that preserves both global and local structure. Ma et al. [2011] proposed a sample- or multi-sample-based representation for point pattern synthesis, or in general, element synthesis that allows example-based synthesis. Roveri et al. [2015] proposed a meshless representation where structures in the example and output are converted into a functional representation. However, both methods are neighborhood-based, meaning that they can easily fail to synthesize patterns that preserve global structure well. A more recent work on point pattern synthesis is from Tu et al. [2019], which uses the VGG-19 network for point pattern synthesis. Though they achieve improved results in terms of preserving global structure, they struggle with synthesizing local structures faithfully.

Other than point patterns, there are works on extending points to shapes. Ecormier-Nocca et al. [2019] extended the pair correlation function (PCF) framework to disk distributions. Landes et al. [2013] proposed a shape-aware model that brings out the elements as polylines, which better preserves element distances. Similarly, Barla et al. [2006] developed a specific method for stroke pattern synthesis and Hurtut et al. [2009] considered the appearance of vector elements during synthesis.

Neural networks for pattern synthesis. Resurgence of neural networks through image classification [Krizhevsky et al., 2012] has changed the dynamics of computer graphics research over the past decade. Recent architectures work on both image and unstructured point clouds domains. Point-based neural networks [Qi et al., 2017a,b; Wang et al., 2019] extend the deep learning approach over irregular point clouds. However, these methods do not take into account point correlations and are not designed for point pattern synthesis. Leimkühler et al. [2019] propose a convolutional architecture that directly optimizes point samples for prescribed Fourier (power spectra) or spatial (pair correlation function) statistics, allowing *blue-, green-. or pink-noise* sample distributions. However, this mainly captures global correlations. In texture synthesis, neural networks have been recently employed to synthesize large canvas textures from small exemplars using pre-trained network features [Gatys et al., 2015b; Sendik and Cohen-Or, 2017] or completely random filters [Ustyuzhaninov et al., 2016; He et al., 2016]. Similarly, Ulyanov et al. [2018] proposed a deep image prior using a randomly initialized neural network for image restoration. Aberman et al. [2018] used pre-trained CNN features to find image correspondence and Bojanowski et al. [2017] presented a latent space optimization technique for generative image synthesis. In Chapter 4, we show that our method, on the other hand, optimizes and synthesizes points instead of images.

Compared to these, using neural networks for point pattern synthesis is a relatively unexplored area. Tu et al. [2019] proposed the first neural network-based idea for point pattern synthesis. They propose an optimization approach that uses a pre-trained VGG-19 network [Simonyan and Zisserman, 2015] to guide random point samples to follow structured patterns from an exemplar. Reddy et al. [2020] developed a differentiable compositing pipeline that allows current deep learning based image methods to effectively handle patterns. Compared to their work, we propose to use Gabor features and a convolutional filtering step that better captures the statistics of point patterns. This enables higher-quality by-example point pattern synthesis and can be extended to multi-class and multi-attribute patterns with minor changes. No training or pre-trained features are required in our framework. It is also worth mentioning that Tu et al. [2019] propose an additional soft optimization scheme to heuristically remove outliers from optimized patterns. However, this may still result in missing points in the output patterns. In Chapter 4, we show that we do not apply any heuristic approach during optimization.

3.2 From point pattern synthesis to editing point correlations

Point patterns are usually characterized by density and correlation, where point correlations are more involved in characterization and editing. In this section, we discuss how previous work considered editing point patterns in terms of density and correlation.

Sample correlations. Correlations among stochastic samples are widely studied in computer graphics. From halftoning [Ulichney, 1987], reconstruction [Yellott, 1983], anti-aliasing [Cook, 1986; Dippé and Wold, 1985] to Monte Carlo integration [Durand, 2011; Singh et al., 2019], correlated samples have made a noticeable impact. Recent works [Xu et al., 2020; Zhang et al., 2019] have also shown the direct impact of correlated samples on training accuracy in machine learning. Among different sample correlations, *blue* noise [Ulichney, 1987] is the most prominent in the literature. Blue noise enforces point separation and is classically used for object placement [Kopf et al., 2006; Reinert et al., 2013] and point stippling [Deussen et al., 2000; Secord, 2002; Schulz et al., 2021]. However, modern approaches do not insist on point separation in faithful stippling [Martín et al., 2017; Kim et al., 2009; Deussen and Isenberg, 2013; Rosin and Collomosse, 2012]. Different kinds of *colored* noises (e.g., green, red) are studied in literature [Lau et al., 1999; Zhou et al., 2012] for half-toning and stippling purposes. But the space spanned by these point correlations is limited to a few bases [Öztireli and Gross, 2012]. We propose an extended space of point correlations that helps express a large variety of correlations. Our framework embeds these correlations in a two-dimensional space, allowing representation of different correlations with simple two-channel color maps. This makes analysis and manipulation of correlations easier by using off-the-shelf image editing software.

Analysis. To analyze different sample correlations, various spectral [Ulichney, 1987; Lagae and Dutre, 2008] and spatial [Wei and Wang, 2011; Öztireli and Gross, 2012] tools are developed. For spectral methods, the Fourier power spectrum and its radially averaged version are used to analyze sample characteristics. In the spatial domain, PCF is used for analysis, which evaluates pairwise distances between samples that are then binned in a 1D or 2D histogram.

Synthesizing blue-noise correlation. Blue noise sample correlation is most commonly used in graphics. There exist various algorithms to generate blue noise sample distributions [Yan et al., 2015]. Several optimization-based methods [Lloyd, 1982; Balzer et al., 2009; Liu et al., 2009; Schmaltz et al., 2010; Fattal, 2011; De Goes et al., 2012; Heck et al., 2013; Kailkhura et al., 2016; Qin et al., 2017], as well as tiling-based [Ostromoukhov et al., 2004; Kopf et al., 2006; Wachtel et al., 2014] and number-theoretic approaches [Ahmed et al., 2015, 2016, 2017] have been developed over the past decades to generate blue noise samples.

Synthesizing other correlations. There exist methods to generate samples with different target correlations. For example, Zhou et al. [2012]; Öztireli and Gross [2012] proposed to synthesize point correlations defined from a user-defined target PCF. Wachtel et al. [2014] proposed a tile-based optimization approach that can generate points with a user-defined target power spectrum. All these methods require heavy machinery to ensure the optimization follows the target. Leimkühler et al. [2019] simplified this idea and proposed a neural network-based optimization pipeline. All these approaches, however, require the user to know how to design a *realizable* PCF or a power spectrum [Heck et al., 2013]. This can severely limit the usability of point correlations to only a handful of expert users. Our framework lifts this limitation and allows us to simply use a two-dimensional space to define correlations. Once a user picks a 2D point —which we visualize as a color of different chroma— our framework automatically finds the corresponding PCF from the embedded space and synthesizes the respective point correlations. It is also straightforward to generate spatially varying correlations using our framework by defining a range of colors as correlations. So far, only Roveri et al. [2017] has synthesized spatially varying correlations, but it remains limited by how well a user can design PCFs.

Image and point pattern editing. Editing software allows artists to tailor the digital content to their needs. Since *color image* is the easily available data representation, today's software is specifically designed to process three-channel (RGB) images. Modern pipelines allow effects like relighting [Sun et al., 2019], recoloring, image retargeting [Rott Shaham et al., 2019], inpainting [Bertalmio et al., 2000], style transfer [Gatys et al., 2016] and texture synthesis [Efros and Freeman, 2001; Zhou et al., 2018; Sendik and Cohen-Or, 2017] to be performed directly in the three-channel RGB representation. However, most digital assets e.g.,, materials, color pigments, light fields, patterns, sample correlations, are best captured in high-dimensional space. This makes it hard for image-based software to facilitate editing these components. A lot of research has been devoted in the past to support editing materials [Pellacini and Lawrence, 2007; An et al., 2011; Di Renzo et al., 2014], light fields [Jarabo et al., 2014], color pigments [Sochorová and Jamriška, 2021], natural illumination [Pellacini, 2010] with workflows similar to images.

Synthesizing textures with elements [Ma et al., 2011] [Reinert et al., 2013] [Emilien et al., 2015] [Reddy et al., 2020] [Hsu et al., 2020] and patterns with different variations [Guerrero et al., 2016] using 2D graphical primitives has also been proposed. These works focus on updating point and element locations to create patterns for user-specific goals and designing a graphical interface for user interactions. However, none of the previous work allows editing spatial-varying point correlations intuitively. In this work, we propose a pipeline to facilitate correlation editing using simple image operations. Instead of directly working with points, we encode their corresponding spectral or spatial statistics in a low-dimensional (3-channel) embedding. This low-dimensional latent space can then be represented as an image in order to allow artists to manipulate point pattern designs using any off-the-shelf image editing software. There exists previous work that encodes point correlations [Leimkühler et al., 2019] for a single target or point pattern structures [Tu et al., 2019] using a neural network. But these representations do not disentangle the underlying density and correlation, thereby not facilitating editing operations.

Latent and perceptual spaces. Reducing high-dimensional signals into low-dimensional ones has several benefits. Different from latent spaces in generative models, which are still high-dimensional, such as for StyleGAN [Abdal et al., 2019], the application we are interested in here is a usability one, where the target space is as low as one-, two- or three-dimensional, so users can actively explore it, e.g., on a display when searching [Duncan and Humphreys, 1989]. This is common to do for color itself [Fairchild, 2013; Nguyen et al., 2015]. Ideally, the embedding into a lower dimension is done such that distance in that space is proportional to perceived distances [Lindow et al., 2012]. This was pioneered by Pellacini and Lawrence [2007] for BRDF, with a methodology close to ours (MDS). Other modalities, such as acoustics [Pols et al., 1969], materials [Wills et al., 2009], textures [Henzler et al., 2019] and even fabricated objects [Piovarči et al., 2018] were successfully organized in latent spaces.

3.3 Connection between point- and pixel-space noise correlations

Until now, we have discussed noise correlations in the space of points, namely point patterns, characterized by point density and correlation. The same concept about correlation can be used in the image/pixel space. For example, noise correlation such as blue noise has been used for sampling light paths for ray tracing and Monte-Carlo rendering proposed by Georgiev and Fajardo [2016]. But the underlying blue noise masks are generated in the image space.

3.4 Correlated Gaussian noise for image diffusion models

Blue noise. Blue noise is a type of noise characterized by high-frequency signals and the absence of low frequencies. It has found numerous applications in computer graphics. One such application is the utilization of blue noise masks, originally introduced by Ulichney [1993], for image dithering to enhance their perceptual quality. Blue noise masks are also employed in Monte Carlo rendering to improve the distribution of error, as demonstrated by Georgiev and Fajardo [2016] and Heitz and Belcour [2019]. The relationship between blue noise and denoising in rendering has been further explored by Chizhov et al. [2022] and Salaün et al. [2022], revealing that combining blue noise with a low-pass filter can reduce perceptual errors. To leverage the advantageous denoising properties of blue noise masks, we propose to use them as additive noise to corrupt the data for diffusion-based generative modeling.

Diffusion models. There are various formulations for image generation by diffusion models, including stochastic [Ho et al., 2020; Song and Ermon, 2019; Song et al., 2021a] and deterministic ones [Song et al., 2021b; Heitz et al., 2023]. Diffusion models also extend beyond image generation to video generation [Ho et al., 2022] and 3D content generation [Poole et al., 2023]. More comprehensive reviews can be found in the surveys by Cao et al. [2024] and Po et al. [2023].

Diffusion models are known to be slow to train, as well as slow in the generative process. How to speed up the generative process to generate images in a few steps becomes an increasingly important research topic [Lu et al., 2022; Liu et al., 2023a,b; Salimans and Ho, 2022; Karras et al., 2022, 2023; Song et al., 2023; Luo et al., 2023]. Orthogonal to reducing the number of inference steps, some work focuses on developing a more general framework that can support various types of noise addition [Jolicoeur-Martineau et al., 2023] or image corruption operations [Bansal et al., 2024]. Some work explicitly takes the frequency of the image content into account to model the generative process in a coarse-to-fine manner [Rissanen et al., 2023; Phung et al., 2023]. However, there exists limited work researching how the frequency of noise used in image corruption can make an impact on the denoising process in diffusion-based generative modeling. To understand this problem, we propose a framework in Sec. 5.1 utilizing correlated noise to improve the denoising process.

3.5 Noisy degradations for stereo video generation and restoration

Further, we study how noise can be used in image degradations for simultaneous stereo video generation and restoration. The motivation is that stereo video generation is a highly challenging problem due to the strict requirements on temporal and view consistency, especially under VR viewing conditions, where visual artifacts are much more perceptible to human observers. However, most existing methods focus on generating 3D stereoscopic videos from monocular 2D videos, assuming that the input monocular video is of high quality. We want to introduce a novel pipeline that not only generates stereo videos but also enhances both left-view and right-view videos consistently with a single model.

In the following, we review papers on diffusion-based stereo image and video generation methods, as well as video restoration methods. Here, we mainly focus on diffusion-based methods for stereo image and video generation, as they represent the current state-of-the-art quality. As discussed in StereoCrafter [Zhao et al., 2024], methods relying on 3D representation for reconstruction, like NeRF [Mildenhall et al., 2021], 3DGS [Kerbl et al., 2023], are not the best choice for stereo generation. These methods require both capturing a lot of frames for each scene and estimating the camera pose of each frame from the input videos. It becomes challenging for these methods based on single-view inputs, with dynamic objects, or visual effects such as fog or fire. While some work based on 3D representation, such as DynIBaR [Li et al., 2023a], RoDynRF [Liu et al., 2023c], can be robust to monocular inputs, they can still contain artifacts and without the capability to generate content.

Training-free diffusion-based stereo generation. Wang et al. [2024] propose StereoDiffusion, to generate stereo images via copying and shifting in the latent space of pretrained image diffusion models. This method is training-free but requires inpainting in the latent space, which can be error-prone and does not scale well to stereo video generation. The RePaint algorithm proposed by Lugmayr et al. [2022] can be repurposed for stereo image generation, similar to the latent inpainting part in Wang et al. [2024]. Dai et al. [2024] propose a training-free method for stereo video generation using pretrained video diffusion models. Compared to Wang et al. [2024], this works better in terms of temporal consistency, but still relies on a pretrained model that is not trained for stereo video generation. Although there exist off-the-shelf video inpainting tools [Li et al., 2022; Zhou et al., 2023], they might generate blurry content and inconsistent artifacts as they are not designed to be integrated into the pipeline. Immersity AI imm and Owl3D owl are 2D-to-3D conversion software that can generate more consistent results. However, all these methods are designed to maintain the details of the left-view input and assume the input is high-quality.

Training-based diffusion-based stereo generation. Recently proposed training-based methods show promising results with spatial and temporal consistency. StereoCrafter [Zhao et al., 2024], SpatialMe [Zhang et al., 2024], StereoConversion [Mehl et al., 2024], Immerse-Pro [Shi et al., 2024a]. But these methods often require a large-scale dataset of stereo videos from the internet or movies, which is usually expensive to capture and annotate. They may require a complicated pipeline for data preprocessing, stereo matching to obtain the video depth, and warped video as the training data. One example is shown in the StereoCrafter [Zhao et al., 2024], where accurate stereo matching is crucial for subsequent steps. We adopt a training-based strategy as well, but with synthetic data, as they are easier to obtain without complicated data preprocessing. Similarly, these models are trained to preserve the details of input videos, without considering that the input can be low-quality.

Multi-view and video diffusion models. Recent progress on multi-view and video diffusion models has shown great capability in novel view synthesis. CAT3D [Gao et al., 2024] supports novel view synthesis from single- or multi-view images, combining multi-view diffusion and NeRFs. Xie et al. [2024] extend Stable Video Diffusion (SVD) [Blattmann et al., 2023] to Stable Video 4D (SV4D), which can reconstruct a 4D scene from a single input video. But their method only considers a foreground animated object without a background. Similar work includes Generative Camera Dolly [Van Hoorick et al., 2024] and CAT4D [Wu et al., 2024]. But these works are focused on novel view synthesis from large camera baselines and are not directly usable for stereo video generation.

Video restoration. Here, we discuss some work on video restoration as it is related to our idea of restoring low-quality video while doing stereo video generation. We are inspired by Real-ESRGAN [Wang et al., 2021], which proposed to train image super-resolution models via degrading synthetic data. Liang et al. [2022, 2024] propose video restoration transformers (e.g., RVRT, VRT) for temporally coherent video super-resolution, deblurring, denoising, etc. DiffBIR [Lin et al., 2024] and DiffIR2VR-Zero [Yeh et al., 2024] start to consider using generative diffusion before improving generalization capability across various degradation types and datasets. Upscale-A-Video [Zhou et al., 2024] is a diffusion-based video restoration model, but it requires significant GPU memory to run.

Flow-guided video super-resolution models, such as BasicVSR [Chan et al., 2021], BasicVSR++ [Chan et al., 2022], are another type of models that can achieve video restoration efficiently with spatial-temporal consistency. FMA-Net [Youk et al., 2024] further advances the quality of joint video super-resolution and deblurring via carefully designed flow-guided modules. Incorporating these off-the-shelf models for stereo video generation and restoration requires more storage and resources to run. The additional restoration step, without being integrated into the stereo generation pipeline, can introduce artifacts and inconsistencies that are more visible in the stereo setup.

Chapter 4
Noise correlations for point pattern synthesis and editing

In this chapter, we summarize our two projects and contributions on point pattern synthesis and editing, respectively.

- Our first project (Sec. 4.1) considers addressing the challenges of example-based point pattern synthesis. We propose a method based on training-free correlations, which characterize any input point pattern in a holistic way without additional training.
- We find in the first project that the holistic encoding of point patterns can limit fine-grained user control of density and correlation. Therefore, in the second project (Sec. 4.2), we propose to address this issue by decoupling density and correlation. This allows us to perform controllable point pattern editing for density and correlation separately.

4.1 Training-free correlations for point pattern synthesis

Point pattern synthesis requires capturing both local and non-local correlations from a given exemplar. Recent works employ deep hierarchical representations from the VGG-19 [Simonyan and Zisserman, 2015] convolutional network to capture the features for both point-pattern and texture synthesis. In this work, we develop a simplified optimization pipeline that uses more traditional Gabor transform-based features. These features, when convolved with simple random filters, give highly expressive feature maps. The resulting framework requires significantly fewer feature maps compared to VGG-19-based methods [Tu et al., 2019; Reddy et al., 2020], better captures both the local and non-local structures, does not require any specific data set training, and can easily extend to handle multi-class and multi-attribute point patterns, e.g.,, disk and other element distributions. To validate our pipeline, we perform qualitative and quantitative analysis on a large variety of point patterns to demonstrate the effectiveness

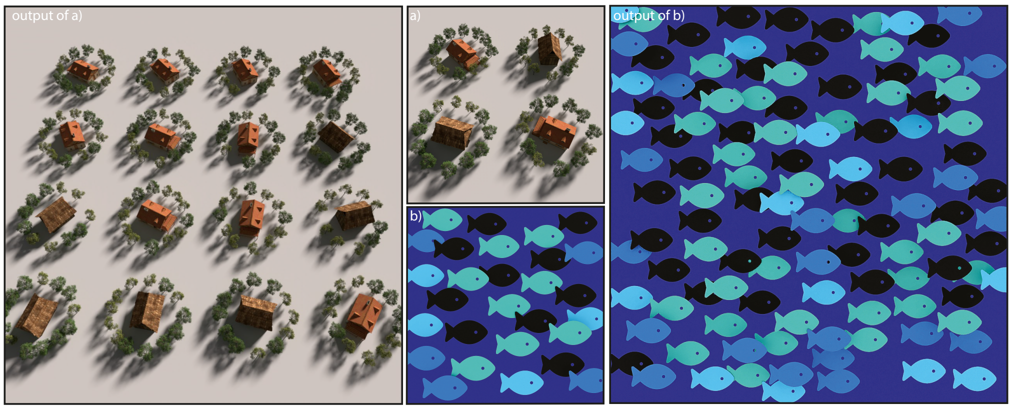


Figure 4.1: *Our method takes simply a point set (with positions, classes, attributes) as input and applies a continuous Gabor transform to extract features. We then use these Gabor features to synthesize a larger-scale output. We show synthesis results of a 2-class point pattern in (a), and a 4-class point pattern with depth and scale as attributes in (b).*

of our approach. Finally, to better understand the impact of random filters, we include a spectral analysis using filters with different frequency bandwidths.

4.1.1 Introduction

Synthesizing point patterns from small exemplars has various applications, from object placement, discrete texture synthesis to creative pattern generation. Synthesizing such patterns has been an active area of research in computer graphics [Lewis, 1989; Deussen et al., 1998; Öztireli and Gross, 2012; Roveri et al., 2015; Tu et al., 2019] and involves two major steps. The first step requires robustly characterizing the underlying correlations from an exemplar. The second step involves point-pattern expansion while preserving the underlying local and non-local correlations.

Characterizing different point patterns is traditionally performed using Fourier and spatial (pair correlation function) tools. Among different point-patterns, blue noise point patterns [Ulichney, 1988] have received special attention in computer graphics due to their minimum distance preservation [Yellott, 1983; Yan et al., 2015]. Blue-noise patterns are named *blue noise* based on their power spectrum characteristics. But not all patterns are easily characterised by traditional Fourier or spatial (pair correlation function) tools. Many recent works [Zhou et al., 2012; Leimkühler et al., 2019; Heck et al., 2013; Roveri et al., 2017] have devoted significant effort to characterising such patterns by matching the low-level statistics.

Synthesizing a large canvas output from an exemplar requires faithful extraction of both local and non-local features. To tackle this, various example-based approaches have been proposed [Ma et al., 2011, 2013; Roveri et al., 2015; Tu et al., 2019; Ecormier-Nocca et al., 2019; Landes et al., 2013] that take as input a small-scale point set, extract both local and non-local features from the exemplars, and use these features to synthesize point patterns on a larger scale.

Recently, Tu et al. [2019] use a pre-trained VGG-19 network [Simonyan and Zisserman, 2015] to extract point pattern correlations. VGG-19 features are used as local and non-local features of an exemplar point set. These features are then used as a target during an optimization process to synthesize large canvas point patterns. Their method shows that hierarchical image-space features and losses are powerful for point pattern synthesis. However, the patterns synthesized by this method struggle to preserve accurate local structures and require multiple steps of optimization or manual intervention for refinement. Moreover, since the VGG-19 network is limited by 2D images as input, it limits the scope of their approach to at most 3-class and limited per-point attribute characterization.

In this section, we propose a simple Gabor transform-based pipeline that extracts local and non-local features. Our method applies the Gabor transform to an exemplar point pattern and extracts multi-resolution image-space features with a subsequent convolutional filtering step. The filtering step is performed using random filters, which have shown success in reducing the feature space in neural networks [Ustyuzhaninov et al., 2016; Ulyanov et al., 2018]. We then optimize a stochastic point set by using these features as a target for by-example point pattern synthesis. We conduct qualitative and quantitative experiments on a large variety of patterns to demonstrate the applicability of our method for pattern creation and object placement. We further perform a spectral analysis for our filtering pipeline to interpret why random filters work.

Overview

Given an exemplar pattern with M points $\{\mathbf{p}_i\}_{i=1}^{M}$ within domain $[0,1]$, our goal is to perform pattern expansion to a larger point set $\{\mathbf{P}_i\}_{i=1}^{N}$ in the same domain. The first step in this process requires extracting local and non-local correlations from the exemplar. In the second step, these correlations are reproduced on a large canvas with N points using an optimization. Normally, $N = S \times M$, where S is an up-scaling factor. For simplicity, we focus on a square domain with $S = 4$.

4.1.2 Holistic characterization using Gabor and Random Filters

Our pipeline is shown in Fig. 4.2. The first step of our approach is to transform continuous points into a pixelized feature map representation (blue block). Each point can be assigned a class and some optional attributes. Our feature maps encode the associated attributes as well.

We apply the following continuous Gabor function to get feature maps from points:

$$h(\mathbf{g}_j, \mathbf{p}, \nu, \sigma, \gamma) = \sum_i \mathbf{w}_i \cdot e^{-\frac{\|\mathbf{p}_i - \mathbf{g}_j\|_2^2}{2\sigma^2}} \cos(\nu \cdot (\mathbf{p}_i - \mathbf{g}_j))^{\gamma} \tag{4.1}$$

where $\mathbf{p}_i$ are continuous point positions of a pattern and $\mathbf{g}_j$ are the grid positions of the underlying Gabor feature maps (normalized to $[0,1]$). $\mathbf{w}_i$ is a vector for each point representing additional attributes such as scale, depth, radii. Points with no attributes and only positions make $\mathbf{w}_i = 1$ a scalar. When each point has an additional attribute(s) such as radii, $\mathbf{w}_i$ vector represents these attribute values, which are normalized to $[0,1]$. ν represents the frequency. The $\gamma = \{0,1\}$ parameter can be either 0 or 1 in Eq. 4.1, and it helps capture both the spatial and spectral features simultaneously. For $\gamma = 0$, the Gabor kernel only captures the spatial statistics of the point pattern. This dials down our model to the irregular convolution proposed by Tu et al. [2019]. When $\gamma = 1$, it also captures the spectral characteristics of the pattern.

By modeling both the spatial and spectral statistics, our method synthesizes patterns with better local structure preservation.

We create a Gabor filter bank by varying the values of ν, σ, and γ. These filters help capture both spatial and spectral features of a point set.

Multi-resolution Convolutional Filtering

In the second step, we extract multi-scale, hierarchical features from the Gabor feature maps using multi-resolution convolutional filtering (orange blocks) in Fig. 4.2). All convolutional filters are 7×7 pixels wide. For the first layer, we use a stride 1 convolution. For the remaining layers, we use a stride 2, which down-samples the feature maps by $2\times$. This makes the filtering process multi-resolution.

All convolutional filters are drawn from a uniform distribution following Glorot and Bengio [2010]. More specifically, our model consists of 4 filtering layers. Each layer uses a combination of convolution, instance normalization [Ulyanov et al., 2016; Yi et al., 2018], and ReLU activation function. For the instance normalization layer, we use the following equation without any trainable parameters:

$$\widetilde{q} = \frac{q - E[q]}{\sqrt{Var[q] + \epsilon}} \tag{4.2}$$

where q is the feature map values after convolution and $\widetilde{q}$ is the output after normalization, ϵ = 1e-5. The mean ($E[q]$) and standard deviation ($\sqrt{Var[q]}$) are calculated per-dimension of the feature map. Note that our proposed model is data-independent, meaning that our random filters do not need to be trained on a specific data set. The randomly initialized filters are used directly as network weights for feature extraction. In Sec. 4.1.4, we perform an additional spectral analysis to understand why these convolutional filters work.

Loss Function

After the convolutional layers, we compute the correlation matrix (Eq. (2.8)) and the Gram matrix (Eq. (2.6)) from the resulting features. The corresponding matrices are used to compute our final objective/loss function that we use for synthesis using optimization. The final loss function is a weighted combination of both losses (Sec. 2.1.3):

$$L_{total} = \sum_{l=4} L_{corr}^{l} + \alpha \sum_{l=2,3,4} L_{gram} \tag{4.3}$$

where α = 0.08. We use the $4th$ layer of the network output for computing L_{corr} and layers 2 to 4 for L_{gram}. It is worth mentioning that computing L_{corr} is more expensive than computing L_{gram}. The computational cost depends on n^l, H^l, W^l at the same time. We experimentally find that using n^l = 120 and the 4-th layer's output ($4\times$ down-sampling of input resolution) for computing L_{corr} works best wrt both the optimization time and quality.

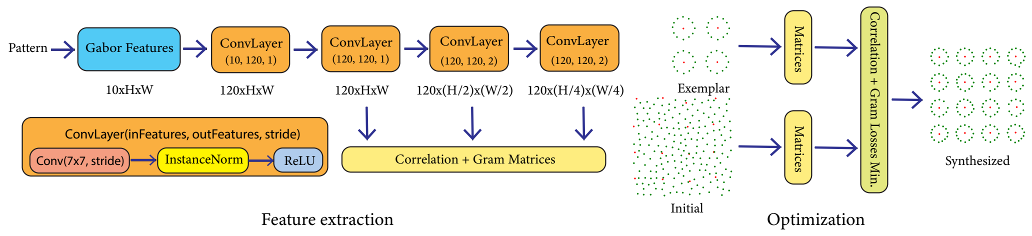


Figure 4.2: *Overview of our pipeline. We propose to use a continuous Gabor transform combined with a multi-resolution convolutional filtering step to compute Correlation and Gram statistics. Since all components are differentiable, we perform end-to-end optimization to update the output pattern by minimizing the Correlation and Gram losses.*

4.1.3 Synthesis

The final step of our pipeline is synthesis using optimization. With our proposed differentiable pipeline, we use a gradient descent-based optimization to update output point locations by minimizing the loss functions. Output point pattern is initialized as a Poisson disk distribution, the same as Tu et al. [2019], within the $[0,1]$ domain. Note that our model also works for random initialization (see appendix Fig. A.3). During optimization, we update point positions by minimizing the proposed loss functions. Since the Gabor transform and convolutional filters are fully differentiable, our optimization process runs end-to-end. Unlike previous methods [Roveri et al., 2015; Tu et al., 2019], we do not apply any heuristics such as removing or re-adding points that might produce distorted local structures. More specifically, we apply the multi-scale optimization procedure [Roveri et al., 2015; Tu et al., 2019] from high σ values to small. Our pipeline naturally extends to handle multi-class and multi-attribute point patterns.

4.1.4 Experiments

We implement our full pipeline using PyTorch [Paszke et al., 2017] and run experiments on an NVIDIA Quadro RTX 8000 graphics card. Additional comparisons and ablation studies can also be found in the appendix.

Gabor features. We compute Gabor features in a multi-scale manner by evenly sampling four σ values in the range $[\sigma_1, \sigma_2]$. The value of σ is analogous to a receptive field; a higher σ value captures non-local structure, while a lower σ value focuses more on local structures. $\sigma_1 = d/c_1$ and $\sigma_2 = d/c_2$, where c_1, c_2 are user-defined and d is the mean minimum distance between any two points within a pattern. We set c_1 to be 0.8 ± 0.2 while c_2 is 2.8 ± 0.2 for all results shown in this work (more details in the appendix Table A.1).

For each scale σ_s, we create 1 spatial map ($\gamma = 0$) and additional 9 frequency feature maps ($\gamma = 1$) from a pattern. The impact of different σ and ν is shown in Fig. 4.3. We consider 3×3 grid frequencies ν in the range $\{-1.5, -1.5\} \rightarrow \{1.5, 1.5\}$. For each frequency ν, we compute the Gabor function from Eq. 4.1. For each pattern, we set the resolution of Gabor feature maps based on the underlying mean average distance d. The resolution of each axis is set to be $round(9/d)$, which is then clamped between $[128, 256]$. The resolution of the output Gabor feature maps is scaled by $\sqrt{S}$, which is the pattern expansion scaling

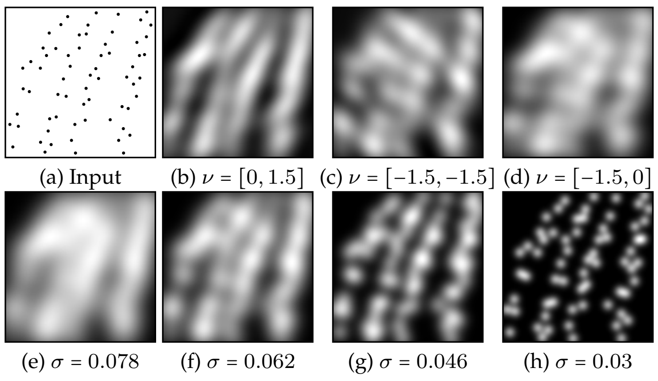


Figure 4.3: *Examples of generated Gabor features from a point pattern (a). (b)-(d) show Gabor features with varying frequencies, the same $\sigma = 0.078$ and $\gamma = 1$. (e)-(h) show Gabor features with varying σ values and $\gamma = 0$ (without considering frequencies).*

factor ($S = 2$). In practice, we use k nearest neighbors of each point $\mathbf{p}$ to compute Gabor features for a given grid position, with $k = min(M, 40)$.

We use the Adam [Kingma and Ba, 2014] optimizer for gradient descent with learning rates $lr = 0.005, 0.002, 0.002, 0.002$ for different 4 scales. Note that using other optimizers, such as L-BFGS, can yield similar results with other components in our pipeline fixed. Fig. 4.4 shows that the optimization starts from a Poisson disk distribution. At each scale, we optimize point locations within the domain $[0, 1]$ until convergence. The convergence criterion is that the decrease of loss during the most recent 100 iterations is smaller than 1%. The whole optimization at all 4 scales takes around $400 \sim 800$ iterations.

Qualitative Comparisons

We compare our point pattern synthesis approach with Ma et al. [2011], Roveri et al. [2015], Tu et al. [2019] and Reddy et al. [2020]. We set the input and output domains to be the same for all comparisons.

Ma et al. [2011] uses random patch copy for pattern initialization. They use a Hungarian matching algorithm to update the sample locations in a patch-based manner. For a fair comparison, we set the neighborhood size in Ma et al. [2011] to ensure the output points are $S = 4$ times the input points. Roveri et al. [2015] also uses random-patch copy initialization for the output pattern. Sample locations are then updated using multi-scale optimization with sampling control by removing and re-adding samples for better convergence.

Tu et al. [2019] (PPS) uses an irregular convolution layer, a pre-trained VGG-19 [Simonyan and Zisserman, 2014] network, and a hierarchical optimization scheme to synthesize point patterns from Poisson disk initialization. We use the same hyperparameters c_1, c_2, and also initialize the output point patterns with Poisson disk for a fair comparison.

Single-class point patterns. Figure 4.5 shows comparisons of regular and irregular structures. Neighborhood-based methods [Ma et al., 2011; Roveri et al., 2015] fail to synthesize the global structure accurately. PPS [Tu et al., 2019] works well compared to other prior methods [Ma et al., 2011; Roveri et al., 2015] but suffers from visible artifacts

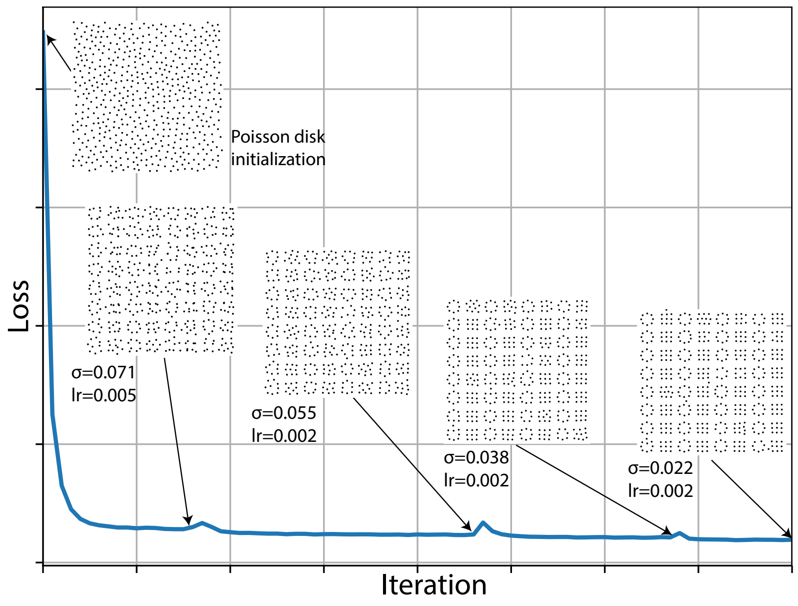


Figure 4.4: *Initial and intermediate results after being optimized at each scale during the multi-scale optimization process.*

like holes or missing points (column b). PPS also does not capture well the regular structures in columns (a) and (c). This may happen due to the fact that VGG-based filters are redundantly learned from the images' dataset and are not expressive enough to deal with point patterns. Our method handles well the regular structures and avoids any artifacts in stochastic distributions (Fig. 4.5b). Please see Appendix A for more comparisons with different point patterns.

Multi-class point patterns. Our method naturally extends to multi-class point pattern synthesis. We sequentially optimize for each class, similar to Ecormier-Nocca et al. [2019]. While synthesizing points of k-th class, we freeze the positions of previous $k - 1$ synthesized classes and compute the Gabor and convolutional feature maps from all these k-class points. Unlike PPS [Tu et al., 2019], which uses one-hot representation for class IDs, our approach does not require additional dimensionality for class IDs. In Figure 4.6, we compare our multi-class synthesis approach with PPS. For a fair comparison, we incorporate our sequential multi-class approach within PPS to better handle multi-class patterns. This is shown as PPS++ in Fig. 4.6. This allows PPS to work with more than 3 classes. The bottom row shows object placement with different objects assigned to different classes. Figure 4.6a shows an example that is also presented in Fig. 4.1.

Multi-attribute point patterns. Our approach can easily incorporate multiple attributes into the pipeline. We assign values (normalized to [0, 1]) to attributes $\mathbf{w}_i$ in Eq. 4.1 for each point while computing the Gabor features.

As a result, for $|\mathbf{w}|$ attributes, we will have $|\mathbf{w}|$ Gabor feature maps concatenated together. Similarly, the number of filters in the convolutional layers would increase by a factor of $|\mathbf{w}|$, resulting in $|\mathbf{w}|$-channel images going as input to the convolutional filters (see Fig. 4.2).

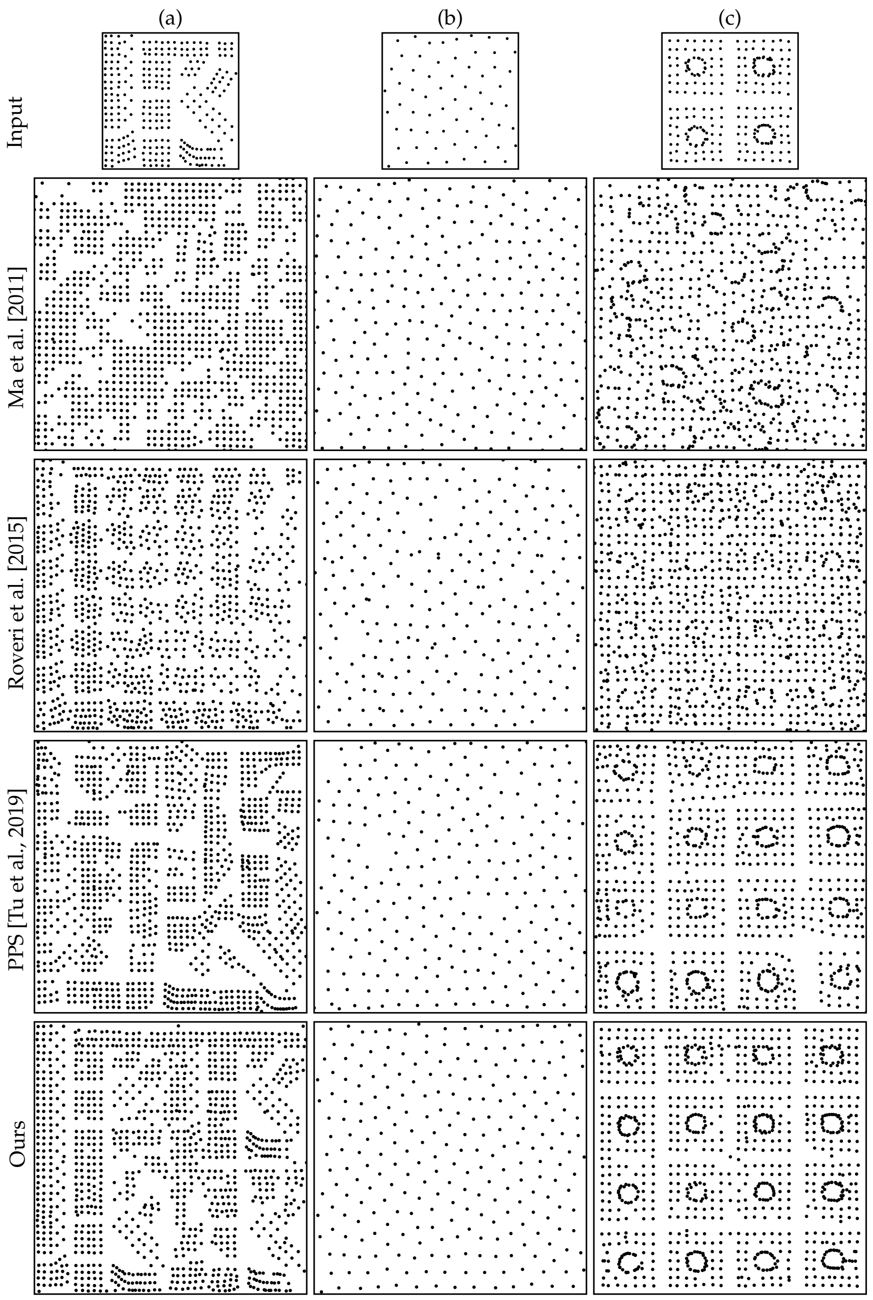


Figure 4.5: *Single-class point-pattern expansion comparison between our method and previous methods. Our approach preserves the local and non-local structures better than previous methods.*

In Figure A.10, we show a comparison of 4-class patterns with attributes (depth and scale) assigned to each point. DiffComp [Reddy et al., 2020] works well for patterns with stochastic structures (column a, c), but it fails to capture the vertical structure in column (b). DiffComp also suffers during pattern expansion with more empty spacing, as shown in column (c). PPS [Tu et al., 2019], on the other hand, works slightly better for structured input, like column (b). However, it still shows some holes in all three

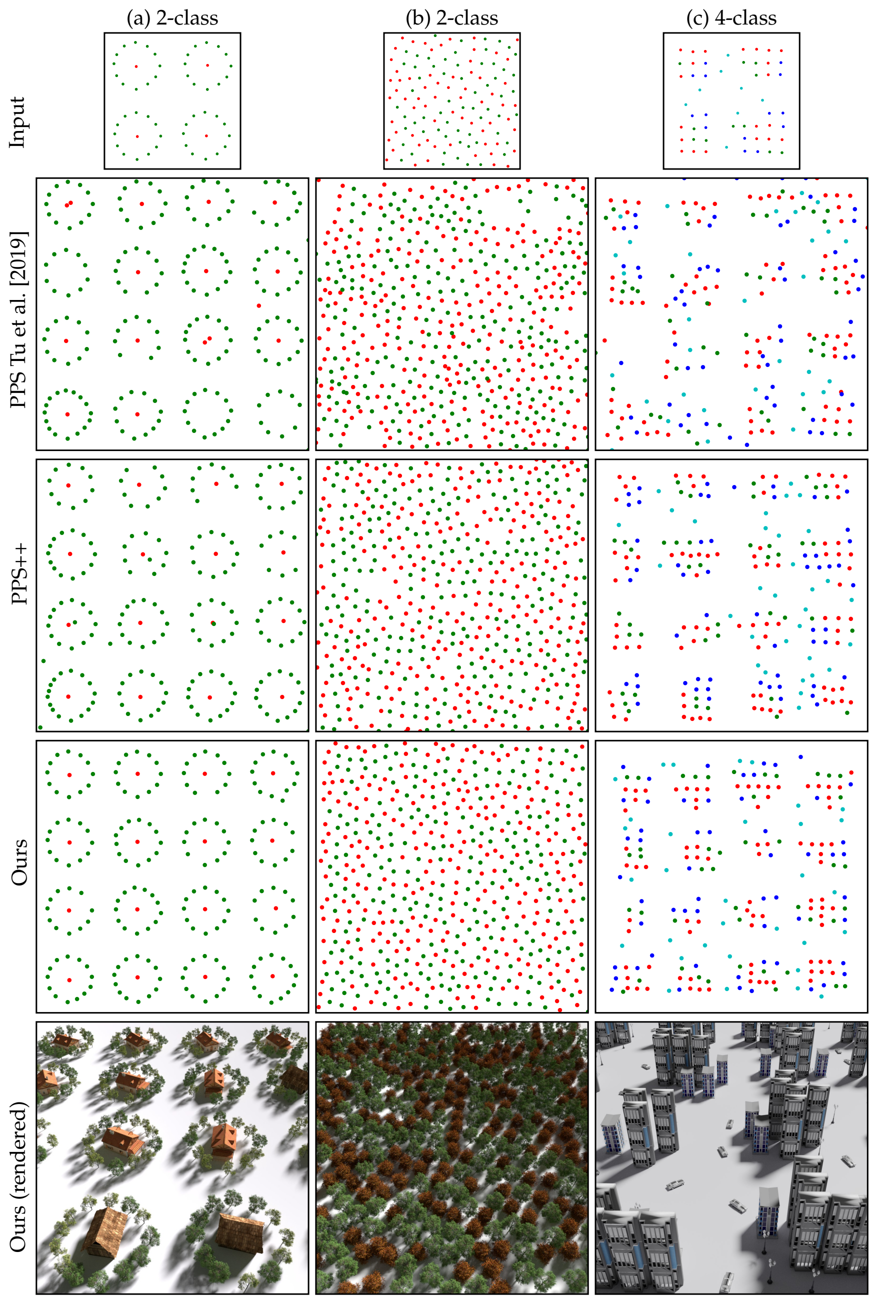


Figure 4.6: *Multi-class point-pattern expansion. Our method performs better in terms of global structure and local distances between point samples. PPS [Tu et al., 2019] does not handle well point patterns with multiple classes. We enhance PPS with our sequential multi-class synthesis approach, but it still shows artifacts. The bottom row shows rendered results with object placement at point locations.*

output patterns. Although we do not explicitly encode element shapes as attributes, our synthesized patterns are visually closer to the exemplars.

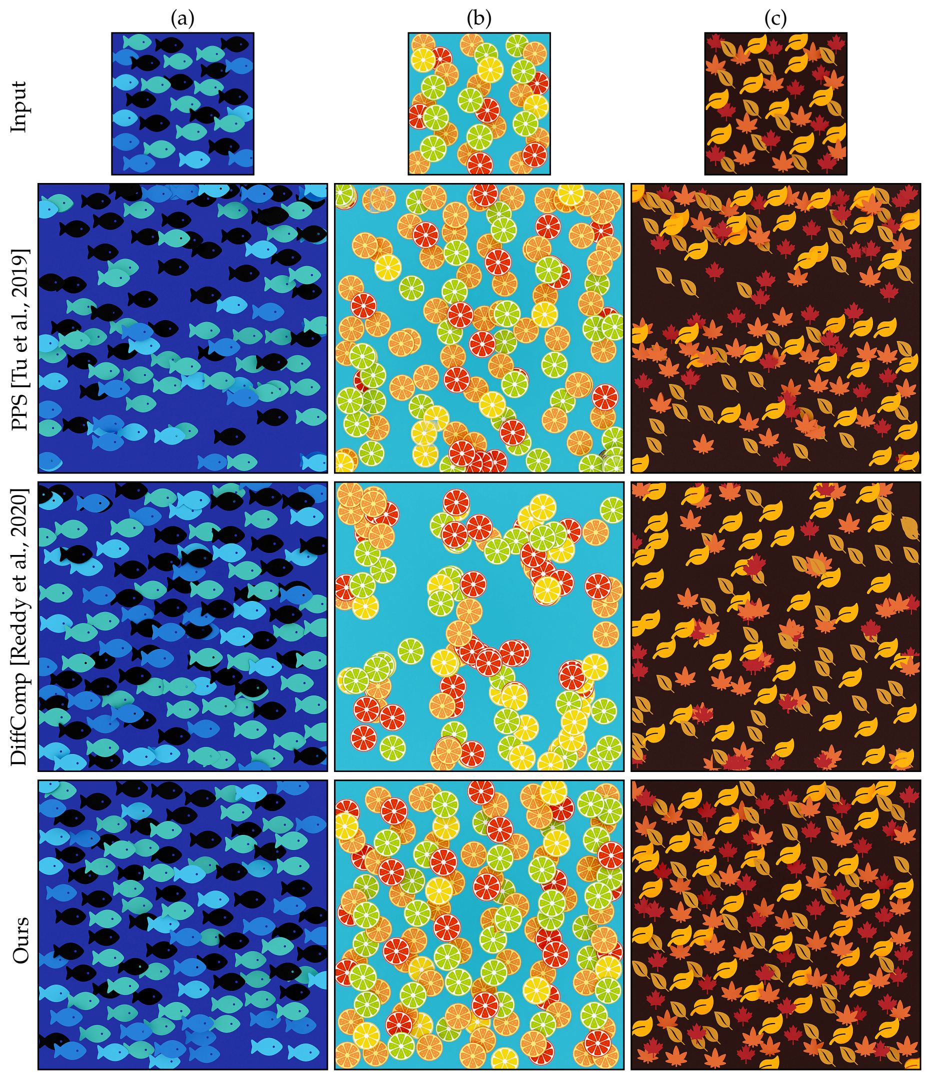


Figure 4.7: *Discrete element-pattern expansion. Our method naturally supports point-based element patterns. We treat each color as a separate class. Each column here is a 4-class pattern with two attributes (scale and depth) per element.*

As DiffComp [Reddy et al., 2020] only uses Gram loss for pattern expansion, we study two more variants of their method using L_{corr} and the weighted combination of L_{gram} and L_{corr} as shown in appendix Fig. A.7. We also compare the same methods on 2- and 3-class element patterns in appendix Fig. A.6 and show more results for our method on 2- and 6-class, 2- and 5-attribute element patterns in appendix Fig. A.10.

In Figure 4.8, we show disk distributions where points are assigned a radius as an attribute. We compare our method against PPS [Tu et al., 2019] and Ecormier-Nocca et al. [2019] (a method based on pair correlation function) on a single- and two-class disk distribution. Our method handles the regular structures in the first row and achieves comparable results in the second row.

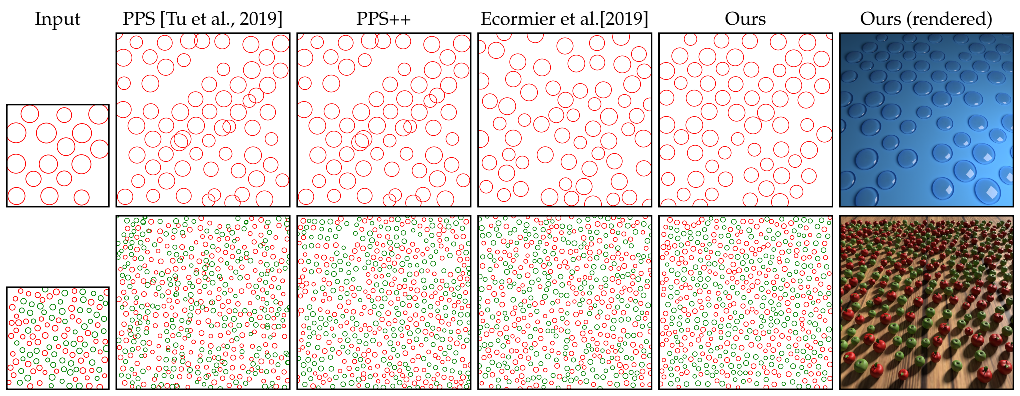


Figure 4.8: *Comparison of our method with PPS [Tu et al., 2019], PPS++ (our enhanced variant of PPS), and Ecormier-Nocca et al. [2019] on disk and multi-class disk distributions. Our method better preserves the overall structure for both regular and irregular structures. Meanwhile, we achieve comparable results with Ecormier-Nocca et al. [2019] on the second row, in terms of preserving non-overlapping disks.*

Table 4.1: *Quantitative comparison of relative radius [Lagae and Dutré, 2008] on a CCVT [Balzer et al., 2009] blue noise pattern as shown in Fig. 4.5 (b). The closer to the exemplar, the better.*

	Exemplar	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
ρ	**0.7341**	0.6619	0.6507	0.6655	**0.7064**

Quantitative Evaluation

To the best of our knowledge, there is no established metric to quantitatively evaluate structured point patterns. Therefore, we perform several experiments, including a user study, providing users with side-by-side comparisons between our method and prior methods.

Quantitative metrics. A quantitative metric called relative radius [Lagae and Dutré, 2008] is well-defined for blue noise patterns. We choose the CCVT [Balzer et al., 2009] profile as the exemplar and apply 4 methods in Fig. 4.5 (b) for synthesis. Relative radius is defined as $\rho = r_{max}/r_{min}$. As shown in Table C.7, the relative radius with our approach is the closest to the exemplar. This means we better preserve the equi-distribution property compared to other methods. Visual inspection consistently supports this behavior as shown in Fig. 4.5(b).

We also include Wasserstein distance, Chamfer distance, and the pair correlation function (PCF) as additional metrics for quantitative comparison. The Pair Correlation Function (PCF) is also widely used for characterizing stationary point patterns. As our goal is to synthesize output patterns on a larger canvas, direct comparison of these three metrics is not reasonable. We, therefore, split the output pattern into 4 non-overlapping patches. We then compute the PCF of each patch and compare it against the PCF of the exemplar. For PCF, we compute the mean-square error (MSE) between the output patch PCF and the exemplar's PCF. The error is averaged across the 4 patches. A similar operation is performed for the Wasserstein and Chamfer distance metrics. Table 4.2 shows the values for all these metrics for point patterns in Fig. 4.5. Additional quantitative analysis for

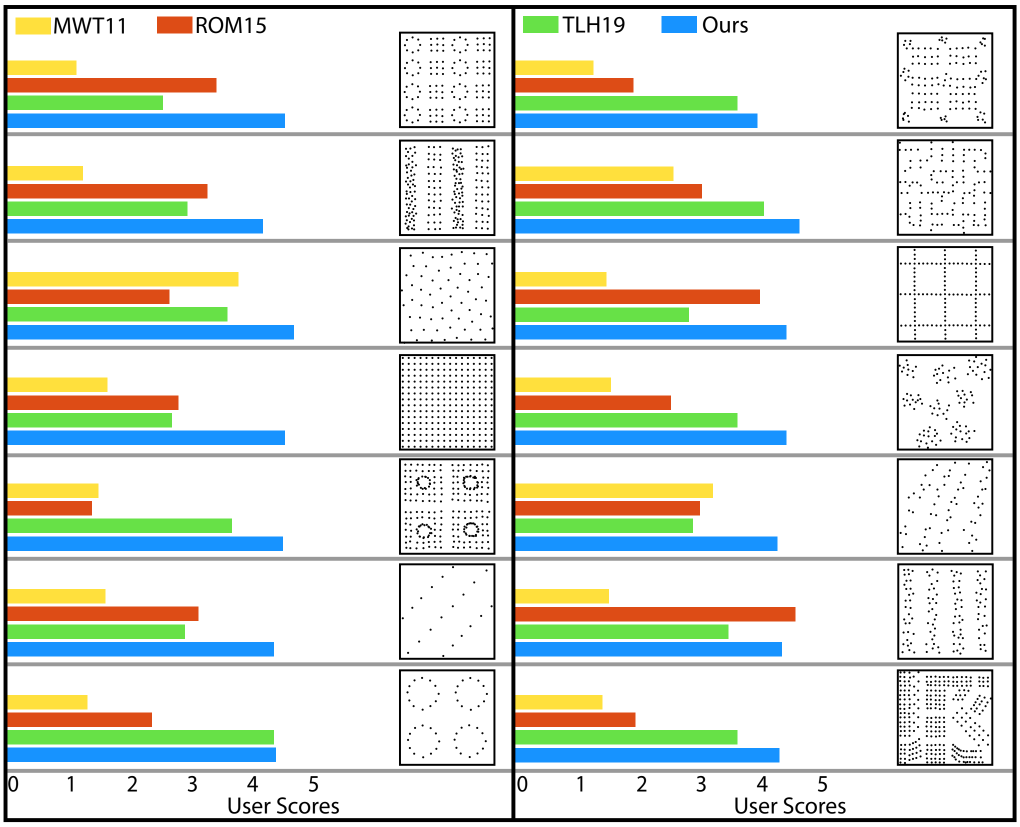


Figure 4.9: *We perform a user study and compute an average score across 28 users. We show different point patterns and ask users to score between 1 (worst) and 5 (best). Ours (in blue) gets better scores for all but one pattern. All patterns and their comparisons can be found in the appendix.*

more patterns can be found in appendix Table A.3.

User study. We perform a user study to analyze the visual differences between our and previous methods. In total, 28 users participated in this experiment. Users were shown the input exemplars and the shuffled outputs from four methods: Ours, Ma et al. [2011], Roveri et al. [2015], and Tu et al. [2019]. We use 14 different point patterns for the study. Three of the patterns are shown in Fig. 4.5. The other patterns can be found in the appendix Fig. A.8, Fig. A.9. Users are asked to give a score from 1 (worst) to 5 (best) for each of the output patterns based on their visual preferences over the local and global structure preservation. Figure 4.9 shows the final averaged score across all users for each pattern. Our method achieves the highest scores on average for all scenes except one. Detailed results and numbers about the user study can be found in the appendix Table A.2.

Table 4.2: *Quantitative comparisons for single-class point pattern synthesis results of previous methods and ours. More quantitative results can be found in appendix Table A.3.*

Scene	MSE of PCFs (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 4.5, (a)	0.2678	0.2836	0.2609	**0.2459**
Fig. 4.5, (b)	0.3296	0.3286	0.3103	**0.3072**
Fig. 4.5, (c)	0.6346	0.4913	0.5436	**0.3911**

Scene	Wasserstein Distance (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 4.5, (a)	1.4483	1.1148	1.0222	**0.6896**
Fig. 4.5, (b)	0.4858	0.5166	0.4885	**0.3352**
Fig. 4.5, (c)	2.1869	2.2596	2.2570	**0.5287**

Scene	Chamfer Distance (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 4.5, (a)	0.0571	0.0440	0.0517	**0.0353**
Fig. 4.5, (b)	0.0962	0.0869	0.0920	**0.0429**
Fig. 4.5, (c)	0.0662	0.0685	0.0719	**0.0184**

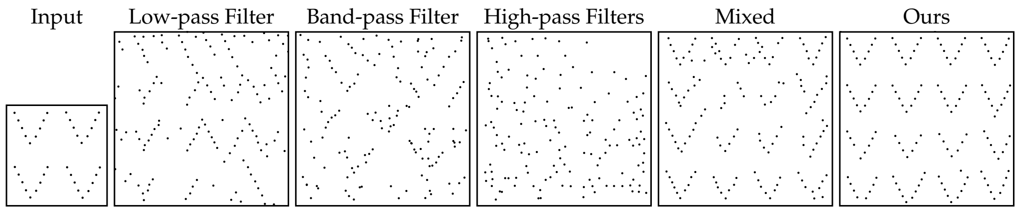


Figure 4.10: *Analysis of using different filters in our filtering pipeline. Using Low-pass (Gaussian), band-pass filters (Sobel), and high-pass filters (derivative) fails to preserve complicated structures on the output pattern. While a simple combination of low-, band-, and high-pass filters starts giving correct orientation and overall structures.*

Analyzing Random Filters

Random filters have shown impressive improvements [Ulyanov et al., 2018; Ustyuzhaninov et al., 2016] for feature extraction. We perform spectral analysis (in Fig. 4.10) to understand the impact of random filters with different frequency spans on the synthesis. For this experiment, we use a simplified filter bank during optimization: 4 Gaussian filters (low-pass) with evenly sampled sigma, 4 Sobel operators as band-pass, and 4 derivative filters as high-pass.

As demonstrated in Fig. 4.10, when using only low-pass, high-pass, or band-pass filters during optimization, we found the resulting synthesis quality to be bad. However, using all filters together starts bringing the quality of synthesis a bit closer to ours (with random filters). This demonstrates that a combined filter bank works better as it contains filters spanning a larger range of frequencies (from low to high). In theory, random filters span all frequencies based on their Fourier power spectrum. This explains why using random filters can generate even better results from the perspective of Fourier analysis. VGG-based filters trained on large-scale image data, however, are not guaranteed to span all frequencies. This is consistent with our point pattern synthesis results, where random filters outperform VGG-based filters.

Ablation Study

We first evaluate the design choices of our pipeline using Gabor and random filters. As shown in Fig. 4.11, using only spatial features by irregular convolution [Tu et al., 2019] results in degraded local and global features with overlaps and outliers. Using Gabor features better preserves local and global structures compared with the input. Further, as illustrated in Fig. 4.12, using only Gabor filters without random filters does not capture the details (column (b)). Using only random filters means we remove the Gabor filters by pixelating the input points. This results in incorrect and broken shapes (column (c)). Our combination of both filters better preserves local and global structures compared with the input.

Also, we study the impact of normalization layers and non-linear activation functions. Fig. 4.13 demonstrates that our method would completely fail without using normalization. Without a non-linear activation function such as ReLU, the synthesized patterns, such as V-structure, are much noisier, and the global structure is not preserved well.

Loss functions. We study how different losses affect the synthesis quality. As shown in Fig. 4.14, deep correlation loss captures better the globally regular structure, while gram loss captures local structures. Without L_{corr}, the overall regularity is not preserved. Without L_{gram}, the disks start to have overlaps.

More recently, Heitz et al. [2021] presents a Sliced Wasserstein loss for image-based texture synthesis, achieving better results than using Gram loss. We have experimented with Sliced Wasserstein loss, noted as L_{sw}, as a replacement of Gram loss. Figure 4.14 illustrates that combining L_{sw} with L_{corr} introduces more distortion of the local structure than ours (L_{gram} + L_{corr}).

More ablation studies about the number of random filters, the chosen layer for L_{corr} computation, and loss functions can be found in Appendix A.3.1.

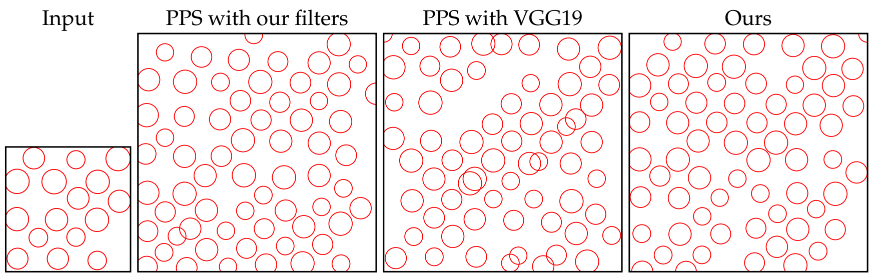


Figure 4.11: *Ablation study on our proposed Gabor features and filters, compared with the irregular convolutional features and VGG19 network in PPS [Tu et al., 2019]. Our filters better capture the overall structure, and Gabor features better preserve the local structure.*

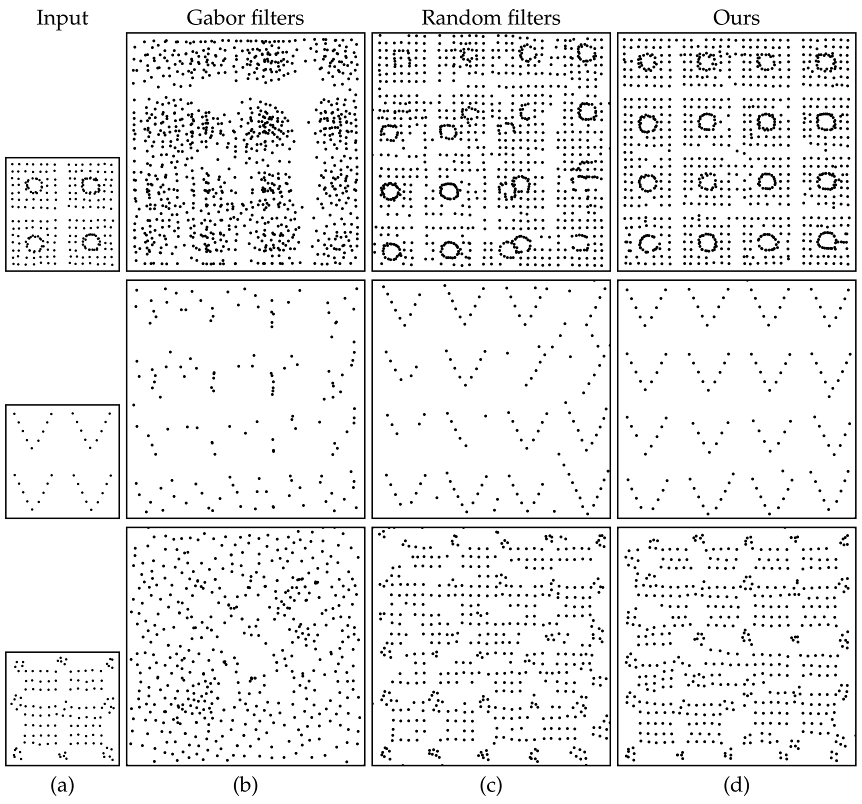


Figure 4.12: *Ablation study on our proposed Gabor and random filters. Our combination of Gabor and random filters better captures the overall structure than using only Gabor or random filters.*

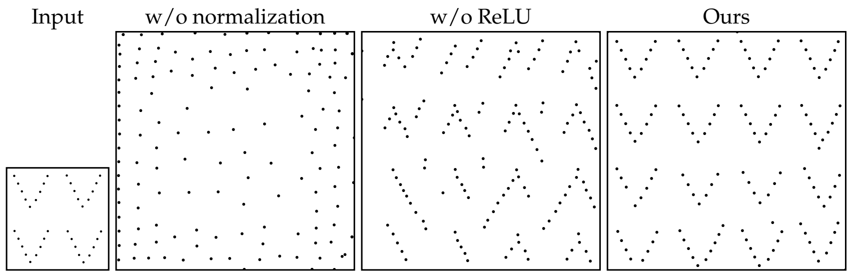


Figure 4.13: *Ablation study on component choices in our convolutional filtering pipeline. We compare our final design with the one without instance normalization or ReLU activation. This shows both normalization and ReLU are important for high-quality synthesis.*

Performance

With our implementation, synthesizing a point pattern from an exemplar takes from 2 to 15 minutes, where the number of output samples varies from 64 to 1124. We show the run-time of PPS [Tu et al., 2019] and our method using an NVIDIA Quadro RTX 8000 on various exemplars in Table 4.3. Our method is up to 8 times faster than theirs. Also, the total number of parameters of our filters is about $2.4M$, while the number of parameters

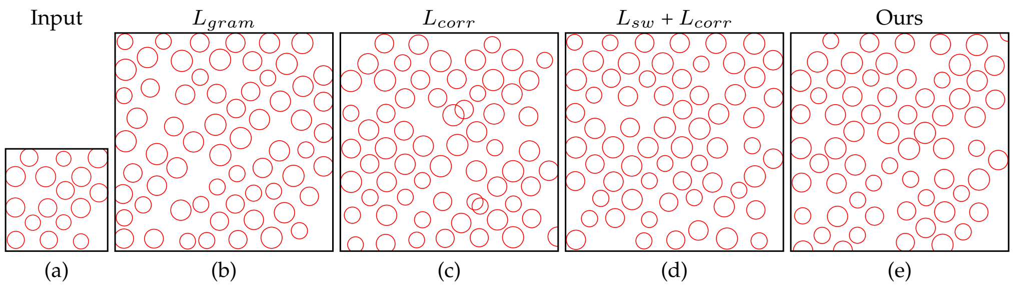


Figure 4.14: *Ablation study on losses. With only L_{gram}, the output patterns do not preserve the global structure. Though using only L_{corr} preserves the globally regular structure, the output patterns fail to preserve local details such as the distances between points and disks. L_{sw} combined with L_{corr} introduces more distortion than ours (column (d), bottom region).*

Table 4.3: *We summarize statistics of different exemplars and the runtime of our method and PPS [Tu et al., 2019]. Note that for the results of multi-class patterns in Fig. 4.6, Fig. A.10, and Fig. 4.8, we compare against the runtime of PPS++ variant for fair comparisons. R refers row number of the corresponding figure.*

Scene	#Classes	#Output Samples	#Attributes	Runtime (m)	
				PPS	Ours
Fig. 4.5, (a)	1	1124	-	23.9	5.7
Fig. 4.5, (b)	1	256	-	11.0	3.8
Fig. 4.5, (c)	1	992	-	22.7	5.6
Fig. 4.6, (a)	2	208	-	18.1	4.4
Fig. 4.6, (b)	2	512	-	58.1	7.4
Fig. 4.6, (c)	4	216	-	44.7	9.7
Fig. A.10, (a)	4	112	2	23.8	9.3
Fig. A.10, (b)	4	124	2	26.7	11
Fig. A.10, (c)	4	160	2	31.7	14.5
Fig. 4.8, R1	1	64	1	5.6	2.3
Fig. 4.8, R2	2	512	1	58.1	7.4

of the VGG-19 feature extractor they used is about $10.6M$, about 5 times more than ours.

4.1.5 Conclusion

We present a Gabor transform-based framework for point pattern synthesis. This allows capturing both spatial and spectral features simultaneously. Unlike previous point pattern synthesis approaches, our method naturally extends to multi-class and multi-attribute point pattern synthesis. We simplify the feature space to Gabor feature maps and random filters' features. We also analyze why random filters work better than well-trained VGG-19-based features. Our analysis hints that since random filters span all frequencies, it helps better capture the details previously missed by VGG-19-based features.

Limitations. Feature map representation [Roveri et al., 2015; Tu et al., 2019] allows us to extract positional and structural information from points, which makes it possible for our method to handle patterns with anisotropic, regular, and irregular structure.

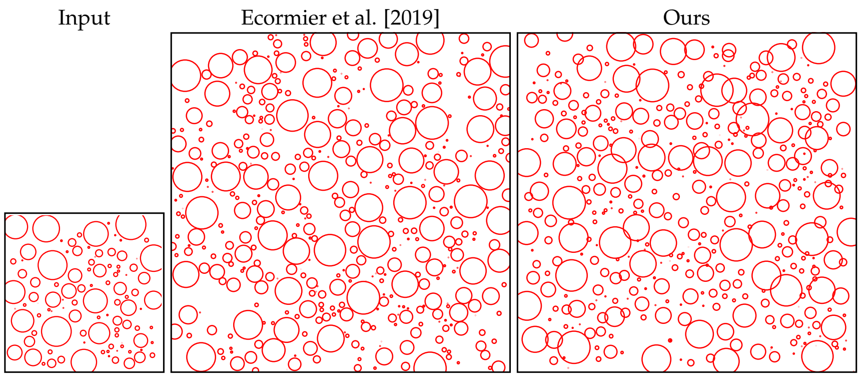


Figure 4.15: *A failure case for highly constrained disk distribution. Since we do not explicitly handle the distance between disks of varying radii, it is hard for our method to synthesize this distribution without overlaps, although the distributions still seem relevant.*

However, similar to previous methods [Ma et al., 2011; Roveri et al., 2015; Tu et al., 2019], we share the same issue that it requires the user to input the window size or kernel size (e.g., σ in Eq. (4.1)) for optimization. These hyperparameters can affect the quality of the synthesis as shown in Tu et al. [2019]. In the multi-class setting, we need to fully run the optimization for each class. Consequently, the time complexity of our optimization grows linearly with the number of classes. This can be problematic for applications where interactive feedback is critical.

Future work. Currently, our feature maps are pixelized. It would be interesting to encode the features in a continuous space. This could potentially avoid artifacts due to the pixelized nature of the feature maps. Though our characterization and synthesis pipeline is end-to-end, with no human intervention, the results are still not perfect according to the user evaluation. Average score for each pattern using our method is from around 4 to 4.5, while 5 is the maximum. One direction for further improvements on local structure can be using an interactive authoring system as shown in Tu et al. [2020]. This would allow users to fix remaining synthesis issues in local regions interactively. Currently, we focus on a unified pipeline for point pattern synthesis. However, as shown in Fig. 4.15, our method might fail to capture disk distributions where disks can have varying radii and are strictly non-overlapping. The next step should be to consider extending our method to explicitly control the distance between disks or even to handle shape-aware elements. Besides, we are interested in extending our point-based framework to related applications such as continuous curve synthesis [Tu et al., 2020], distribution infilling and clustering [Nicolet et al., 2020].

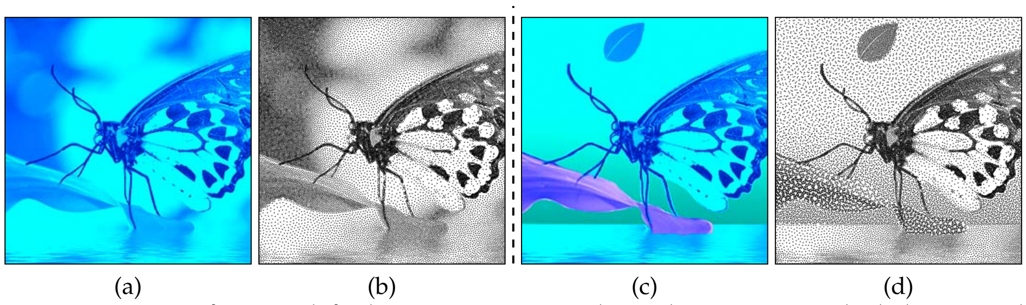


Figure 4.16: *Our framework facilitates point pattern design by representing both density and correlation as a three-channel raster image (a). These images can be edited (c) in terms of their density or correlation using off-the-shelf image manipulation software. The resulting point patterns are shown before (b) and after the edits (d). Please see the accompanying supplemental HTML for vector graphic images.*

4.2 Noise correlation embedding for point pattern editing

We have introduced our method on example-based point pattern synthesis using training-free correlations. However, one limitation is that our method does not provide local, explicit, and decoupled control on point density and correlation with the proposed holistic training-free correlations. In this section, we introduce a new framework, called Patternshop [Huang et al., 2023], providing explicit, decoupled control over point pattern density and correlation.

Point patterns are characterized by their density and correlation. While spatial variation of density is well-understood, analysis and synthesis of spatially-varying correlation is an open challenge. No tools are available to intuitively edit such point patterns, primarily due to the lack of a compact representation for spatially varying correlation. We propose a low-dimensional perceptual embedding for point correlations. This embedding can map point patterns to common three-channel raster images, enabling manipulation with off-the-shelf image editing software. To synthesize back point patterns, we propose a novel edge-aware objective that carefully handles sharp variations in density and correlation. The resulting framework allows intuitive and backward-compatible manipulation of point patterns, such as recoloring, relighting, to even texture synthesis that have not been available to 2D point pattern design before. The effectiveness of our approach is tested in several user experiments.

4.2.1 Introduction

Point patterns are characterized by their underlying density and correlations. Both properties can vary over space. As shown in Fig. 4.16, density is encoded via the gray-scale color while point correlations are encoded as the bluish and purplish colors. However, two key challenges limit the use of such patterns: first, a reliable representation, and second, the tools to manipulate this representation.

While algorithms [Zhou et al., 2012; Öztireli and Gross, 2012] are proposed in the literature to generate point patterns with specific correlations (e.g., blue-, green-, pink-, etc. noise), designing specific correlation requires understanding of the power spectrum or PCF [Heck et al., 2013] only a handful of expert users might have. Further, the space spanned by these *colored* noises (correlations) is also limited to a handful of noises studied in the literature [Zhou et al., 2012; Öztireli and Gross, 2012]. Addressing this, we embed point correlations in a 2D space in a perceptually optimal way. In that space, two-point correlations have similar 2D coordinates if they are perceptually similar. We simply sample all realizable point correlations and perform MDS on a perceptual metric, without the need for user experiments, defined on the pairs of all samples. Picking a 2D point in that space simply selects a point correlation. Moving a point changes the correlation with perceptual uniformity.

The next difficulty in adopting point patterns of spatially varying density and correlations is the lack of tools for their intuitive manipulation. Modern creative software gives unprecedented artistic freedom to change images by relighting [Pellacini, 2010], painting [Strassmann, 1986; Hertzmann et al., 2001], material manipulation [Pellacini and Lawrence, 2007; Di Renzo et al., 2014], changing canvas shape [Rott Shaham et al., 2019], completing missing parts [Bertalmio et al., 2000], as well as transferring style [Gatys et al., 2016] or textures [Efros and Freeman, 2001; Zhou et al., 2018; Sendik and Cohen-Or, 2017]. Unfortunately, no such convenience exists when aiming to manipulate point patterns, as previous approaches are specifically designed to process three-channel (RGB) raster images. Our core idea is to convert point patterns into an off-the-shelf three-channel raster image. We use the $CIELAB$ color space to encode density as one-dimensional luminance (L) and two-dimensional chroma (AB) as correlation. The resulting images can be manipulated, harnessing all the power of typical raster image editing software.

While spatially-varying density is naturally mapped to single-channel images, this is more difficult for correlation. Hence, it is often assumed spatially-invariant and represented by either the power spectrum [Ulichney, 1987; Lagae and Dutre, 2008] or the PCF [Wei and Wang, 2011; Öztireli and Gross, 2012]. Some work also handles spatially-varying density and/or correlation [Chen et al., 2013; Roveri et al., 2017], but with difficulties in handling sharp variation of density and/or correlation. We revisit bilateral filtering to handle sharp variation both in density and correlation.

Finally, we show how to generate detailed spatial $CIELAB$ maps of correlation and density given an input point pattern using a learning-based system, trained by density and correlation supervision on a specific class of images, e.g., human faces.

In summary, we make the following contributions:

- Two-dimensional perceptual embedding of point correlations,
- Spatially-varying representation of point pattern density and perceptually-embedded correlation as LAB raster images,
- A novel definition of edge-aware correlation applicable to hard edges in density and/or correlation,
- An optimization-based system to synthesize point patterns defined by density and embedded correlation maps according to said edge-aware definition,
- Intuitive editing of point pattern correlation and density by recoloring, painting, relighting, etc., of density and embedded correlation maps in legacy software such as Adobe Photoshop.
- A learning-based system to predict density and embedded correlation maps from an input point pattern.

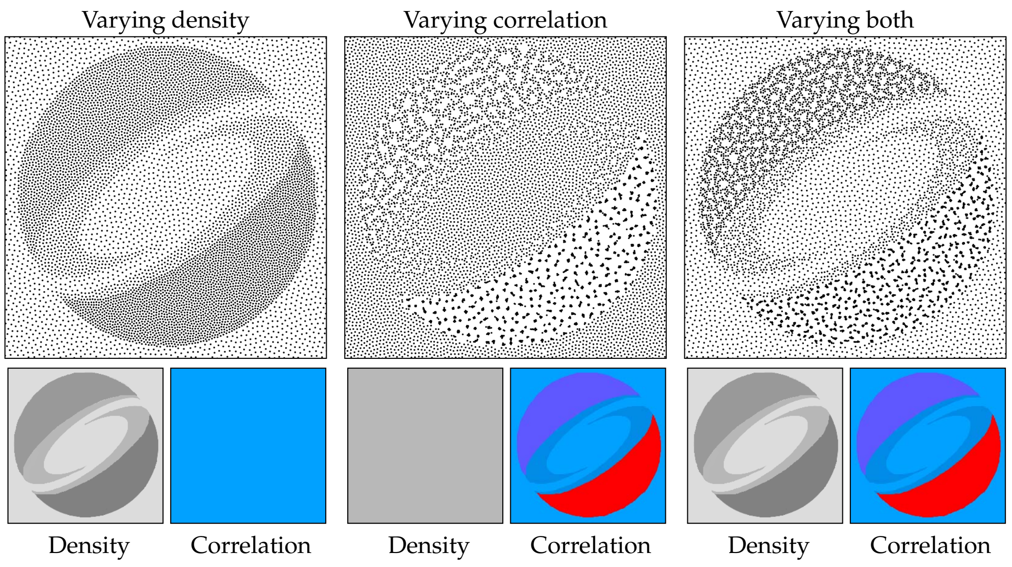


Figure 4.17: *Our framework allows for manipulating density and correlations independently. The top row shows the point set synthesized using the density (left) and the correlation map (right) shown in the bottom row. The first column shows a point set when both density and correlation are spatially varying. The second column has constant correlation but spatially varying density. The third column point set reproduces the Siggraph logo just from spatially varying correlations.*

Overview

Fig. 4.18 shows the workflow of Patternshop: A user selects a point pattern with spatially varying density and correlation and runs this through a neural network (NN) to produce a three-channel raster image. Alternatively, they can draw that image directly, ab-initio, in a program like Adobe Photoshop or GIMP. This raster image can then be edited in any legacy software. Density can be manipulated by editing luminance. To change correlation, we provide a perceptual 2D space of point correlations (Fig. 4.20). This space is perceptually uniform, i.e., distances are proportional to human perception and covers a wide range, i.e., it contains most realizable point correlations. Figure 4.17 demonstrates one such example edit. A user may edit spatially varying density (Fig. 4.17b) or correlation (Fig. 4.17c) or both (Fig. 4.17a) using our three-channel representation. A final optimizer produces the edited point pattern.

Correlations are generally characterized by a PCF or power spectrum. A PCF encodes spatial statistics—e.g., the pairwise distances between points—distributed over an m-bin histogram. Handling such a m-dimensional correlation space is neither intuitive nor supported by existing creative software. To tackle this problem, in Sec. 4.2.2, we embed PCF into a two-dimensional space. We optimize the distance between all pairs of latent coordinates in that space to be proportional to the perceived visual distance between their point pattern realizations.

Moreover, synthesizing such patterns requires support for edge-aware transitions for both density and correlation. In Sec. 4.2.3, we outline our formalism to handle such edge-aware density and correlations. Sec. 4.2.4 defines the optimization objective, which enables finding a point pattern to have the desired density and correlation. We also provide details on the implementation, and outline various application scenarios supported

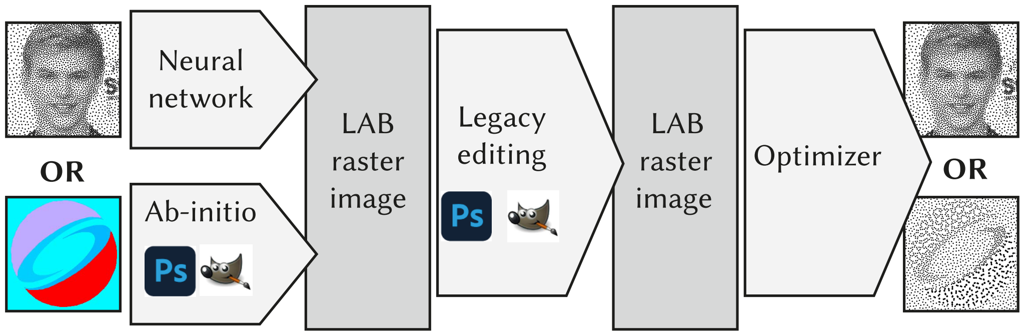


Figure 4.18: *The workflow of Patternshop. A user can select a point pattern and run it through a neural network to produce a three-channel raster image, or draw that image directly, ab-initio, in a program like Adobe Photoshop or GIMP. The edited pattern can be synthesized using our optimization.*

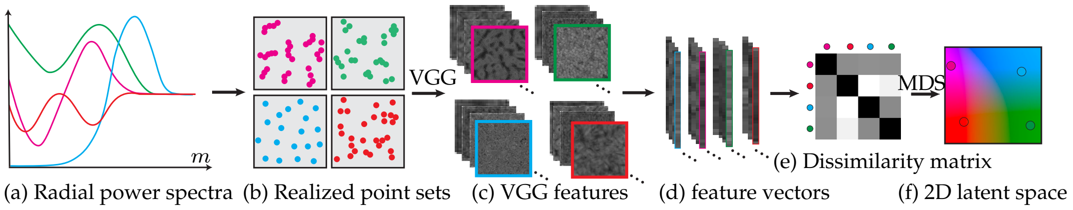


Figure 4.19: *We illustrate the embedding from m-dimensional space to a 2D space. (a) Random power spectra are generated, and their corresponding realizable point patterns in (b) are synthesized using gradient descent. The point patterns are then rasterized and pass through a pre-trained VGG network [Simonyan and Zisserman, 2015] to generate (c) feature maps, which are flattened into feature vectors (d). A dissimilarity matrix (e) computed from these VGG feature vectors is used to bring m-dimensional representation to a 2D space (f) using MDS (Sec. 4.2.2).*

by our pipeline, followed by comparative analyses, concluding discussions, and future works.

4.2.2 Latent embedding of point correlations

Overview. In order to encode the correlation of a point pattern as a 2D coordinate, we first generate a corpus of basic point patterns (each with constant correlation). Next, we extract perceptual features of these patterns. The pairwise distance between all features forms a perceptual metric. This metric is used to embed a discrete set of basis correlations into a 2D space. From this discrete mapping, we can further define a continuous mapping from any continuous correlations into a latent space, as well as back from any continuous latent space coordinate to correlations. This process is a pre-computation, only executed once, and its result can be used to design many point patterns, same as one color space can be used to work with many images. We detail each step in the following:

Data generation. First, we systematically find a variety of point patterns to span a gamut of correlations $\{g_i\}$. We start by defining a power spectrum as a Gaussian mixture of either (randomly) one or two modes with means randomly uniform across the domain and uniform random variance of up to a quarter of the unit domain (Fig. 4.19a). Not all such power spectra are realizable. Therefore, we run a gradient descent optimization [Leimkühler et al., 2019] to obtain realizable point patterns (Fig. 4.19b). We finally use the PCF of that realization P_i as g_i. Please see Appendix B.1.1 for details.

Metric. A perceptual metric $D_{i,j}$ assigns a positive scalar to every pair of stimuli i and j (here: point patterns) which is low only if the pair is perceived as similar. As point patterns are stationary textures, their perception is characterized by the spatially aggregated statistics of their visual features [Portilla and Simoncelli, 2000]. Hence, the perceived distance between any two point patterns with visual features $\mathbf{v}_i$ and $\mathbf{v}_j$ is their L_1 distance $D_{i,j} = |\mathbf{v}_i - \mathbf{v}_i|_1$.

To compute visual feature statistics $\mathbf{v}_i$ for one pattern P_i, we first rasterize P_i three times with point splats of varying Gaussian splat size of 0.015, 0.01, and 0.005 [Tu et al., 2019]. Second, each image of that triplet is converted into VGG feature activation maps [Simoncelli and Olshausen, 2001]. Next, we compute the Gram matrices [Gatys et al., 2016] for the `pool_2` and `pool_4` layers in each of the three images. Finally, we stack the triplet of pairs of Gram matrices into a vector $\mathbf{v}_i$ of size $3 \times (64^2 + 128^2) = 61440$.

Figure 4.19 shows four example patterns leading to a 4×4 dissimilarity matrix.

Dimensionality reduction. To first map the basis set of $\{P_i\}$ to latent 2D codes, we apply MDS (Fig. 4.19f). MDS assigns every pattern P_i a latent coordinate $\mathbf{z}_i$ so that the joint stress of all assignments

$$c_{\mathrm{Emb}}(\{\mathbf{z}_i\}) = \sum_{i,j} (D_{i,j} - ||\mathbf{z}_i - \mathbf{z}_j||)^2 \tag{4.4}$$

is minimized across the set $\{\mathbf{z}_i\}$ of latent codes.

After optimizing with Eq. (4.4), we rotate the latent coordinates so that blue noise is indeed distributed around a bluish color of the chroma plane, and then normalize the 2D coordinates to $[0,1]^2$.

Encoding. Once the latent space is constructed, we can sample correlation g at any continuous coordinate $\mathbf{z}$ in the latent space using the IDW method:

$$f(\mathbf{z}) = \sum_i g_i w_i(\mathbf{z}) / \sum_i w_i(\mathbf{z}) \quad \text{with} \tag{4.5}$$

$$w_i(\mathbf{z}) = 1/\max((||\mathbf{z} - \mathbf{z}_i||_2)^{\phi(\mathbf{z})}, 10^{-10}). \tag{4.6}$$

The idea is to first compute the distance between the current location $\mathbf{z}$ and the existing locations $\mathbf{z}_i$ in the MDS space. The inverse of these distances, raised to a power $\phi(\mathbf{z})$, is used as the weight to interpolate the correlations g_i. $\phi \in \mathbb{R}^2 \mapsto \mathbb{R}$ is a function that is low in areas of the latent space with a low density of examples and high in areas where MDS has embedded many exemplars. We implement ϕ by kernel density estimation on the distribution $\{\mathbf{z}_i\}$ itself with a Parzen window of size 0.01 and clamping and scaling this density linearly to the $(3,6)$-range.

Decoding. We can now also embed any correlation $\bar{g}$, that is not in the discrete basis set $\{g_i\}$. Let $\bar{\mathbf{v}}$ be the visual features stats of $\bar{g}$. We simply look up the visual-feature-nearest training exemplar and return its latent coordinate

$$f^{-1}(g) = \arg\min_{\mathbf{z}_i} ||\bar{\mathbf{v}} - \mathbf{v}_i||_2. \tag{4.7}$$

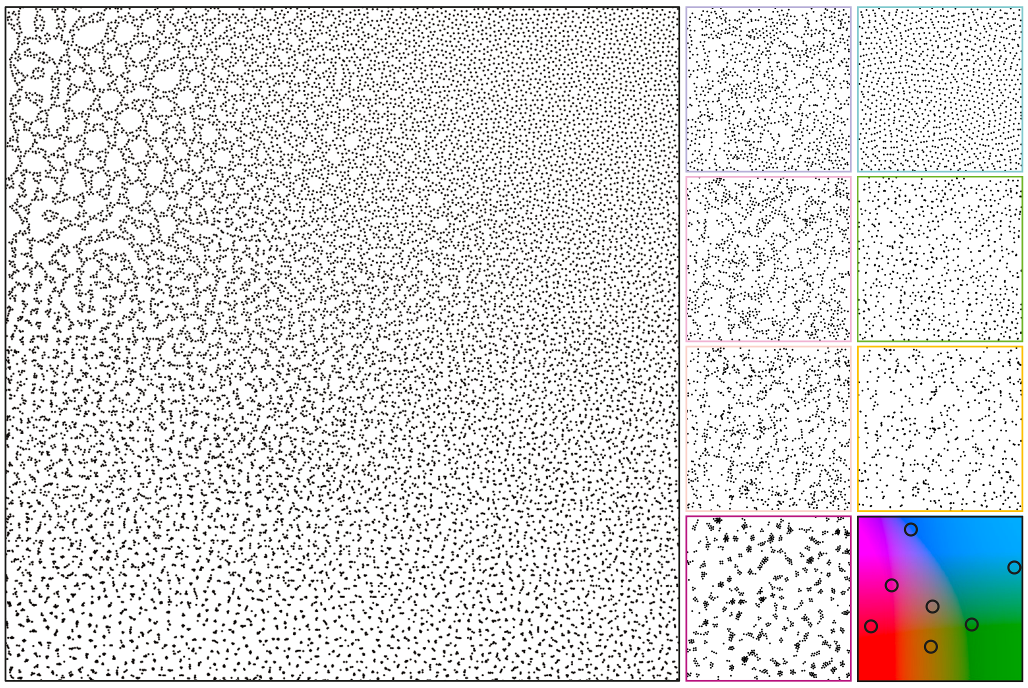


Figure 4.20: *The left image shows the resynthesized point pattern with a continuously varying correlation in the 2D latent space $f(\mathbf{z})$. The bottom-right (black) square shows different coordinates and their color in $CIELAB$. Point patterns with the corresponding spatially-invariant correlations for each of these coordinates are shown with respective color coding in the right half. Blue noise can be spotted on the top-right side of the space.*

Latent space visualizations. The latent space $f(\mathbf{z})$ of correlations is visualized in Fig. 4.20, and more visualizations can be found in appendix Fig. B.5. Every point in $f(\mathbf{z})$ is a PCF, and we use the machinery that will be defined in Sec. 4.2.4 to generate a point pattern that has such a spatially-varying correlation.

4.2.3 Edge-aware density and correlations

Assumptions. Input to this step is a discrete point pattern P, a continuous density map d, and a continuous guidance map h to handle spatially-varying correlations. The aim is to estimate correlation at one position so it is unaffected by points that fall onto locations on the other side of an edge. Output is a continuous spatially-varying correlation map.

Definition. We define the edge-aware radial density function as

$$\hat{g}(P,d,h)(\mathbf{x},r) = \sum_{i=1}^{n} \kappa(\mathbf{x},\mathbf{x}_i,r,d,h), \tag{4.8}$$

a mapping from location $\mathbf{x}$, radius r to density, conditioned on the point pattern P with n points subject to the kernel

$$\kappa(\mathbf{x},\mathbf{x}_i,r,d,h) = \frac{S(\mathbf{x}_i,\mathbf{x},r,d)\cdot H(\mathbf{x}_i,\mathbf{x},h)}{\sum_{\mathbf{y}\in\mathbb{R}^2} S(\mathbf{y},\mathbf{x},r,d)\cdot H(\mathbf{y},\mathbf{x},h)}, \tag{4.9}$$

that combines a *spatial* term S and a *guidance* term H, both to be explained next. Intuitively, this expression visits all discrete points $\mathbf{x}_i$ and soft-counts if they are in the "relevant distance" and on the "relevant side" of the edge relative to a position $\mathbf{x}$ and a radius r. The $\mathbf{y}$ locations represent a dense grid used to compute the normalization term as explained next.

Spatial term. The spatial S-term is a Gaussian N with mean r and standard deviation (bandwidth) σ. It is non-zero for distances similar to r and falls off with bandwidth σ:

$$S(\mathbf{x}_i, \mathbf{x}, r, d) = N\left(\frac{\|\mathbf{x}_i - \mathbf{x}\|_2}{d(\mathbf{x})}; r, \sigma\right) \tag{4.10}$$

The distance between two points is scaled by the density at the query position. As suggested by Zhou et al. [2012], this assures that the same pattern at different scales, i.e., densities, indeed maps to the same correlation. Bandwidth σ is chosen proportional to the number of points n.

Guidance term. The H-term establishes if two points $\mathbf{x}$ and $\mathbf{x}_i$ in the domain are related. This is inspired by the way Photon Mapping adapts its kernels to the guide by external information, such as normals [Jensen, 2001] or joint image processing makes use of guide images [Petschnigg et al., 2004]. If $\mathbf{x}$ is related to $\mathbf{x}_i$, $\mathbf{x}_i$ is used to compute correlation or density around $\mathbf{x}$, otherwise, it is not. Relation of $\mathbf{x}$ and $\mathbf{x}_i$ is measured as the pair similarity

$$H(\mathbf{x}, \mathbf{x}_i, h) = \|h(\mathbf{x})^T \cdot \Sigma \cdot h(\mathbf{x}_i)\|, \tag{4.11}$$

where h is a guidance map and Σ is the (diagonal) bandwidth matrix, controlling how guide dimensions are discriminating against each other. The guidance map can be anything with distances defined on it, but our results will use correlation itself.

Normalization. The denominator in Eq. (4.9) makes the kernel sum to 1 when integrated over $\mathbf{y}$ for a fixed $\mathbf{x}$ as in normalized convolutions [Knutsson and Westin, 1993]. Also, if κ extends outside the domain for points close to the boundary, this automatically rescales the kernel.

Example. Figure 4.21 shows this kernel in action. We show the upper left corner of the domain (white) as well as some area outside the domain (grey with stripes). Correlation is computed at the yellow point $\mathbf{x}$. The kernel's spatial support is the blue area. The area outside the domain (stripes) will get zero weight. Also, the grey area inside the domain, which is different in the guide (due to different correlation), is ignored, controlled by the H-term.

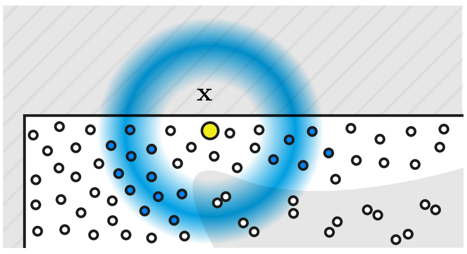


Figure 4.21: *Density estimation kernel.*

Discussion. Our formulation is different from the one used by Chen et al. [2013], who warp the space to account for guide differences, as Wei and Wang [2011] did for density variation. This does not work for patterns with varying correlation: a point on an edge of a correlation should not contribute to the correlation of another point in a scaled way. Instead of scaling space, we embed points into a higher-dimensional space [Chen et al., 2007; Jensen, 2001] only to perform density estimation on it. Consider estimating the PCF at the yellow point, close to an edge between an area of blue and an area of green noise as shown on Fig. 4.22. The dark gray point lies on the other side of the edge. In common bilateral handling, it would contribute to a different bin (orange horizontal movement, distance in PCF space) in the formulation of Wei [2010], which is adequate for density, but not for correlation. Our approach suppresses those points (blue arrow and vertical movement, density in PCF space).

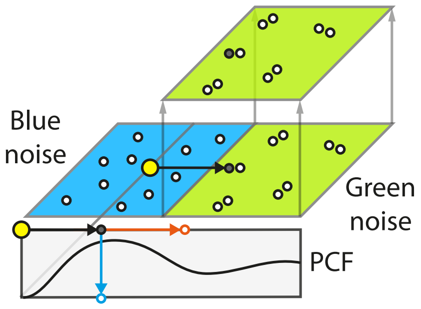


Figure 4.22: *Bilateral.*

4.2.4 Synthesis and Editing

We will now derive a method to synthesize a point pattern with a desired spatially varying target density and correlation (Sec. 4.2.4). Users can then provide these target density and correlation as common raster images (Sec. 4.2.4) and edit them in legacy software (Sec. 4.2.4).

Synthesizing with desired correlation and density

Given Eq. (4.8), we can estimate $\hat{g}(P, d, h)$, the spatially-varying PCF field for a triplet of point pattern P, density map d, and guidance map h. Recall, we can also compare this spatially-varying PCF to another one, we shall call $\bar{g}$, e.g., by spatially and radially integrating their point-wise norms. Hence, we can then optimize for a point pattern P to match a given density map d, a given guidance map h, and a given target PCF $\bar{g}$ by

$$\arg\min_P c_{\text{Syn}}(P) = \int_x \int_r (\hat{g}(P, d, h)(\mathbf{x}, r) - \bar{g}(\mathbf{x}, r))^2 \, \mathrm{d}\mathbf{x} \, \mathrm{d}r. \tag{4.12}$$

We use $\bar{g}$ as our guidance map h to evaluate $\hat{g}$. Note that we do not explicitly ask the result P to have a specific density. This happens implicitly: recall that our edge-aware correlation estimate Eq. (4.10) will scale point-wise distances according to density before computing the PCF. Hence, the only way for the optimization to produce the target $\bar{g}$ is to scale the distances proportional to density.

In practice, we Monte Carlo-estimate this objective using 10×10 jittered samples $\{\mathbf{q}_i\} \in [0, 1]^2$ along the spatial dimension and m regular samples $\{r_j\}$ along the radius dimension (ranging from 0.1 to $2r_{\max}$ as detailed in Sec. 4.2.4), as in

$$\hat{c}_{\text{Syn}}(P) = \sum_{i=1}^{10\times10} \sum_{j=1}^{m} (\hat{g}(P, d, \bar{g})(\mathbf{q}_i, r_j) - \bar{g}(\mathbf{q}_i, r_j))^2. \tag{4.13}$$

and optimize over P using Adam [Kingma and Ba, 2014].

To have this synthesis produce a point pattern P, a user now only needs to provide a spatially-varying density d and correlation $\bar{g}$.

Encoding into raster images

Users provide density d, and correlation $\bar{g}$ as common discrete raster images on which

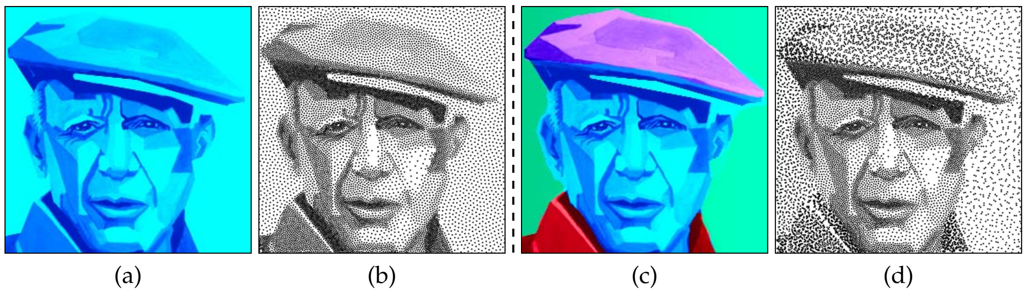


Figure 4.23: *We start with a given density and correlation map (a), and perform edits directly on this three-channel raster image (c) to obtain the point pattern in (d) after resynthesis. In (c), we edit the density map (L-channel) with gradients from left to right and edit correlations in the AB-channel to enhance different segments of the image. No neural network is used here. Source image credit: Artmajeur user Bianchini Jr. Used with permission under Media Licence.*

we assume suitable sampling operations to be defined (e.g., bilinear interpolation) to evaluate them at continuous positions.

For density d, this is trivial and common practice in the point pattern literature. For correlation, $\bar{g}$, we use our embedding f from Sec. 4.2.2 to decode the complex PCF from a latent 2D coordinate, from only two numbers per spatial location. Hence, density and correlation require only three numbers per spatial position. We pack those three numbers into the three image color channels. More specifically, density into the L and latent correlation into the AB channel of a $CIELAB$ color image, we call F.

Editing point patterns as raster images

Any editing operation that is defined on a three-channel image in any legacy image manipulation software can now be applied to the point pattern feature image F.

Working in $CIELAB$ color space, users have the freedom to select the first channel to edit density, the latter two channels to edit the correlations, or edit both at the same time. Since L and AB channels do not impact each other, $CIELAB$ color space is ideal for manipulating the density or the correlation independently, as luminance and chrominance are perceptually decorrelated by design.

While this is, in a sense, the optimal space to use when editing point patterns in legacy image editing software, it is not necessarily an intuitive user interface. In a fully-developed software system, on top of legacy software, a user would not be shown colors or pick colors to see, respectively, edits, correlations. Instead, they would only be presented with the point pattern, and select from an image similar to Fig. 4.20, and all latent encoding would be hidden. We give examples of such edits in Fig. 4.23. For the case of ab-initio design, no input point pattern is required, and a user would freely draw correlation and density onto a blank canvas.

Implementation

We implement our framework mainly in PyTorch [Paszke et al., 2017]. All experiments run on a workstation with an NVIDIA GeForce RTX 2080 graphics card and an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz.

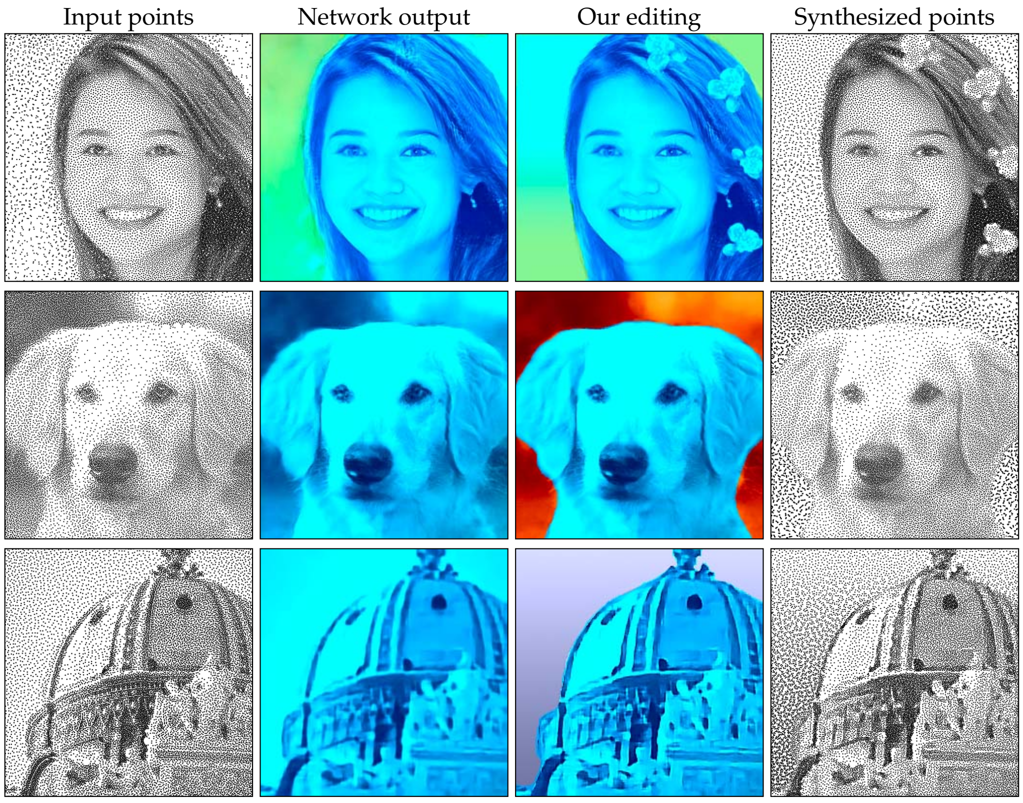


Figure 4.24: *Input points (first column) generated by our synthesis and mapped to raster images (second column). An edit is performed in Adobe Photoshop (third column), and a new point pattern is resynthesized (fourth column). In the first row, we edit the background correlation using the AB channels and add some flowers on the hair using the L channel. The second row shows another edit where we change the correlation in the background using the AB channels. In the third row, we change the correlation in the background (AB channels) and add a density gradient (L channel). To generate the results in the second column from the first column, we use the neural networks trained on our human faces, animal faces, and churches datasets, respectively.*

Embedding. In total, we collect 1,000 point patterns, and each of them has 1,024 point samples. To perform MDS, the 2D latent coordinates $\{\mathbf{z}_i\}$ are initialized randomly in $[0, 1]^2$. The MDS optimization Eq. (4.4) runs in batches of size 20, using the ADAM optimizer [Kingma and Ba, 2014] with a learning rate of 0.005 for $1,000$ iterations.

If latent coordinates are quantized to 8 bits, there are only 256^2 many different possible correlations $\{g_i\}$. We pre-compute these and store them in a $256 \times 256 \times m$ look-up table lut w.r.t. each latent 2D coordinate to be used from here on.

Edge-aware PCF estimator. To compute the pair similarity between two guide values in h, the bandwidth matrix Σ is set to be 0.005-diagonal. We use $m = 20$ bins to estimate the PCF. The binning is performed over the distance range from 0.1 to 2 $r_{\max}$, where $r_{\max} = 2\sqrt{\frac{1}{2\sqrt{3}n}}$. The point count n is chosen as the product between a constant and the total inverse intensity in the L-channel of the given feature image F, such that an image with dark pixels has more points. To compute local PCF for each pixel in F, we consider

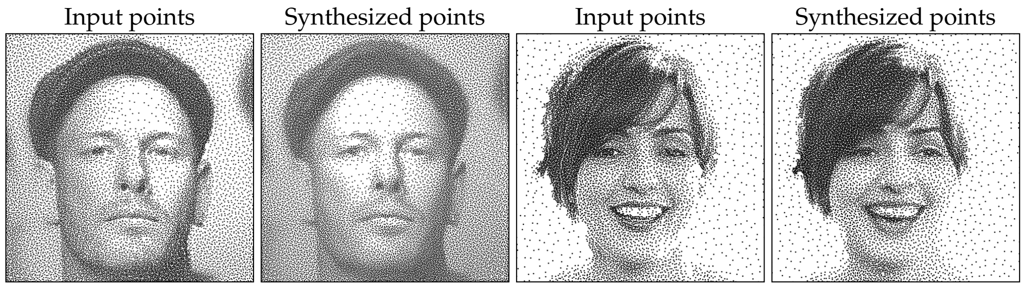


Figure 4.25: *Our neural network maps the input point pattern to a density and correlation map. To obtain different point pattern editing effects, we apply different advanced filters to the output density map. From left, the second column shows the relighting effect using the method by ClipDrop [2023]. The fourth column shows the change in the facial expression and the eye direction performed using a neural filter from Adobe Photoshop.*

only the k-nearest neighbor points, and not all points, where $k = 50$ and $\sigma = 0.26$.

With each PCF, g_i we also pre-compute and store λ_i, the best learning rate (LR) (see Appendix B.1.2 for a definition) as we found different spectra to require different LRs. During synthesis, we find the LR for every point by sampling the correlation map at that point position and using the LR of the manifold-nearest exemplar of that correlation.

Synthesis. The initial points when minimizing Eq. (4.12) are randomly sampled in $[0,1]^2$ proportional to the density map d and the optimization runs for $5,000$ iterations. We note that the denominator in Eq. (4.8) is a sum over many points, which could be costly to evaluate, but as it does not depend on the point pattern P itself, it can be pre-computed before optimizing Eq. (4.12). We use `C++` and `Pybind11` to accelerate this computation, and the whole initialization stage takes around 5 seconds.

To faithfully match the intensity of point stipples with the corresponding density map [Spicker et al., 2017], we also optimize for dot sizes as a post-processing step.

Editing. We use Adobe Photoshop 2022 for editing, which has native $CIELAB$ support. We devised two simple interfaces scripted into Adobe Photoshop 2022, one for interactive visualization of colors and point correlations and the other for editing and synthesis.

In this section, we demonstrate our design choices and compare our pipeline with existing methods through some application scenarios. We also perform a user study to evaluate the effectiveness and usability of our framework. More results can be found in the accompanying supplemental HTML.

4.2.5 Ab-initio point pattern design

We demonstrate examples of point pattern edits in Fig. 4.16 and Fig. 4.23. We start with a given density and correlation map (1st and 3rd columns). We choose blue noise for the correlation map to start with and start editing both the density (L-channel) and the correlation (AB-channel). For Fig. 4.16, we add a density gradient transition on the background and add a simple leaf by editing the density channel. We also assign different correlations to different segments, like the butterfly, leaf, and background. For Fig. 4.23, the correlation edits are done in the AB channel of the Picasso image to assign different

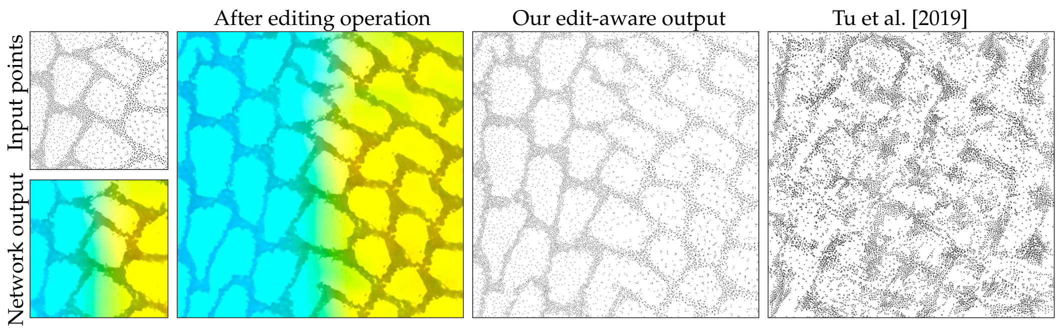


Figure 4.26: *Starting from an input pattern of tree cover density (left, top), we use a network specialized on geomatic data to estimate correlation and density (left, bottom). We can then apply existing image operations, such as Adobe Photoshop's "Content-Aware Fill" (second column), to achieve "Point Pattern Expansion" (third column), which compares favorably to direct optimization of points to match the inputs' VGG-based Gram matrices, deep correlation matrices, and histograms [Tu et al., 2019] (last column).*

correlations to different image segments, e.g., background, face, cap, and the shirt. We also add a gradient density in the background from left to right by editing the L-channel of the image.

We show more results in Appendix Fig. B.4 to demonstrate that our framework provides a straightforward way to edit spatially-varying point correlations by picking correlations from our correlation space (Fig. 4.20), instead of by designing PCFs or power spectra.

4.2.6 Neural network-aided point pattern design

Besides manually drawing correlation and density, we propose a second alternative: a neural network (NN), based on `pix2pix` [Isola et al., 2017], which automatically produces density and correlation maps from legacy point patterns. We curated paired synthetic datasets for class-specific training from three categories, including human faces from `CelebA` by Lee et al. [2020], animal faces from Choi et al. [2020], and churches from Yu et al. [2015a]. As the NN maps point patterns to raster images, training data synthesis proceeds in reverse: For each of the three-channel raster images, the gray-scale image of each original image is directly used as the L-channel. We generate the correlation map (AB-channel) by assigning them random chroma, i.e., latent correlation coordinates. Next, our synthesis from Sec. 4.2.4 is used to instantiate a corresponding point pattern. Finally, the patterns are rasterized, as `pix2pix` works with raster images. For further data generation, network architecture and training details, please see Appendix B.1.4. We also perform an ablation study on the network architecture and training in appendix Fig. B.6.

This pipeline enables freely changing local density or correlation in point patterns of different categories as seen in Fig. 4.24. As shown in Fig. 4.25, this also allows advanced filtering such as relighting or facial expression changes on point patterns. In appendix Fig. B.7, we show additional results where the input point patterns, generated by other image stippling methods [Zhou et al., 2012; Salaün et al., 2022], can be edited using our framework.

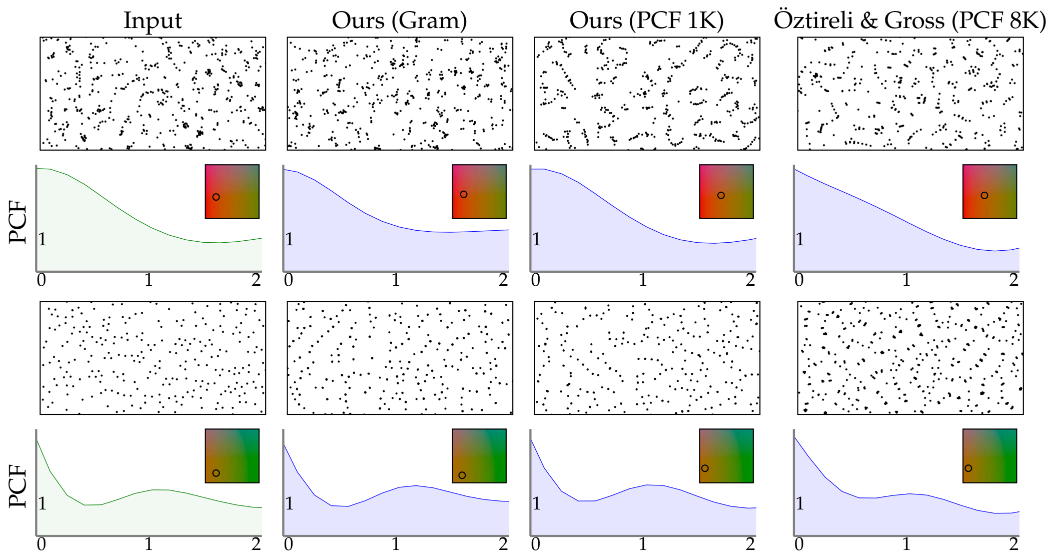


Figure 4.27: *For an input point pattern, we query its nearest neighbor in the latent space. Using Gram matrices (middle-left), the nearest neighbor quality is closer to the input than using PCF (middle-right). The inset images alongside PCFs show the latent space coordinates. The correlation space from Öztireli and Gross [2012] uses 8K PCFs but is still not diverse enough to provide a good nearest neighbor for the input. Our latent space has only 1000 base point patterns. We also perform a user study on this, as detailed in Sec. 4.2.8.*

4.2.7 Point pattern expansion

Here we train our network on density from the Tree Cover Density [Büttner and Kosztra, 2017] dataset in combination with random spatially-varying correlation maps using anisotropic Gaussian kernel with varying kernel size and rotation as detailed in Appendix B.1.3. Similar works can be found in Kapp et al. [2020], Tu et al. [2019] and Huang et al. [2022] for sketch-based or example-based point distribution synthesis.

Figure 4.26 illustrates one such representative example of point-based texture synthesis using our method. Our network reconstructs the density and correlation map that captures the gradient change of correlation and spatially-varying density. By using the content-aware filling tool in Adobe Photoshop, we can perform texture expansion (second column) based on the network output. More specifically, we first expand the canvas size of the map, select the unfilled region, and use content-aware filling to automatically fill the expanded background. We also compare our method with a state-of-the-art point pattern synthesis method from Tu et al. [2019] which takes an exemplar point pattern as input and uses VGG-19 [Simonyan and Zisserman, 2015] features for optimization.

4.2.8 Experiments

Realizability We construct the manifold (Fig. 4.20) from exemplars that are realizable, but an interpolation of realizable PCF is not guaranteed to be realizable. We test if realizability still holds in practice as follows. For each PCF $g(\mathbf{z}_i)$, we generate a point

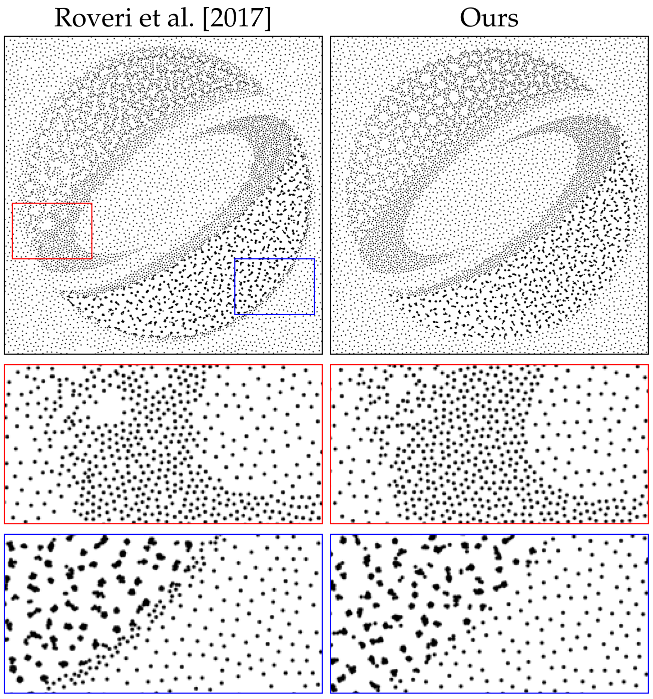


Figure 4.28: *We compare the synthesis quality of our optimization against Roveri et al. [2017]. To run Roveri et al. method (left), we use the stored PCFs within each pixel of F. The zoom-ins in the bottom two rows show that Roveri et al. cannot handle the sharp transitions in the correlations.*

set instance using our method and compute the resulting PCF $g'(\mathbf{z}_i)$. We then compute $A = \mathbb{E}_i[(|g'(\mathbf{z}_i) - g(\mathbf{z}_i)|)]$, the average error between PCF and realized PCF and $B = \max_{ij}(|g(\mathbf{z}_j) - g(\mathbf{z}_i)|)$ the maximum difference between any two PCFs. The relative error A/B is 3×10^{-5}, indicating that the realization error is five order of magnitude smaller than the PCF signal itself.

Comparisons

Comparisons with Öztireli and Gross [2012]. To the best of our knowledge, Öztireli and Gross [2012] is the most relevant related work which studies the space of point correlations using PCFs. However, we observe that PCFs are not the best way to characterize the perceptual differences between different point correlations. In Fig. 4.27, we show that the Gram matrices (our input proximity to MDS) better encode the perceptual similarity between neighboring point correlations.

Comparisons with Roveri et al. [2017]. Figure 4.28 shows an example of synthesizing point patterns using our synthesis method and the synthesis method proposed by Roveri et al. [2017]. To the best of our knowledge, Roveri et al. [2017] is the only competitor that supports point pattern synthesis from spatially-varying density and correlation. We demonstrate that their method may synthesize point patterns with artifacts around the sharp transitions between two correlations (bottom-right and top-left of the logo). Our method, by taking the bilateral term into account, can handle sharp transitions between correlations more accurately.

This relation can be further quantified as follows: Let P be a point pattern, $\bar{P}$ be an edited version of that, and $g(P)$, respectively, $g(\bar{P})$ be their correlations. Now, first, let $g_{\mathrm{PCA}}(\bar{P})$ be the correlations of $\bar{P}$, projected into the space spanned by the PCA of the correlations in and only in P, and, second, $g_{\mathrm{Ours}}(\bar{P})$ be the correlations of $\bar{P}$, projection into our palette (Fig. 4.20). The error of those projections is $e_{\mathrm{PCA}} = |g_{\mathrm{PCA}}(\bar{P}) - g(\bar{P})|$ and $e_{\mathrm{Ours}} = |g_{\mathrm{Ours}}(\bar{P}) - g(\bar{P})|$, respectively. We evaluate these error values for different figures. Fig. 4.16 shows 96× improvement, Fig. 4.23 shows 130× improvement and the three rows in Fig. 4.24 shows 471×, 461× and 59× improvement using our approach (e_{Ours} vs. e_{PCA}). This indicates that our latent space can preserve one to two orders of magnitude more edit details than Roveri et al. [2017] approach, which restricts itself to the input exemplar.

User study

We performed a set of user experiments to verify i) the perceptual uniformity of our embedding, ii) the ability of users to navigate in that space iii) the usefulness of the resulting user interface.

Embedding user experiment. 34 subjects (S) were shown a self-timed sequence of 9 four-tuples of point patterns with constant density and correlation in a horizontal arrangement (Fig. 4.27). The leftmost pattern was a reference. The three other patterns were nearest neighbors to the reference in a set of: i) our basis patterns using our perceptual metric, ii) our basis patterns using the PCF metric, and iii) patterns from the PCF space suggested by Öztireli and Gross [2012] using the PCF metric. Ss were instructed to rate the similarity of each of the three leftmost patterns to the reference on a scale from 1 to 5 using radio buttons.

The mean preferences, across the 10 trials, were a favorable 3.59, 2.52, and 2.55, which is significant ($p < .001$, two-sided t-test) for ours against both other methods. A per-pattern breakdown is seen in Fig. 4.29. This indicates our metric is perceptually more similar to user responses than the two other published ones. Figure 4.27 shows two examples where the nearest-neighbor of the query point pattern is perceptually different using different metrics. The PCFs of two point patterns can be similar even when they are perceptually different.

Navigation user experiment. In this experiment, $N = 9$ Ss were shown, 8 reference point correlations, and the palette of all correlations covered by our perceptual embedding (Fig. 4.20). Ss were asked to pick coordinates in the second image by clicking locations in the embedding, so that the corresponding point correlations of the picked coordinates perceptually match the reference correlations.

The users' locations were off by 14.9 % of the embedding space diagonal. We would not be aware of published methods for intuitive correlation design to compare this number to. Instead, we have compared the mistakes users make when picking colors using the LAB color picker in Adobe Photoshop. Another $N = 9$ Ss, independent of the ones shown the palette of correlations, made an average mistake of 21.3% in that case. We conclude that our embedding of pattern correlation into the chroma plane is significantly more intuitive ($p < 0.2$, t test) than picking colors. Fig. 4.30 shows the points users clicked (small dots), relative to the true points (large dots).

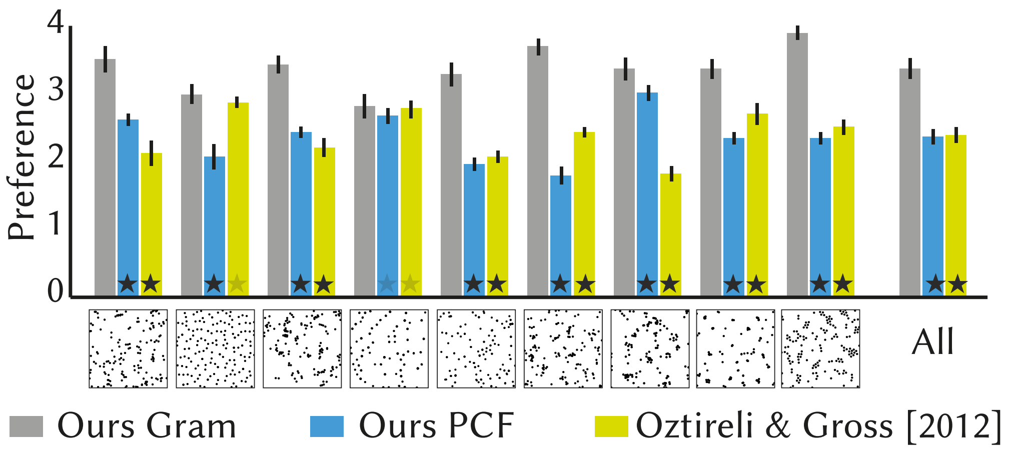


Figure 4.29: *Results of the embedding user experiment. Error bars are standard errors of the mean. A black or gray star is significant against our method at the* $p = .001$ *or* $.005$*-level.*

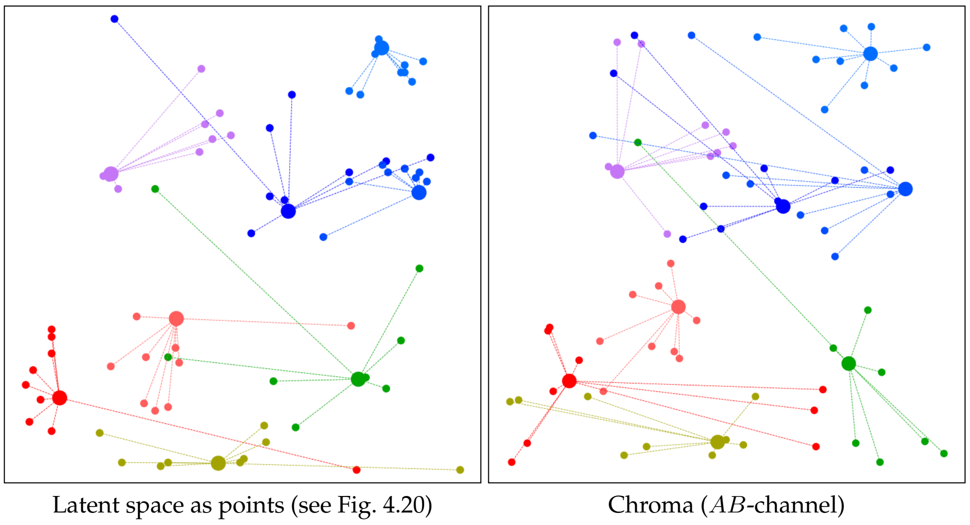


Figure 4.30: *Results of the navigation user experiment. Users are shown the latent space visualized as spatially-varying points (Fig. 4.20) on the left and as spatially-varying chroma on the right. The large dots represent the locations of our selected 8 reference point patterns. The small dots are the locations that the users chose as perceptually similar point patterns wrt. the reference.*

Usefulness experiment. Ss were asked to reproduce a target stippling with spatially varying correlation and density by means of Adobe Photoshop, which was enhanced to handle point correlation as LAB space using our customized interface. Ss were shown the point patterns of our perceptual latent space, and asked to edit the density and correlation separately to reproduce the reference from an initialized LAB image. After each editing, generating the point pattern incurred a delay of one minute. Note that we intentionally reduce the number of iterations to optimize the point patterns from user edits to offer faster feedback in around a minute with around 10,000 points. Details on this experiment are found in Appendix B.2.4.

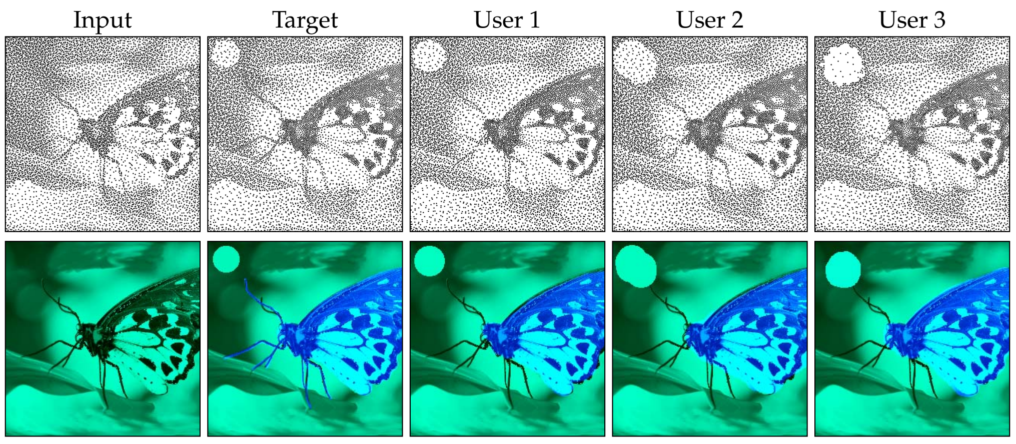


Figure 4.31: *User edits from the usefulness experiment. We show that users can use our system to design point patterns (first row) by editing L-channel and AB-channel (second row) to match the target one from an initialized input.*

There is no objective metric to measure the quality, so we report three user-produced point patterns in Fig. 4.31. The whole process takes 15 minutes on average for all three users, as they usually require multiple trials to pick the correlations, and our synthesis method does not run interactively.

Performance

We summarize the run-time statistics of our synthesis method w.r.t. the number of synthesized points in Fig. 4.32. Editing time is not reported as it can be biased by the editing skills of the users.

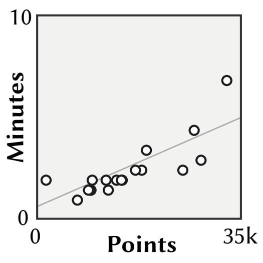


Figure 4.32: *Timing.*

4.2.9 Conclusion

We propose a novel framework to facilitate point pattern design by introducing a perceptual correlation space embedded in a two-channel image using a dimensionality reduction method (MDS). This allows users to design or manipulate density and correlation by simply editing a raster image using any off-the-shelf image editing software. Once edited, the new features can be resynthesized to the desired point pattern using our optimization (Sec. 4.2.4). To better handle sharp transitions in density and correlation during synthesis, we formulate a novel edge-aware PCF (Sec. 4.2.3) estimator. The resulting framework allows a wide range of manipulations using simple image editing tools that were not available to point patterns before. Users can design novel spatially varying point correlations and densities without any expert knowledge on PCF or power spectra, or can use a neural network to get correlation and density for a specific class, such as faces.

Limitations. The latent space spanned by the basis point correlations in Fig. 4.20 is by no means perfect. Synthesizing point patterns with smooth transitions from two extreme locations in the latent space may result in some unwanted artifacts. Our synthesis method takes minutes to synthesize points, which is far from the interactive rate that is more friendly to artists and users.

Future work. A neural network with such a latent space is a promising direction where the correlations within the geomatic data can be predicted and edited according to user-defined conditions (environment, pollution, global warming, etc). Our approach happens to rasterize points, which ideally is to be replaced by directly operating on points [Qi et al., 2017a; Hermosilla et al., 2018].

The editing operations can also be improved by proper artistic guidance or by interactive designing tools that intuitively manipulate the desired density and correlations for desired goals. Extending our pipeline for visualization purposes is another fascinating future direction. Another interesting future direction is to extend the current pipeline to multi-class and multi-featured point patterns, which appear in real-world patterns. Designing a latent pattern space that spans a larger gamut of correlations (also anisotropic and regular ones) present in nature (e.g., sea shells, trees, stars, galaxy distributions) is a challenging future problem to tackle. There is an exciting line of future work that can be built upon our framework. Many applications, like material appearance, haptic rendering, smart-city design, reforestation, and planet colonization, can benefit from our framework.

Chapter 5
Noise correlations for diffusion-based image and video generation

In this chapter, we discuss how noise correlation can impact the training and inference of diffusion models.

- Our first project of this chapter (Sec. 5.1) shows that injecting correlated noise, such as blue noise, can help train diffusion models for generating finer image details compared to using random Gaussian noise.
- Secondly, we explore how noise can also be useful for training diffusion models for stereo video generation and restoration (Sec. 5.2), by training with noise-degraded videos.

5.1 Blue noise for diffusion models

Most of the existing diffusion models use Gaussian noise for training and sampling across all time steps, which may not optimally account for the frequency contents reconstructed by the denoising network. Despite the diverse applications of correlated noise in computer graphics, its potential for improving the training process has been underexplored. In this section, we introduce a novel and general class of diffusion models taking correlated noise within and across images into account. More specifically, we propose a time-varying noise model to incorporate correlated noise into the training process, as well as a method for fast generation of correlated noise masks. Our model is built upon deterministic diffusion models and utilizes blue noise to help improve the generation quality compared to using Gaussian white (random) noise only. Further, our framework allows introducing correlation across images within a single mini-batch to improve gradient flow. We perform both qualitative and quantitative evaluations on a variety of datasets using our method, achieving improvements on different tasks over existing deterministic diffusion models in terms of the FID metric.

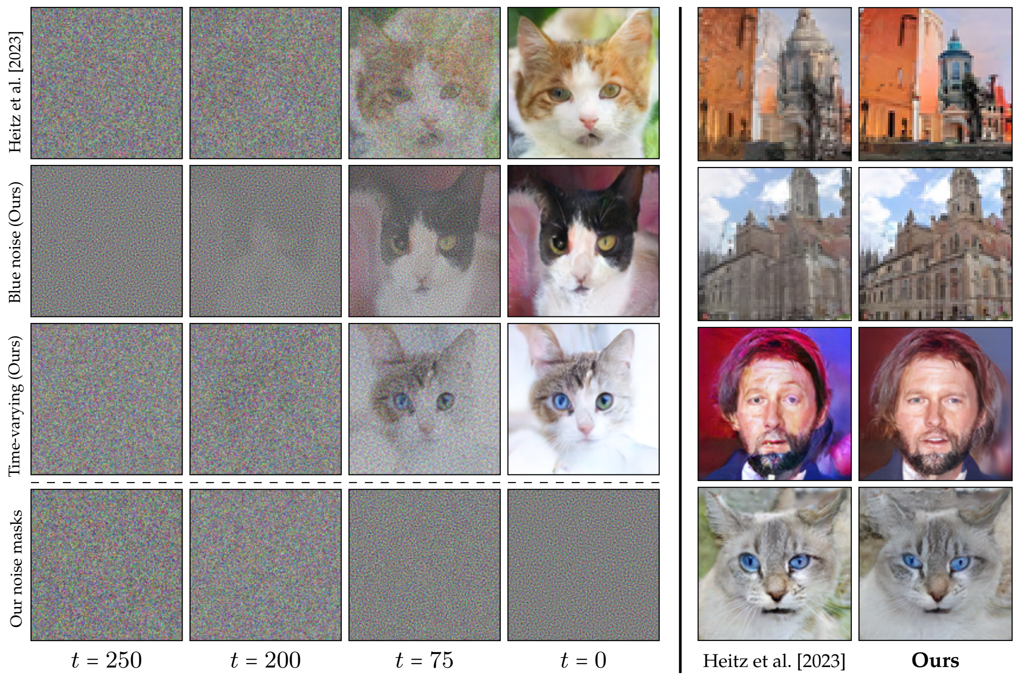


Figure 5.1: *In conventional diffusion-based generative modeling, data is corrupted by adding Gaussian (random) noise (as illustrated in the top row). Here, we explore the alternative approach of using correlated Gaussian noise in diffusion-based generative modeling. Different correlations can be used, such as blue noise (second row) or time-varying correlations (third row). The bottom row shows the corresponding noise mask for the time-varying example at different time steps. The rightmost two columns show visual comparisons on generated images (from the same initial noise) between Heitz et al. [2023] (using only Gaussian noise) and Ours (using time-varying noise). Our generated images are more natural-looking and detailed with fewer artifacts.*

5.1.1 Introduction

Since the groundbreaking work by Sohl-Dickstein et al. [2015]; Ho et al. [2020]; Song and Ermon [2019], there has been extensive research on diffusion models. These models have demonstrated superior performance in terms of generative quality and training stability compared to Generative Adversarial Networks (GANs) [Dhariwal and Nichol, 2021] for image synthesis. Additionally, diffusion models can be trained to perform various tasks such as text-to-image synthesis, image inpainting, image super-resolution, and image editing.

Typically, a diffusion model consists of two processes: forward and backward. In the forward process, the model gradually adds noise to an original data point (e.g., an image), transforming it into a random noise pattern. In the backward process, the model learns to reconstruct the original data from this noise using a denoising neural network. The denoising network initially focuses on reconstructing the coarse shape and structure (low-frequency components) in the early time steps. As the time steps decrease, the denoising network progressively refines the details (high-frequency components). This behavior indicates that diffusion models generate data in a coarse-to-fine manner and have a hidden relationship with frequency components.

Despite these advancements, there is limited research on the relationship between this behavior and the noise used during the forward and backward processes. Most existing diffusion models rely solely on Gaussian noise, also known as uncorrelated Gaussian noise or Gaussian white noise, as its frequency power spectrum spans all frequencies (similar to the *white* color). While correlated noise has not been thoroughly examined in diffusion models, there has been some relevant research that has delved into this domain. For example, Rissanen et al. [2023] proposes a diffusion process inspired by heat dissipation to explicitly control frequencies. Similarly, Voleti et al. [2022] suggests using non-isotropic noise as a replacement for isotropic Gaussian noise in score-based diffusion models. Despite their potential, both methods face limitations regarding the quality of the generated images, which could explain their limited adoption in mainstream models.

In this section, we propose a new diffusion model that supports a diffusion process with time-varying noise. Our goal is to utilize correlated noise, such as blue noise ([Ulichney, 1987]), to enhance the generative process. Blue noise is characterized by a power spectrum with no energy in its low-frequency region. Our focus is on blue noise masks ([Ulichney, 1999]), which provide noise profiles with blue noise properties. We propose using these blue noise *masks* to design a time-varying noise for diffusion-based generative modeling. Generating such correlated noise masks for diffusion is a time-consuming process, as it may require generating thousands to millions of masks on the fly. To address this issue, we propose an efficient method to generate Gaussian blue noise masks on the fly for both low-dimensional and high-dimensional images. In summary, our contributions are as follows:

- We propose a framework that investigates the impact of correlated noise and correlation across training images on generative diffusion models.
- Based on our framework, we introduce a deterministic diffusion process with time-varying noise, allowing control over the correlation introduced in the model at each step.
- We overcome the computational challenges of generating correlated noise masks by introducing a real-time mask generation approach.
- By interpolating Gaussian noise and Gaussian blue noise using our proposed time-varying noise model, our model outperforms existing deterministic models like IADB [Heitz et al., 2023] or DDIM [Song et al., 2021b] in various image generation tasks.

5.1.2 Diffusion models with uncorrelated Gaussian noise

A generative model based on diffusion comprises two key processes: a forward process and a backward process. In the forward process, noise, denoted as ϵ, is introduced to corrupt an initial image, $\mathbf{x}_0$, by scaling it with a factor determined by a discrete-time parameter, t. Here, $\mathbf{x}_0$ represents a real image sampled from the training data distribution, denoted as p_0. The time step, t, ranges from 0 to $T-1$, where T is the total number of discrete time steps. The corrupted image, along with the corresponding time step t, is then used as input to train a neural network, $f_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$.

In the backward process, the trained network is employed to denoise pure noise and generate new images. Figure 5.2 illustrates this process. Starting from Gaussian noise (blue distribution), the image iteratively passes through the network to eventually yield a fully denoised image, aligning with the target distribution (red distribution). Intermediate steps of the process involve a mixture of noise and image. Three examples are visible

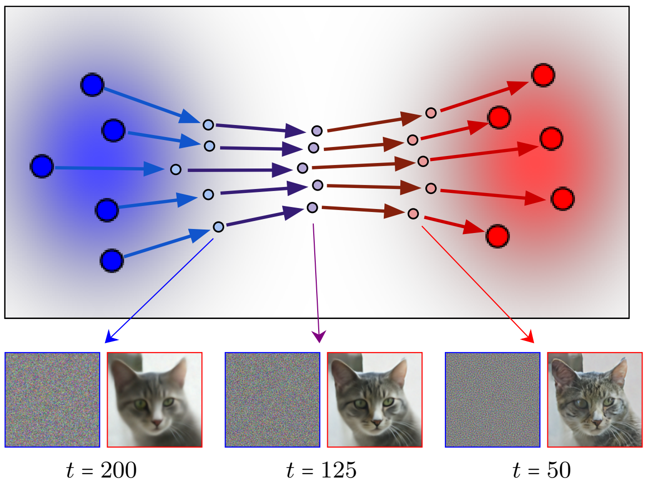


Figure 5.2: *Schema of diffusion process using our time-varying noise. The diffusion transforms the initial noise distribution (blue) into the target data distribution (red). Five examples are shown with the intermediate diffusion steps between the two distributions. For one of the data we illustrate the intermediate time steps with the current expected result and noise. The evolution of the noise from random to blue noise is visible, as well as the quality of the expected result.*

in the figure. As more time steps are executed (with t closer to 0), the image quality improves, and more details emerge. In this example, the intermediate noise transitions from Gaussian noise to Gaussian blue noise following a time-varying schedule, following Sec. 5.1.4.

This section explores correlations across two different axes: across pixels in the noise and across images within a mini-batch. To demonstrate the impact of correlations between noise masks and images, we build a deterministic diffusion process with time-varying noise following the work by Heitz et al. [2023], namely, the IADB method. For the sake of simplicity and fair one-to-one comparison, our method was developed on top of IADB while preserving its characteristics and hyperparameters other than the ones described as new in our method. But our method is general enough that it could be potentially explored on top of other existing generative diffusion processes.

For IADB, the forward and backward processes and the objective function are defined as follows:

$$\mathbf{x}_t = \alpha_t \boldsymbol{\epsilon} + (1 - \alpha_t)\mathbf{x}_0 \tag{5.1}$$

$$\mathbf{x}_{t-1} = \mathbf{x}_t + (\alpha_t - \alpha_{t-1}) f_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \tag{5.2}$$

$$L_{IADB} = \sum_t (f_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - (\mathbf{x}_0 - \boldsymbol{\epsilon}))^2 \tag{5.3}$$

With $\mathbf{x}_0$ a target image, $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \boldsymbol{I})$ a random Gaussian noise and α_t and α_{t-1} two blending coefficient. The network model is referred to as $f_{\boldsymbol{\theta}}$ and takes 2 input parameters: a corrupted image $\mathbf{x}_t$ and the timestep t. A stochastic formulation of IADB also exists, but this work will focus on the deterministic variant for its stability.

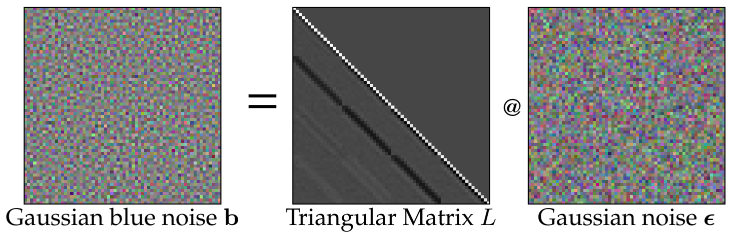


Figure 5.3: *To generate on-the-fly Gaussian blue noise masks* $\mathbf{b}$ *(leftmost), we pre-compute a lower triangular matrix L. We then multiply this matrix with Gaussian noise ϵ to obtain* $\mathbf{b} = L\epsilon$*. For Gaussian noise of size* 64×64*, the lower triangular matrix has a size of* $64^2 \times 64^2$*. Here we show a* 64×64 *zoom-in version of the matrix to better visualize the positive and negative correlations shown as the white and black lines, respectively.*

5.1.3 Correlated noise

In a deterministic diffusion process, noise masks are used as initialization of the backward process to generate images and at each training step to corrupt target images. Mask generation during training is a critical factor and must meet specific requirements. The process must be stochastic to produce different masks at each iteration, reducing overfitting and increasing diversity in the generated results. Mask generation also requires to be of fast computation as it is employed at every training step.

Gaussian noise naturally meets these requirements, but it is not the case for all correlated mask methods. In particular, IADB uses masks generated from a multivariate Gaussian distribution with zero mean and an identity covariance matrix. A method to create correlated noise, such as blue noise, requires a non-identity covariance matrix. The covariance matrix of the blue noise mask can be estimated from a collection of masks. We employed simulated annealing using the objective function from Ulichney [1993] to generate ten thousand blue noise masks. While this method yields high-quality masks, it does require significant optimization time. Then the blue noise correlation matrix Σ can be computed by averaging the respective correlation matrix of the example masks.

To create a noise mask with a specified covariance matrix Σ, the Cholesky decomposition is applied to Σ, resulting in a lower-triangular matrix L ($LL^T = \Sigma$). Finally, the random vector is multiplied by L to produce the desired noise mask efficiently:

$$\mathbf{b} = L\epsilon \tag{5.4}$$

where $\epsilon \sim N(0, I)$ is a unit-variance Gaussian distribution.

Figure 5.3 shows one realization of Gaussian blue noise mask generated using Eq. (5.4). Each row or column in the L represents a pixel index of the noise mask. Each cell in the L represents the correlation strength between the pixels in the noise mask. Positive values correspond to bright cells that represent positive correlations. Similarly, negative values correspond to dark cells representing negative correlations. For each pixel, only its adjacent pixels have values far from zero, while other non-adjacent pixels have values close to zero, as indicated by the white and black lines. Note that we use the name Gaussian blue noise for $\mathbf{b}$, but it is different from the Gaussian Blue Noise method from Ahmed et al. [2022].

Higher-dimension. Matrix-vector multiplication is computationally expensive when the L matrix is high-dimensional. Increasing the matrix size to generate higher-dimensional

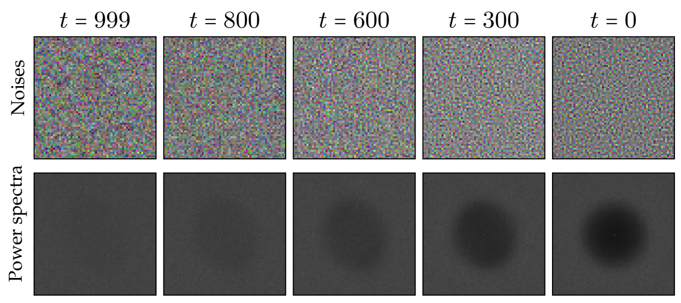


Figure 5.4: *Visualization of linearly interpolated noises from Gaussian noise to Gaussian blue noise at resolution* 64^2 *(top row), and the corresponding frequency power spectra (bottom row).*

noise becomes slower than generating (uncorrelated) Gaussian noise using a modern machine learning framework such as PyTorch [Paszke et al., 2017]. As noise generation is used at every training step, the overhead should remain minimal. Therefore, for generating higher-dimensional noise masks, we adapt Kollig and Keller [2002] method for producing our blue noise masks in which a padding of a set of lower-dimensional masks is applied.

More specifically, to generate a batch of high-dimensional Gaussian blue noise mask, a larger batch is generated at resolution 64^2 using Eq. (5.4) with $L \in \mathbb{R}^{64^2 \times 64^2}$. Then the 64^2 masks are padded into larger tiles to get Gaussian blue noise masks at higher dimensions. Thus, the computation overhead for generating a 128^2 resolution Gaussian blue noise is negligible (~ 0.0002 seconds). Figure 5.1 shows examples of Gaussian blue noise with resolution 128^2 generated by padding. Using padding for a higher-dimensional mask creates seams between the padded tiles. This artifact is practically not visible and is compensated by the low overhead of the method. We provide the masks at different resolutions, with corresponding frequency power spectra for Gaussian blue noise in Appendix C.3, demonstrating that the property of blue noise is preserved at different resolutions using our padding method.

5.1.4 Diffusion model with time-varying noise

With a single matrix L, only a single correlation can be generated. For a diffusion model, controlling the amount of correlation introduced within the model at each time step is necessary. A time-varying L can be computed from 2 fixed matrices encoding 2 correlation types:

$$L_t = \gamma_t L_w + (1 - \gamma_t) L_b, \tag{5.5}$$

where L_w and L_b represent 2 different matrices and γ_t the blending coefficient. Based on it, the forward process is defined as:

$$\mathbf{x}_t = \alpha_t (L_t \boldsymbol{\epsilon}) + (1 - \alpha_t)\mathbf{x}_0 \tag{5.6}$$

With this forward process, Gaussian noise and Gaussian blue noise are interpolated based on the time step t. More generally, this model supports smooth interpolation of any two types of noises based on γ_t. Figure 5.4 shows an example of linear interpolation from Gaussian noise to Gaussian blue noise. The corresponding frequency power spectra, computed by the Discrete Fourier Transform, show that the energy in the low-frequency region is decreasing from left to right.

Algorithm 1. Pseudocode for `forward` method

```
1: function forward($L_w, L_b, \gamma_t$)        ← White and blue noise matrices $L$,
2:     $\mathbf{b} \leftarrow$ get_noise($L_w, L_b, \gamma_t$)      blending coefficient $\gamma_t$
3:     $\alpha_t \sim U(0,1)$
4:     $\mathbf{x}_0 \sim p_0$
5:     $\mathbf{x}_t \leftarrow \alpha_t \mathbf{b} + (1-\alpha_t)\mathbf{x}_0$      ← Eq. (5.6)
6:     return $\mathbf{x}_t$
```

Algorithm 2. Pseudocode for `backward` method

```
1: function backward( )
2:     $\mathbf{x} \sim N(\mathbf{0}, \boldsymbol{I})$
3:     for $t \leftarrow T$ to 1 do
4:         $\alpha_t \leftarrow$ get_alpha($t$)              } User defined
5:         $\alpha_{t-1} \leftarrow$ get_alpha($t-1$)        } $\alpha$-scheduler
6:         $\gamma_t \leftarrow$ get_gamma($t$)                } Eq. (5.9)
7:         $\gamma_{t-1} \leftarrow$ get_gamma($t-1$)          }
8:         $\mathbf{x} \leftarrow \mathbf{x} + (\alpha_t - \alpha_{t-1}) f'_{\boldsymbol{\theta}}(\mathbf{x}, t)$
9:             $+(\gamma_t - \gamma_{t-1}) f''_{\boldsymbol{\theta}}(\mathbf{x}, t)$      ← Eq. (5.7)
10:    return $\mathbf{x}$
```

Next, the forward process needs to be inverted to define the backward process. Based on the definitions of the L and the forward process, we can derive the backward step as follows:

$$\mathbf{x}_{t-1} = \mathbf{x}_t + (\alpha_t - \alpha_{t-1})(\mathbf{x}_0 - L_t\boldsymbol{\epsilon}) + (\gamma_t - \gamma_{t-1})\alpha_{t-1}(L_b\boldsymbol{\epsilon} - L_w\boldsymbol{\epsilon}) \tag{5.7}$$

Detailed derivations can be found in Appendix C.1. Here, L_w is an identity matrix representing Gaussian (white) noise, L_b is the matrix defined in Eq. (5.4). When $L_b = L_w$, our model falls back to IADB. When $L_b \neq L_w$, we obtain a more general model with time-varying noises.

In IADB, the network is designed to learn only the term $\mathbf{x}_0 - L_t\boldsymbol{\epsilon}$, where L_t is simply an identity matrix in their case. Here we can train the network to learn both terms in Eq. (5.7). A brute force way to achieve this would be using two neural networks. However, this is not practical as it will introduce significantly more computation than IADB. We choose to output a 6-channel image, representing the two terms as two 3-channel images, noted as $f'_{\boldsymbol{\theta}}$ ($\mathbf{x}_t$, t) and $f''_{\boldsymbol{\theta}}$ ($\mathbf{x}_t$, t), respectively. The desired network output becomes $\mathbf{x}_0 - L_t\boldsymbol{\epsilon}$ and $\alpha_{t-1}(L_b\boldsymbol{\epsilon} - L_w\boldsymbol{\epsilon})$. Therefore, the loss function becomes:

$$\begin{aligned} L_{Ours} = \sum_t &((f'_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - (\mathbf{x}_0 - L_t\boldsymbol{\epsilon}))^2 \\ &+ \frac{\gamma_t - \gamma_{t-1}}{\alpha_t - \alpha_{t-1}}(f''_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \alpha_{t-1}(L_b\boldsymbol{\epsilon} - L_w\boldsymbol{\epsilon}))^2)) \end{aligned} \tag{5.8}$$

Note that though our model is trained with time-varying noises, it is still deterministic during the backward process. The backward process starts with an initial Gaussian noise and no additional noise is required in the intermediate time steps. Instead, the network learns to guide the backward process in a time-varying denoising manner.

The procedures of forward, backward, and noise generation are summarized in Algorithms 1 to 3. In Algorithm 2, we consider `get_alpha` (α-scheduler) as a linear function ($\alpha_t = t/T$) following Heitz et al. [2023], but it can be non-linear functions as well. Next,

Algorithm 3. Pseudocode for time-varying `get_noise` method

```
1: function get_noise($L_w, L_b, \gamma_t$)
2:     $\epsilon \sim N(\mathbf{0}, \boldsymbol{I})$
3:     $L_t \leftarrow \gamma_t L_w + (1 - \gamma_t) L_b$          ← Eq. (5.5)
4:     $\mathbf{b} \leftarrow L_t \epsilon$          ← Eq. (5.4)
5:     return b
```

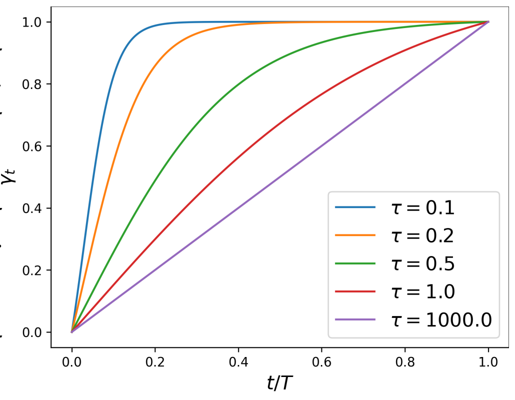
we define `get_gamma` as a general sigmoid-based function in Eq. (5.9). The weighted term $(\gamma_t - \gamma_{t-1})/(\alpha_t - \alpha_{t-1})$ in Eq. (5.8) automatically accounts for the difference between α-schedulers and γ-scheduler. When $\gamma_t - \gamma_{t-1}$ is small, the contribution of $f''_{\boldsymbol{\theta}}$ ($\mathbf{x}_t$, t) decreases. This is consistent to the backward process described in Eq. (5.7), where $f''_{\boldsymbol{\theta}}$ ($\mathbf{x}_t$, t) is less important when $\gamma_t - \gamma_{t-1}$ is small.

Noise scheduler. Inspired by the study from Chen [2023], the scheduler choice has an important impact, in particular with increasing image resolution. We parameterize `get_gamma`, the γ-scheduler, as a sigmoid-based function to control the interpolation between two noises. More specifically, the γ-scheduler is parameterized by 3 parameters: $start, end, \tau$ according to Chen [2023]:

$$sigmoid\left(\frac{start + (end - start) * t/T}{\tau}\right) \tag{5.9}$$

where $sigmoid(x) = 1/(1 + e^{-x})$ and t is the time step.

Since it is not known how to set $start$, end, and τ in advance, we consider optimizing them in addition to the network parameters where $start \in [-3, 0)$, $end \in (0, 3]$, $\tau \in [0.01, 1000.0]$. During the initial experiments, we found that $start$ and end converged stably to around 0 and 3, while τ converged to around 0.2 or kept increasing, depending on the image resolution. Meanwhile, we found that optimizing these 3 parameters took extra epochs to converge and made the training of the network more difficult, due to their changes over epochs.



To make the choice of the 3 parameters more practical, we choose to fix $start = 0, end = 3$ and set τ based on the image resolution: $\tau = 0.2$ for 128^2, $\tau = 1000$ for 64^2 images. The curves of the γ-scheduler with different τ values are shown in the inset image. We summarize the values of the 3 parameters we use in Appendix C.2 for all experiments.

Discussion. Our time-varying noise model provides more capacity to choose a data-dependent scheduler for γ_t to improve the denoising process. One potential issue is that we need extra epochs to search for optimal parameters for the γ-scheduler. To alleviate this problem, we propose a practical solution, which is to fix τ based on our initial optimization. But how to choose the γ-scheduler in a more efficient way requires more study in the future.

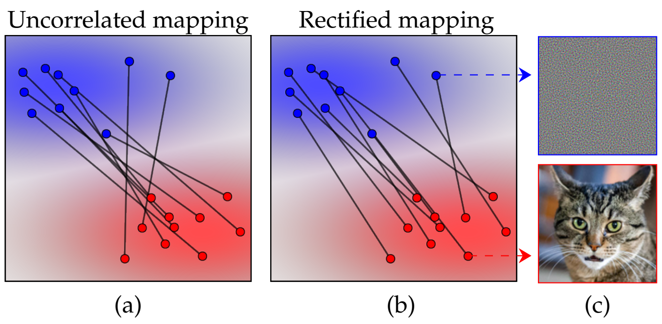


Figure 5.5: *Visualization of the impact of rectified mapping on mini-batch noise-mapping pairing. Blue and red points, respectively, represent the randomly selected noise and target image sampled in a given mini-batch from the underlying blue and red distributions. Standard practice (a) consists of a random mapping between the noise and target images. Our rectified mapping (b) improved it by reducing the distance between the data pair. One example of noise and target from the mini-batch is visible (c). This example has been generated using our noise mask and mapping algorithm.*

5.1.5 Data sample correlation using rectified mapping

The previous paragraphs have demonstrated the use of correlated noise across pixels to enhance the diffusion process. Correlation can also be employed within a single mini-batch to improve the mapping between noise and the target image.

Inspired by Rectified flow [Liu et al., 2023a] and Instaflow [Liu et al., 2023b], correlation can be utilized to rectify the paired noise-image. Figure 5.5 visualizes a single mini-batch of paired data sample $\mathbf{x}_0$ (red distribution) and noise $\mathbf{b}$ (blue distribution) during a training iteration, and our rectified mapping. Previous work (Fig. 5.5(a)) applies a random mapping between $\mathbf{x}_0$ and $\mathbf{b}$. The noise-data mapping can be improved by applying an in-context stratification before feeding it into the forward process (Fig. 5.5(b)). This rectified mapping reduces the distance between each noise and its target image, resulting in a more direct trajectory. To find the mapping, we compute the squared distance between noises and images at the individual pixel level using the L2 norm. Then, for each $\mathbf{b}$, the $\mathbf{x}_0$ with the shortest distance that has not yet been used is selected. This improved mapping ensures that a specific image will consistently be associated with the same type of noise during the training process, resulting in a smooth gradient flow across time steps.

5.1.6 Experiments

Implementation details

We use CelebA [Lee et al., 2020], AFHQ-Cat [Choi et al., 2020], and LSUN-Bedroom [Yu et al., 2015b] datasets, with different resolutions, for unconditional/conditional image generation. More details on the experimental setup can be found in Appendix C.2.

Our framework is implemented in Pytorch [Paszke et al., 2017] based on the official implementations from Song et al. [2021b]; Heitz et al. [2023]. We use 2D U-Net [Ronneberger et al., 2015] implemented in diffusers library [von Platen et al., 2022]. More details about the network architecture and training details, including the values of τ in Eq. (5.9), can be found in Appendix C.2. Regarding the hyperparameters in diffusion models, we use $T = 1000$ for training and $T = 250$ for testing. To optimize the network parameters, we use the AdamW optimizer [Loshchilov and Hutter, 2017] with a learning rate of 0.0001. We use 4 NVIDIA Quadro RTX 8000 (48 GB) GPUs to train and test on all datasets.

For evaluation, we use FID [Heusel et al., 2017], Precision and Recall [Kynkäänniemi et al., 2019] to measure the generative quality of all models. The metrics are computed using the implementation from Stein et al. [2024], with the Inception-v3 network [Szegedy et al., 2016] as backbone. We generate 30k images to compute FID, Precision, and Recall for all datasets.

Image generation

We compare our method with two existing deterministic diffusion models DDPM [Ho et al., 2020], DDIM [Song et al., 2021b], and IADB [Heitz et al., 2023] on unconditional image generation. To ensure equitable comparisons, we employ identical initial Gaussian noise across all methods throughout the generative process. Note that DDIM is trained using the diffusers library [von Platen et al., 2022] with the same training setup compared to IADB and Ours. We also compare with stochastic diffusion models DDPM [Ho et al., 2020] and IHDM [Rissanen et al., 2023] for completeness, using the same number of time steps. Our method shows consistent improvement over the two methods.

The results on AFHQ-Cat (64^2), LSUN-Church (64^2), and Celeba (64^2) at the same resolution (64^2) are shown in Figure 5.6. Our method exhibits the blue noise effect starting from approximately $t = 75$, which visually distinguishes it from other methods. In terms of the generated images at time step $t = 0$, our method produces images with less distortion around the pillars of the building and more detailed content around the windows and doors. In addition to visual comparisons, the quantitative evaluations presented in Table C.7 demonstrate consistent improvements of our method over IADB and DDIM for datasets with a resolution of 64^2.

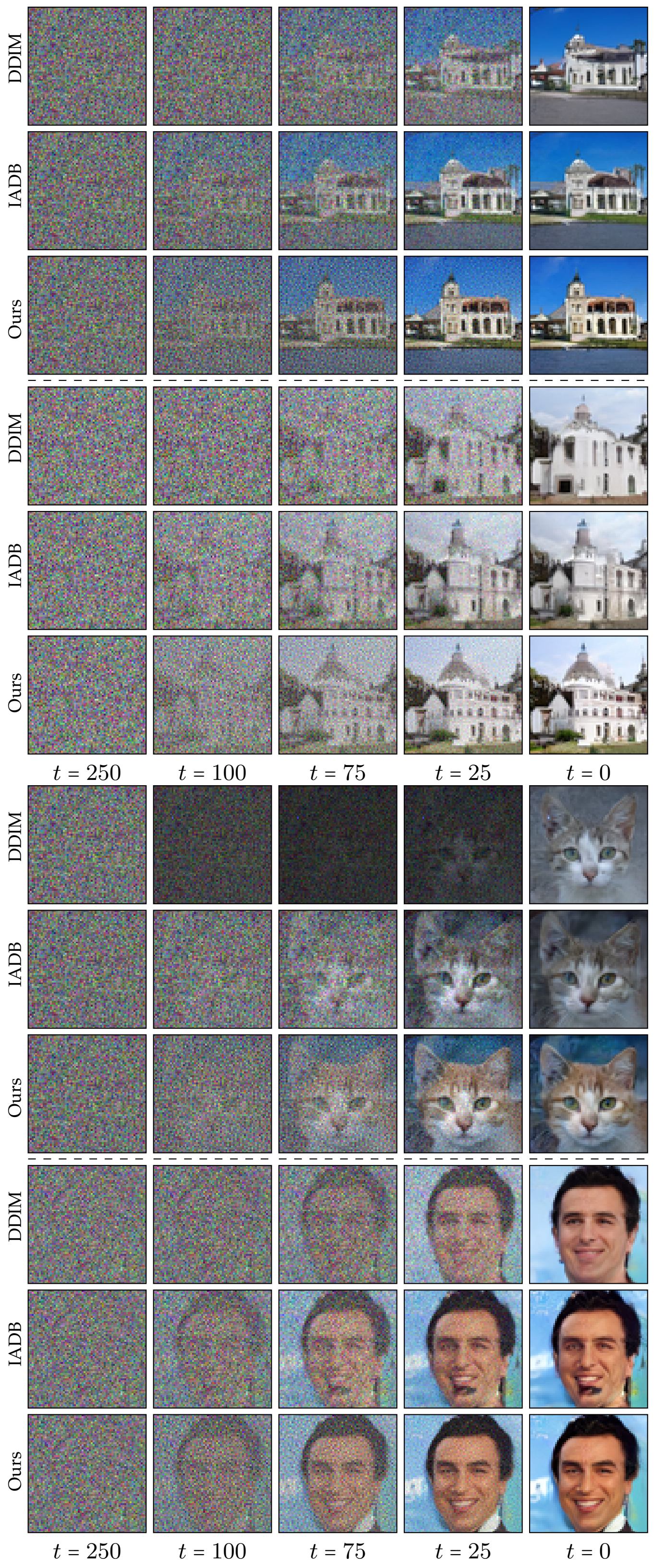


Figure 5.6: *Comparisons of image generation using DDIM, IADB, and Ours trained on LSUN-Church* (64^2)*, AFHQ-Cat* (64^2)*, and CelebA* (64^2) *datasets. For each example, all methods start the diffusion from the same noise. In all cases, our method achieves the highest quality result with more realistic images. Quality in detail generation can be seen in the windows and doors of the buildings. By looking at the noise at different time steps, the evolution from random to blue noise is visible for our method.*

Table 5.1: *Quantitative FID score comparisons among IHDM [Rissanen et al., 2023], DDPM [Ho et al., 2020], DDIM [Song et al., 2021b], IADB [Heitz et al., 2023], and our method across diverse datasets. Notably, our approach exhibits improvements over IADB on every evaluated dataset. While our method is outperformed by DDIM on only one dataset, it's worth noting that IADB also performs poorly on the same dataset. Additional metrics are provided in Appendix C.3.*

FID (↓)	IHDM	DDPM	DDIM	IADB	Ours
AFHQ-Cat (64^2)	11.02	9.75	9.82	9.19	**7.95**
AFHQ-Cat (128^2)	15.40	12.41	10.73	10.81	**9.47**
CelebA (64^2)	14.30	8.56	9.26	7.53	**7.05**
CelebA (128^2)	36.93	15.06	**11.92**	20.71	16.38
LSUN-Church (64^2)	17.76	13.07	16.46	13.12	**10.16**

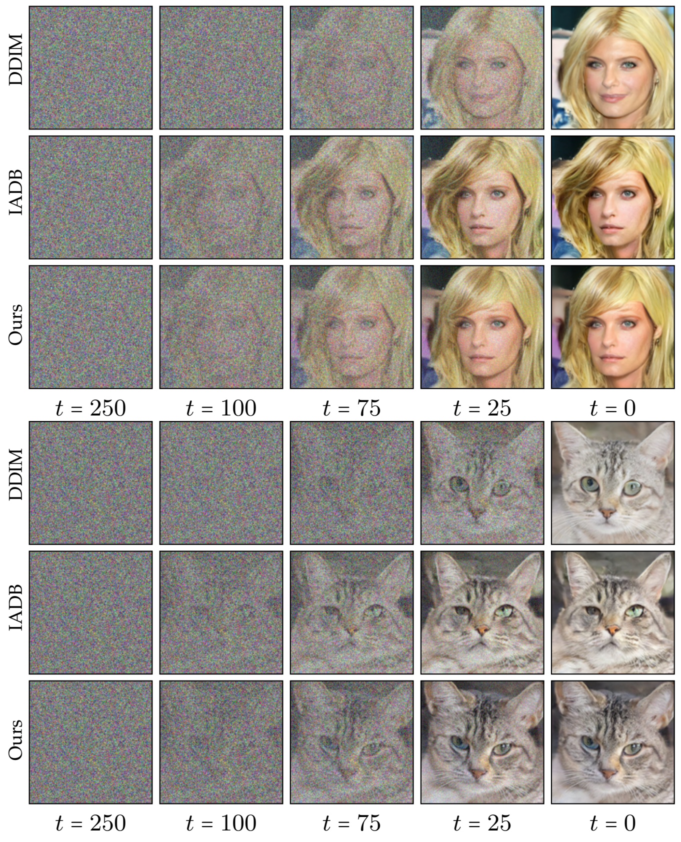


Figure 5.7: *Image generation comparisons between DDIM, IADB, and Ours trained on CelebA (128^2) and AFHQ-Cat (128^2) datasets, respectively. All methods start with the same initial Gaussian noise during the backward process. Our method generates more realistic content around the hair, eye, and mouth regions compared to IADB. Compared to DDIM, our method achieves similar visual quality. The impact of time-varying noise (we use $\tau = 0.2$ in Eq. (5.9)) can be seen by comparing IADB and ours starting from around $t = 75$.*

For higher-resolution results, we observe the difference in the generated content starting at approximately $t = 75$, as depicted in Fig. 5.7. Towards the end of the backward process, around $t = 25$, we begin to notice the emergence of the blue noise effect as we use

$\tau = 0.2$ (Eq. (5.9)). To examine the details more closely, please zoom in. Additionally, we offer a Supplemental HTML viewer where the intermediate generated images can be interactively visualized at various time steps during the backward process. In terms of realism, our generated images exhibit improved quality in regions such as hair, mouth, and eyes compared to IADB, as shown in Fig. 5.7. When compared to DDIM, our method achieves similar visual quality. Quantitative results on the CelebA (128^2) dataset, as presented in Table C.7, demonstrate that DDIM outperforms IADB and our method. This outcome is attributed to DDIM employing a different expression for α_t in Eq. (5.1), as discussed by Heitz et al. [2023]. Depending on the dataset, DDIM may outperform IADB due to the distinct choice of α_t. However, this is not a limitation for either IADB or our method.

Additionally, our framework can also consider rectified mapping across images during training. Table 5.2 provides the FID score with and without data correlation with respect to the number of diffusion steps, tested on AFHQ-Cat (64^2). Rectified mapping in the mini-batch achieves a lower FID when the number of steps remains low, but a slightly higher FID when increasing the step count.

We provide additional results in Appendix C.3, including detailed timing of the Gaussian blue noise generation and the backward process, and a nearest neighbors test to confirm that our method does not overfit the training data.

Extension to other diffusion models. Our method can be extended to DDIM, supported by derivations and preliminary results in Appendix C.1 and Appendix C.3. Also, our method can be incorporated into LDM [Rombach et al., 2022] for high-res image generation. As shown in Fig. 5.8, ours generates more realistic eyes and has better FID (**11.45** $<$ 12.19) compared with IADB on AFHQ-Cat (512^2). Nevertheless, developing a new framework based on other models necessitates additional effort, which we defer to future work.

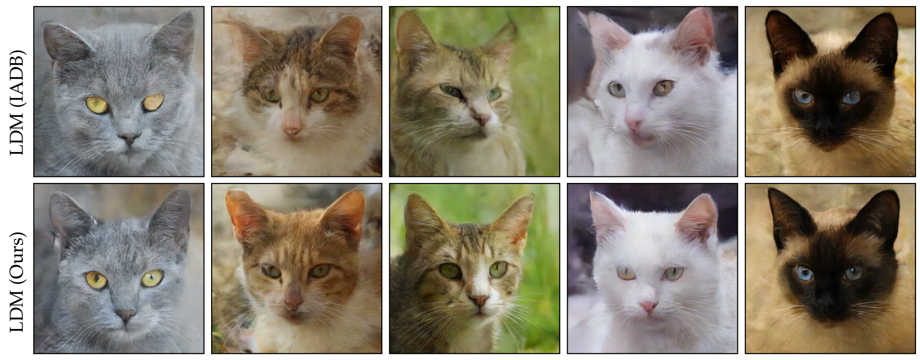


Figure 5.8: *Latent diffusion model (LDM) based high-res image generation using IADB and Ours on AFHQ-Cat (512^2). IADB introduces more artifacts around the eye regions and has worse FID (12.19 $>$ **11.45**) compared with Ours.*

Conditional image generation

Besides unconditional generation from noise, our model also works for conditional image generation, such as image super-resolution, by simply concatenating the conditional

Table 5.2: *Comparing the impact of rectified mapping during training on AFHQ-Cat (64^2). FID scores (↓) are provided with and without rectified mapping across different step counts. Correlation in the mini-batch results in lower FID at low steps but higher during slow diffusion.*

Diffusion step count (t)	1	2	4	16	128	250
Uncorrelated batch mapping	402.4	330.5	130.8	14.3	**7.9**	**7.95**
Correlated batch mapping	**397.0**	**321.8**	**118.3**	**12.4**	8.0	8.2

low-resolution image with the noisy image as input.

Figure 5.9 shows comparisons between IADB and Ours for image super-resolution on the LSUN-Church dataset from resolution 32^2 to 128^2. Our method outperforms IADB quantitatively according to SSIM [Wang et al., 2004], PSNR, and mean squared error (MSE). Our method outperforms IADB in terms of fidelity to the reference, as evidenced by its lower MSE. Visually, IADB tends to introduce excessive details, particularly in the bottom portion of the first image. Our method also effectively preserves straight lines throughout the image. The quantitative results of all image super-resolution experiments can be found in Appendix C.3, showing that our method consistently outperforms IADB.

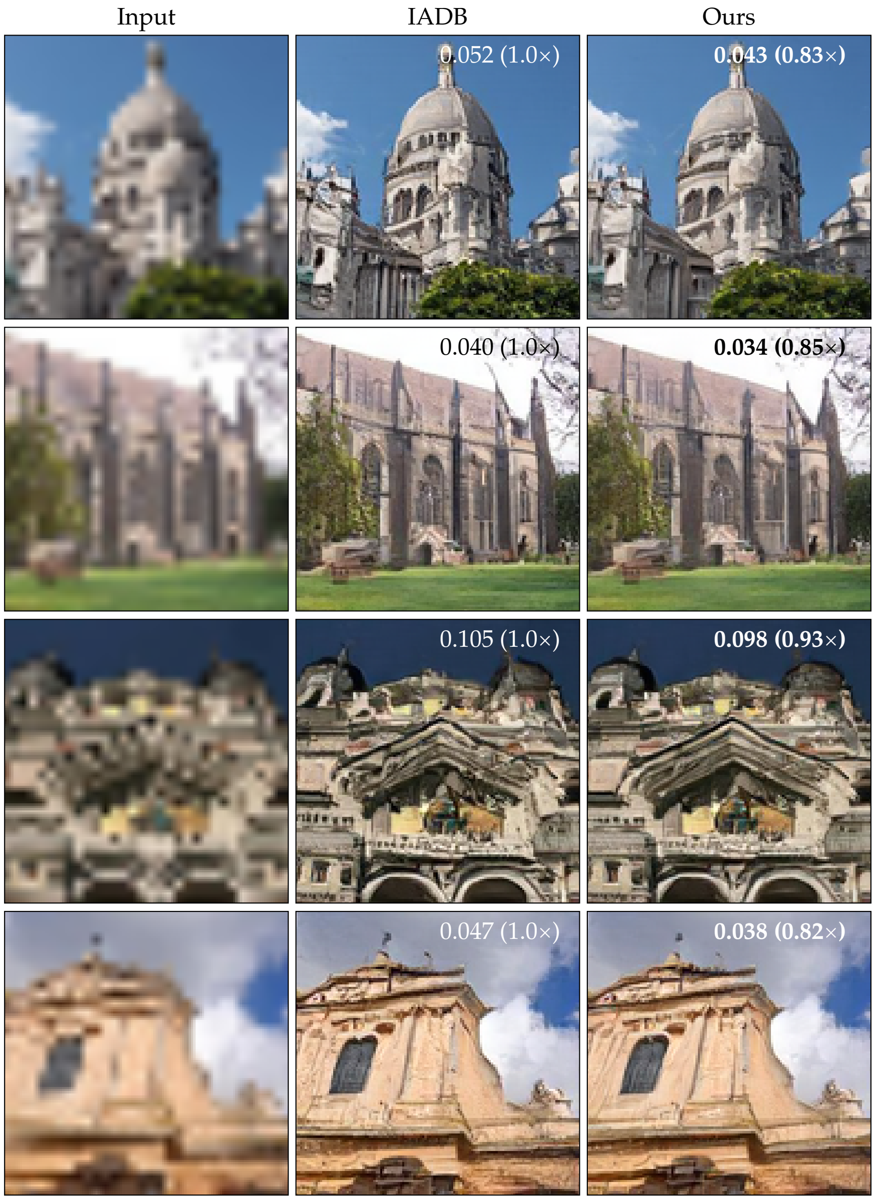


Figure 5.9: *Image super-resolution comparisons between IADB (SSIM/PSNR=0.57/19.46) and Ours (SSIM/PSNR=**0.59/20.00**) on LSUN-Church ($32^2 \rightarrow 128^2$). The mean squared error w.r.t the reference is visible in the upper corner with the relative error to IADB. Our method achieves lower error and more plausible details with less hallucination.*

Ablation study and analysis

Combinations of noises. To confirm that Gaussian blue noise works due to its high-frequency property, we replace Gaussian blue noise by Gaussian red noise, a low-frequency noise visualized in Fig. 5.10.

Red noises are generated using the same method [Ulichney, 1993] by simply maximizing the objective function instead of minimizing it. Then we compute the corresponding

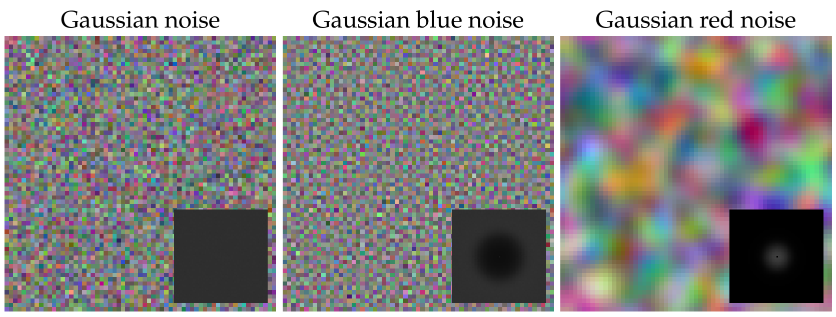


Figure 5.10: *Visualization of Gaussian noise, Gaussian blue noise, and Gaussian red noise at resolution* 64^2*. The power spectrum is shown in the bottom right corner. While Gaussian blue noise shows mostly high frequency variations, Gaussian red noise shows only low frequencies.*

covariance matrix and lower triangular matrix so that we can generate Gaussian red noise for our framework. As shown in Fig. 5.11, using Gaussian red noise in our framework failed to recover the fine details due to its low-frequency property. Table 5.3 shows that replacing Gaussian blue noise by Gaussian red noise dramatically drops the Precision, while the Recall is comparable. This is consistent with the visual observations in Fig. 5.11 as Precision mainly measures the realism of the generated images.

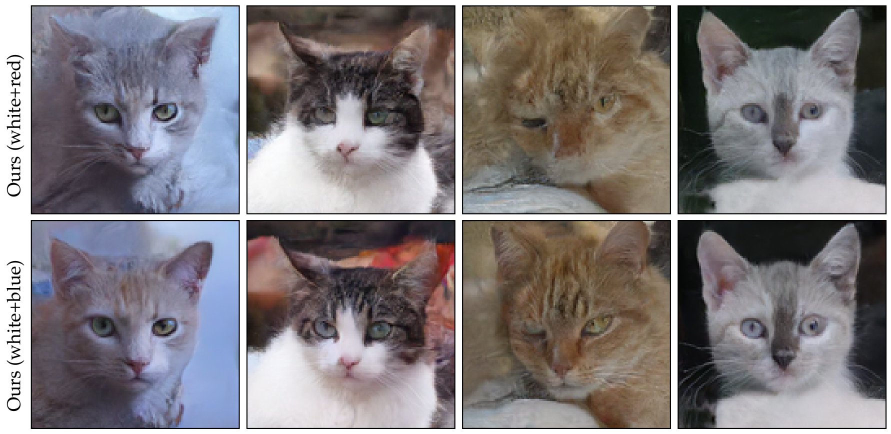


Figure 5.11: *Qualitative image generation comparisons between Gaussian red noise and Gaussian blue noise using our method on AFHQ-Cat* (128^2)*. Our method with Gaussian blue noise creates more high-frequency details, while using Gaussian red noise introduces visible artifacts.*

Another option is to use only Gaussian blue noise, which has been shown in Fig. 5.1 (second row). The final generated images are less realistic compared to IADB and Ours. The visual quality is also consistent with the quantitative metrics as shown in Table 5.3. But it is worth mentioning that in the case of using only Gaussian blue noise, we can observe that the content of the image appears faster and cleaner at early time steps compared to other choices, as shown in Fig. 5.1. We performed an additional experiment called early stopping. This is to show that if we stop at early steps, using only Gaussian blue noise gives better results than using only Gaussian noise. As shown in appendix Fig. C.3, the results of using only Gaussian blue noise (second row) stopped at $t = 200$ have sharper details than using only Gaussian noise (first row). Quantitative evaluation of early stopping can be found in Appendix C.3. However, as blue noise has

Table 5.3: *Ablation study on different combinations of noises using our framework on AFHQ-Cat (128^2). The last two rows mean blending Gaussian noise with Gaussian red or blue noise using the γ-scheduler with $\tau = 0.2$.*

Noise model	FID (↓)	Precision (↑)	Recall (↑)
Ours (Gaussian white noise only)	10.81	**0.78**	0.31
Ours (Gaussian blue noise only)	17.61	0.59	0.18
Ours (Gaussian white+red noise)	13.64	0.67	**0.34**
Ours (Gaussian white+blue noise)	**9.47**	**0.78**	**0.34**

no energy in the low-frequency region, it restricts the diffusion process to a limited range of directions. For this reason, it becomes difficult to refine the intermediate generated content in later time steps and thus results in worse quality, as shown in Table 5.3. Instead, our method takes blue noise into account from middle or later time steps, when low-frequency components are already visible, and the network is more focused on refining high-frequency details.

Diffusion with different noise magnitude. Since using only Gaussian blue noise at all time steps leads to degraded quality, we further conduct an analysis on diffusion at later time steps by explicitly ignoring the early time steps. We compare Gaussian noise and Gaussian blue noise by training a diffusion model (IADB) up to some time steps with a certain noise magnitude (e.g., 30%). During the testing phase, ground truth images are provided so we are able to compare the denoised images with the ground truth ones. Running with 100% of noise, the experiment would fall back to the standard diffusion generative process. Based on Fig. 5.12, by using Gaussian blue noise, we generate more detail- and content-preserving images under various noise magnitudes, compared to the ones using Gaussian noise. This indicates that Gaussian blue noise is suitable for denoising when low-frequency components become visible. This is consistent with our idea of blending Gaussian blue noise from middle or later time steps.

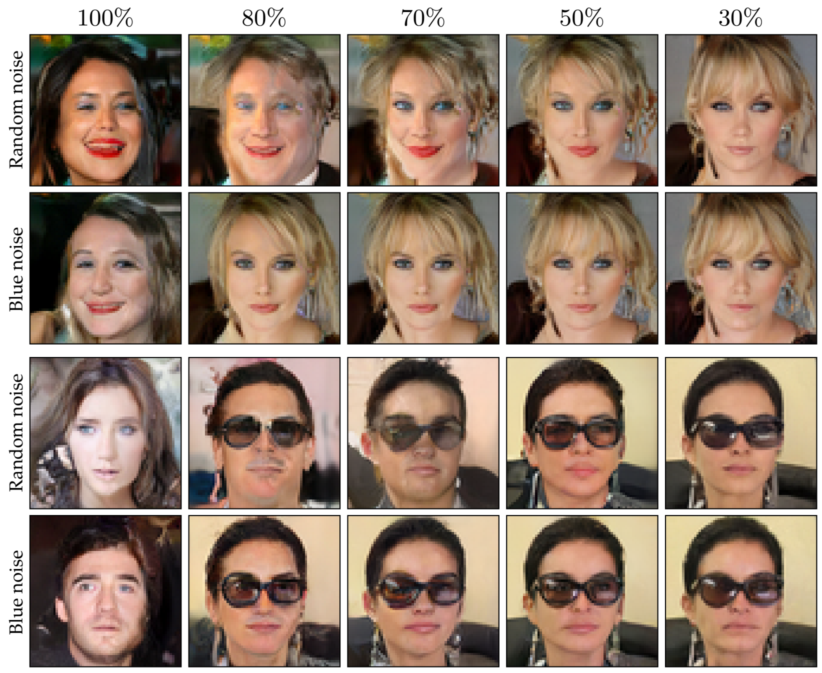


Figure 5.12: *Evaluating the impact of noise magnitude on detail enhancement. Our Gaussian blue noise method better preserves fine details even with increased noise magnitude, while maintaining the integrity of the content. With* 100% *noise, both models fall back to the full generative process.*

More ablations. We further compare different γ values and a cosine-based scheduler [Nichol and Dhariwal, 2021] used for the γ-scheduler. Also, we compare different Gaussian blue noise mask sizes used for padding/tiling. More details can be found in Appendix C.3.

5.1.7 Conclusion

We have presented a new method for incorporating correlated noise into deterministic generative diffusion models. Our technique involves using a combination of uncorrelated and correlated noise masks generated using matrix-based methods. By investigating different noise correlations, we have uncovered the intricate relationship between noise characteristics and the quality of generated images. Our findings indicate that high-frequency noise is effective at preserving details but struggles with generating low-frequency components, whereas low-frequency noise hinders the generation of complex details. To achieve optimal image quality, we propose selectively using different types of noise in a time-dependent manner, leveraging the strengths of each noise component. To validate the effectiveness of our approach, we conducted extensive experiments using it in conjunction with the well-known method IADB [Heitz et al., 2023]. By keeping the training data and optimization hyperparameters consistent, we consistently observed significant improvements in image quality across various datasets. These results demonstrate the superiority of our approach in enhancing the image generation capabilities of deterministic diffusion models.

Limitations. Currently, parameter tuning for our γ-scheduler depends on the image resolution. It is also computationally intensive to compute Gaussian blue noise masks while extending our approach to higher resolution models. All data from a specific mini-batch must be on a single GPU for our rectified mapping to function. Further research is needed to expand this to distributed training involving inter-GPU synchronization.

Future work. We believe our proposed model will inspire new research directions in designing noise patterns for improving the efficiency of generative diffusion models. An interesting future work would be extending our model to interpolate more than two noises to take into account more different types of noises, such as low-pass and band-pass noises. This may provide a greater degree of freedom to improve the training and sampling efficiency of the diffusion models. Further, we can design more advanced techniques to correlate data samples during training, which is orthogonal to using correlated noise. Extending our framework (e.g., the time-varying noise model) to stochastic models [Ho et al., 2020; Song et al., 2021a] and even fewer-step models [Karras et al., 2022; Song et al., 2023; Luo et al., 2023] would be another interesting future direction. In this way, our framework can be generalized to state-of-the-art denoising diffusion models.

In terms of applications, we tested our model on 2D unconditional and conditional image generation. Interesting future work would include generalizing our model to synthesize other data representations, such as video and 3D mesh.

5.2 Diffusion stereo video generation and restoration

Besides designing noise for the diffusion process, we also observe that noise plays an important role in diffusion-based restoration tasks. In particular, we investigate its impact in stereo video generation, a highly challenging problem due to the strict requirements on temporal and view consistency, especially under VR viewing conditions, where visual artifacts are much more perceptible to human observers.

Stereo video generation has been gaining increasing attention with recent advancements in video diffusion models. However, most existing methods focus on generating 3D stereoscopic videos from monocular 2D videos. These approaches typically assume that the input monocular video is of high quality, making the task primarily about inpainting occluded regions in the warped video while preserving disoccluded areas. In this section, we introduce a novel pipeline that not only generates stereo videos but also enhances both left-view and right-view videos consistently with a single model. Our approach achieves this by training on degraded data for restoration, as well as conditioning on the warped mask for consistent stereo generation. As a result, our method can be trained on synthetic stereo video datasets and applied to low-quality real-world videos, performing both stereo video generation and restoration. Experiments demonstrate that our method outperforms existing approaches both qualitatively and quantitatively in stereo video generation from low-resolution inputs.

5.2.1 Introduction

Stereo video generation is becoming increasingly crucial for creating immersive experiences in modern VR applications. Recently, diffusion models have been showing great potential for realistic video generation from text prompts. However, most of the existing models focus on generating monocular videos, while stereo video generation remains underexplored.

Training new diffusion models for stereo videos from scratch can be expensive and time-consuming. There has been recent progress on stereo video generation using training-free strategies by leveraging pretrained models such as Stable Diffusion [Rombach et al., 2022] and ModelScope/Zeroscope [Wang et al., 2023]. Existing training-free camera control methods (e.g., Hou et al. [2024]) are unsuitable for fine-grained stereo video generation. Shi et al. [2024b] propose zero-shot stereo video generation with noisy restart. Similar works include SVG from Dai et al. [2024], StereoDiffusion from Wang et al. [2024] and T-SVG from Jin et al. [2024]. These training-free methods do not require training on large datasets, making them lightweight and easy to use. However, these methods are limited to some extent as they only rely on the pretrained diffusion models to inpaint the unknown region of the right-view image in the latent space. Since the pretrained diffusion models are not specifically trained on stereo data, the inpainting can be spatially and temporally inaccurate. After decoding the inpainted latent representation, the generated right-view images are not guaranteed to be consistent with the left-view image. This may also lead to artifacts around the occluded region in the right-view images.

On the other hand, training-based methods such as StereoCrafter [Zhao et al., 2024], SpatialMe [Zhang et al., 2024], StereoConversion [Mehl et al., 2024] and ImmersePro [Shi et al., 2024a] have shown promising results in stereo video generation. These methods are designed to train on a large-scale dataset of stereo videos from the internet or the movie industry, and have become foundation models in stereo video generation. While these datasets are diverse with indoor and outdoor scenes, they are expensive to capture and annotate for individuals and have not been publicly available. A bigger issue for these methods is that they are trained to inpaint the left warped videos to generate the right videos by construction. These models assume the input video is high-quality and learn to maintain details from the input view and inpaint the occluded regions naturally. But when the input video is low-quality, their models still keep the details from the input video, making the generated stereo video low-quality and cannot manage to enhance the video at the same time.

In this work, we suggest a novel idea for simultaneous stereo video generation and restoration. To achieve this, we propose to use data degradation (e.g., noise addition, blurring, compression) during training, inspired by Real-ESRGAN [Wang et al., 2021]. We design a consistent training and inference pipeline to learn the generation of both left and right views with restoration. In this way, we can not only enhance the input left-view video, but also generate a right-view video consistent with the left-view one. Further, this pipeline can be trained via a synthetic dataset that is relatively easier to generate compared to real-world data. Synthetic data comes with ground truth information like depth, which removes the complicated preprocessing pipeline, including stereo matching, depth estimation, data filtering, etc, as proposed in StereoCrafter [Zhao et al., 2024].

In summary, we make the following contributions:

- a consistent training and inference pipeline for both left and right views conditioned on warped masks,
- a process of data augmentation with image degradations to train the model for robust stereo video generation and restoration simultaneously,
- a novel pipeline for stereo video generation and restoration using synthetic data (both the pipeline and data will be publicly available upon acceptance).

5.2.2 Stereo video generation and restoration

Overview

Given an input left-view video represented by $\boldsymbol{x}^{(l)}$ with F =16 frames, our goal is to generate both the left-view video $\hat{\boldsymbol{x}}^{(l)}$ and right-view video $\hat{\boldsymbol{x}}^{(r)}$. Different from previous methods [Wang et al., 2024; Zhao et al., 2024], we target stereo video generation and restoration where input video can be low-resolution and enhancement is required for the left-view as well. Therefore, our pipeline consists of training both left-to-right and left-to-left generation branches, as shown in Fig. 5.13. We fine-tune a single video diffusion model for inpainting and restoration during left-to-right generation, as well as restoration during left-to-left generation.

Stereo video data generation

To train such a video diffusion inpaint model, we first need to generate training data. We use Kubric [Greff et al., 2022], a Blender [Blender Foundation, 2023] based graphics data generation pipeline, to generate synthetic training data in a fully controllable manner. This allows us to create 3D scenes with customized objects, lighting, and camera positions, provided with ground truth data like depth maps. We rely on the Kubric preprocessed ShapeNet dataset [Chang et al., 2015] and environment map [Hold-Geoffroy et al., 2019]

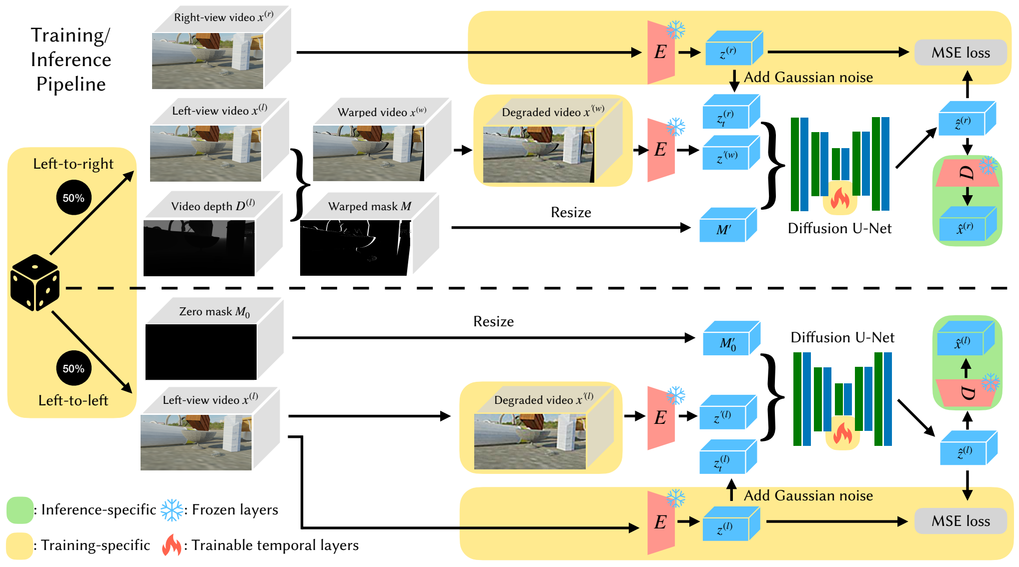


Figure 5.13: *Training and inference pipeline of our method. We fine-tune the Diffusion U-Net for both left-to-right and left-to-left generation and restoration branches. During training, we randomly sample a branch, where left-to-right requires depth maps, forward warping, and the right-view target video. For left-to-left, no warping is required, and we use a zero mask as the condition and left-view video as the target. Both branches require data augmentation/degradation during training. During inference, we run both branches as well as the decoder to generate videos for both views, without using the yellow boxes. Note that the two U-Nets share the same weights and CLIP [Radford et al., 2021] features of $z'^{(w)}$ and $z'^{(l)}$ are also part of the conditional input to the U-Net, omitted for simplicity. Details are discussed in Sec. 5.2.2.*

to create realistic scenes. More specifically, we use a subset of ShapeNet with 14 classes. For the environment map, we have 458 environment maps preprocessed by Kubric for training. For each class of object and environment map, we split the dataset to have non-overlapping training and test sets. For each scene, we generate a left-view video and a right-view video using 2 cameras. We set the camera baseline to be sampled from a normal distribution with a mean of 65mm and a standard deviation of 1mm. Camera position is randomly sampled on spheres with different radii, and the left camera's look-at position is the origin. The video is generated by moving the objects forward for 21 frames and removing the first 5 frames with the remaining $F = 16$ frames. We show training examples in Fig. 5.13 and in appendix Fig. D.1.

Data augmentation. We further augment the dataset by degrading images with a set of blurring, downsampling, adding Gaussian noise, and JPEG compression operations, for training the restoration model. This is inspired by Real-ESRGAN [Wang et al., 2021] to train a generator for image restoration by degrading synthetic images. To make sure the data degradation is temporally-consistent, we use the implementation of Wang et al. [2021] while fixing the random seeds during augmentation for all image frames in each video. We show that data augmentation is crucial for simultaneous generation and restoration in Fig. 5.14. Our generated right-view video with augmentation contains sharper details around the edges than the one without augmentation.

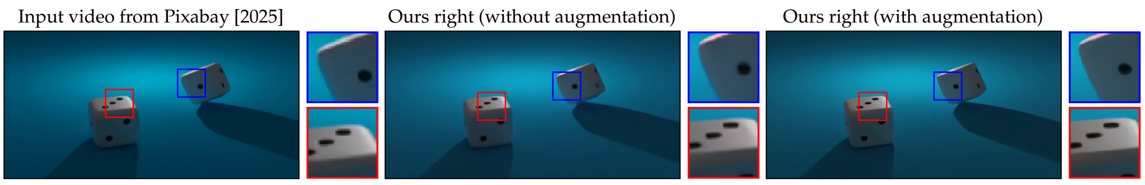

Input video from Pixabay [2025]
Ours right (without augmentation)
Ours right (with augmentation)

Figure 5.14: *Data augmentation with degradations is the key to restoration, given low-resolution input. Our right-view output with augmentation contains sharper details around the edges than the one without augmentation. Input is from Pixabay [2025] degraded to* 320×160 *following Eq.* (5.10).

Discussion. There exist other large-scale object datasets like Objaverse [Deitke et al., 2023], but they are not preprocessed to be used in Kubric. Other stereo video datasets like KITTI [Menze and Geiger, 2015] and DrivingStereo [Yang et al., 2019] are limited to self-driving scenarios, while the Sintel [Butler et al., 2012] dataset is limited in terms of the number of objects and videos. The StereoCrafter data dataset is not publicly available and requires a lot of resources to capture the data. Therefore, we believe our dataset will be useful for fine-tuning video diffusion models for stereo video generation in a wide range of scenarios.

Video warping

We mainly follow the pipeline of StereoDiffusion [Wang et al., 2024] for disparity-based forward warping. The forward warping function F performs disparity-based image warping given the left-view video and the depth maps as input. First, the function computes pixel displacements using the inverse of the depth maps times a disparity scaling S. Each pixel displaces according to the scaled disparity to generate the forward warped video representing the right-view video. In some pixel locations of the right-view video, there might be overlapping values, and we maintain the pixel value based on the minimum depth utilizing a z-buffer. For pixels on the right-view that are not assigned any values, we apply bilinear interpolation following StereoCrafter [Zhao et al., 2024] to remove those flying pixels.

During training, the ground truth depth map is given by the graphics engine, while during inference, we use DepthCrafter [Hu et al., 2024] as the video depth estimator. According to the depth estimation quality, the warped mask might have unwanted artifacts. We further postprocess the warping mask using morphological dilation that can eliminate the flying pixels and holes within the occluded regions.

Training

Our training pipeline consists of both left-to-right and left-to-left generation and restoration. This is the key difference between previous methods and our proposed method. Note that in the following, we might use fine-tuning and training interchangeably to present the same meaning.

At each training iteration, we randomly sample a branch as shown in Fig. 5.13 left-most column. For the left-to-right branch, input consists of a left-view input video $\boldsymbol{x}^{(l)} \in \mathbb{R}^{F \times H \times W \times 3}$, a warped video $\boldsymbol{x}^{(w)} \in \mathbb{R}^{F \times H \times W \times 3}$ based on disparity-based forward mapping, and a binary warped mask $\boldsymbol{M}$ representing the occluded region after warping.

$\boldsymbol{x}^{(w)}$ is then degraded via our data augmentation to be $\boldsymbol{x}'^{(w)}$. Then $\boldsymbol{x}'^{(w)}$ is compressed via a frozen VAE encoder to latents $\boldsymbol{z}'^{(w)} \in \mathbb{R}^{F \times H' \times W' \times 4}$ with a factor of 8 ($H' = H/8, W' = W/8$). $\boldsymbol{M}$ is resized to $\boldsymbol{M}' \in \mathbb{R}^{F \times H' \times W' \times 1}$, with nearest-neighbor interpolation to ensure that the mask's binary nature is preserved.

As we are training a diffusion model, another input is the target video added with Gaussian noise, represented as $\boldsymbol{z}_t^{(r)}$, where t is a timestep of a diffusion forward process. $\boldsymbol{z}_t^{(r)}$, $\boldsymbol{M}'$ and $\boldsymbol{z}'^{(w)}$ are concatenated along the channel dimension, which serves as the input to the Diffusion U-Net [Blattmann et al., 2023; Ronneberger et al., 2015]. The output of Diffusion U-Net is $\hat{\boldsymbol{z}}^{(r)}$, which is used to compute the mean-square-error (MSE) with $\boldsymbol{z}^{(r)}$. The temporal layers of the U-Net are trainable to minimize the MSE loss.

For the left-to-left branch, input becomes the left-view video $\boldsymbol{z}^{(l)}$ without warping. The warped mask becomes a zero mask $\boldsymbol{M}_0$, indicating that there is no region for inpainting. After the same data degradation and VAE encoding processes, the input to the U-Net includes a resized warped mask $\boldsymbol{M}_0'$, a degraded left-view video latent $\boldsymbol{z}'^{(l)}$, and a noisy left-view latent $\boldsymbol{z}_t^{(l)}$. Similarly, we apply MSE loss between the output latent $\hat{\boldsymbol{z}}^{(l)}$ and the left-view video latent $\boldsymbol{z}^{(l)}$ to optimize the trainable temporal layers in the U-Net.

Inference

Consistent left and right view generation. After training, the model has learnt to maintain or enhance the details in the unmasked regions, as well as to inpaint the mask region, conditioned on the warped mask. Therefore, we can use $\boldsymbol{M}$ as an indicator for left-to-right generation and $\boldsymbol{M}_0$ for left-to-left generation. During inference, as shown in Fig. 5.13, the yellow boxes are no longer used. We do inference on both left-to-right and left-to-left branches. After generating the latents $\hat{\boldsymbol{z}}^{(l)}$ and $\hat{\boldsymbol{z}}^{(r)}$ via iterative diffusion denoising, the VAE decoder is applied (as shown in the green boxes) to reconstruct the videos of both views $\hat{\boldsymbol{x}}^{(l)}$ and $\hat{\boldsymbol{x}}^{(r)}$.

Note that during inference, we use DepthCrafter [Hu et al., 2024] to obtain video depth $D^{(l)}$ and use disparity scaling S as 0.03 for all test videos. We also show our method is robust to different S values in Sec. 5.2.3.

Post-processing with histogram matching. The output is not guaranteed to have the same brightness, exposure compared to the input. This is observed in StereoCrafter [Zhao et al., 2024]. To make sure the left and right views are consistent with each other, we post-process the output right-view with input as reference using histogram matching in the scikit-image library [van der Walt et al., 2014]. Figure 5.15 shows that histogram matching can help improve the brightness to better match the input.

5.2.3 Experiments

Implementation details

Data Generation. We preprocess for each input left-view video, the warped video, and mask via searching for a disparity scaling S in [0.02, 0.2], such that the warped video overlaps with the right-view video. Then we filter out those warped videos that do not overlap with the right-view one. Finally, we generated 958 videos (each with 16 frames) for training, with a total of 15,328 frames. For data augmentation, we apply

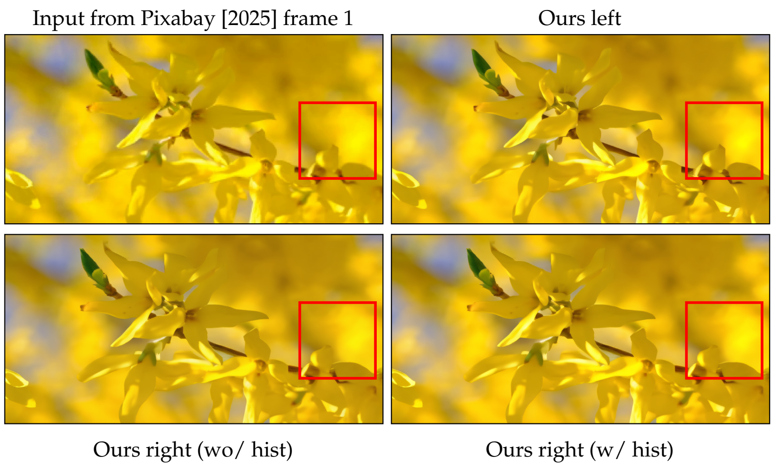


Figure 5.15: *The color histogram of Ours with histogram matching between left and right is better matched than Ours without histogram matching. This can be better observed around the red box region, where the output gets darker without histogram matching. Input is from Pixabay [2025] downsampled to* 320 × 160 *following Eq.* (5.10).

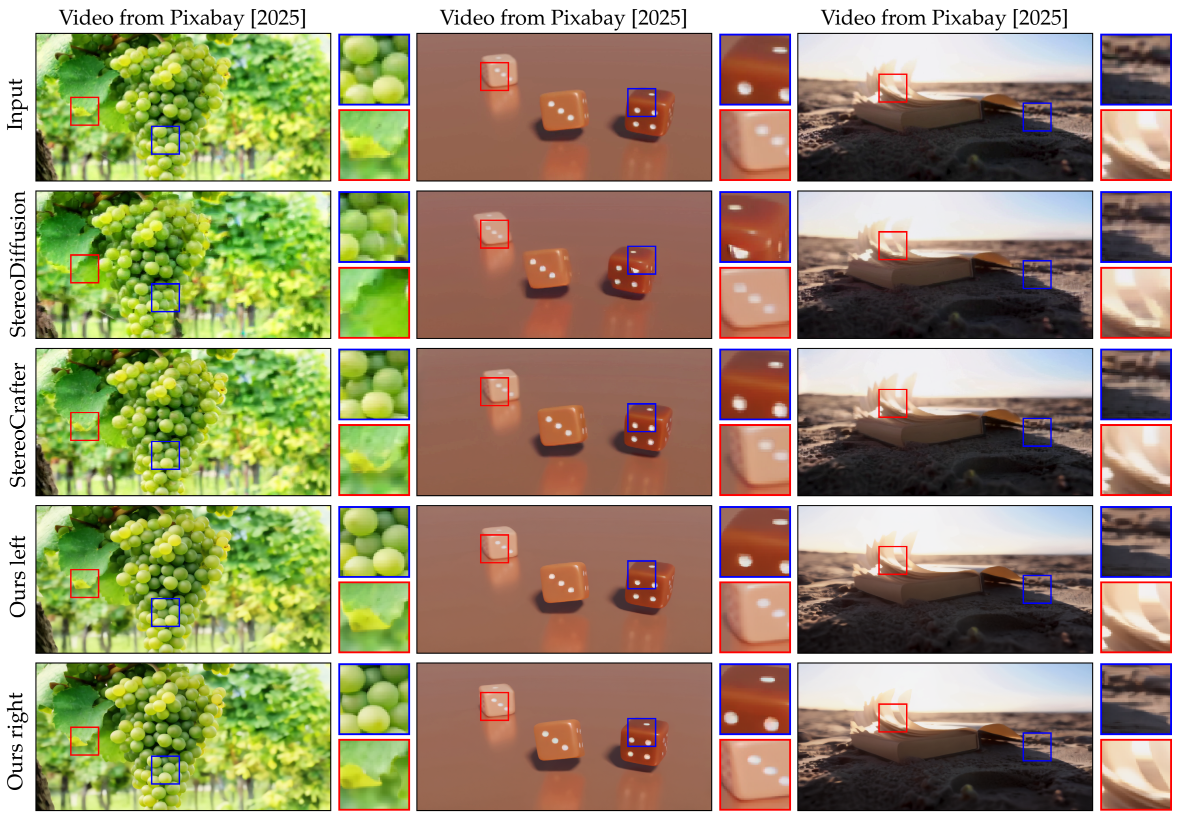

Video from Pixabay [2025]
Video from Pixabay [2025]
Video from Pixabay [2025]
Input
StereoDiffusion
StereoCrafter
Ours left
Ours right

Figure 5.16: *Stereo generation comparisons between StereoDiffusion [Wang et al., 2024], StereoCrafter [Zhao et al., 2024] and Ours. Our method shows sharper details across different scenes, highlighted in the zoom-in insets. Inputs are from Pixabay [2025] downsampled to* 320 × 160 *following Eq.* (5.10).

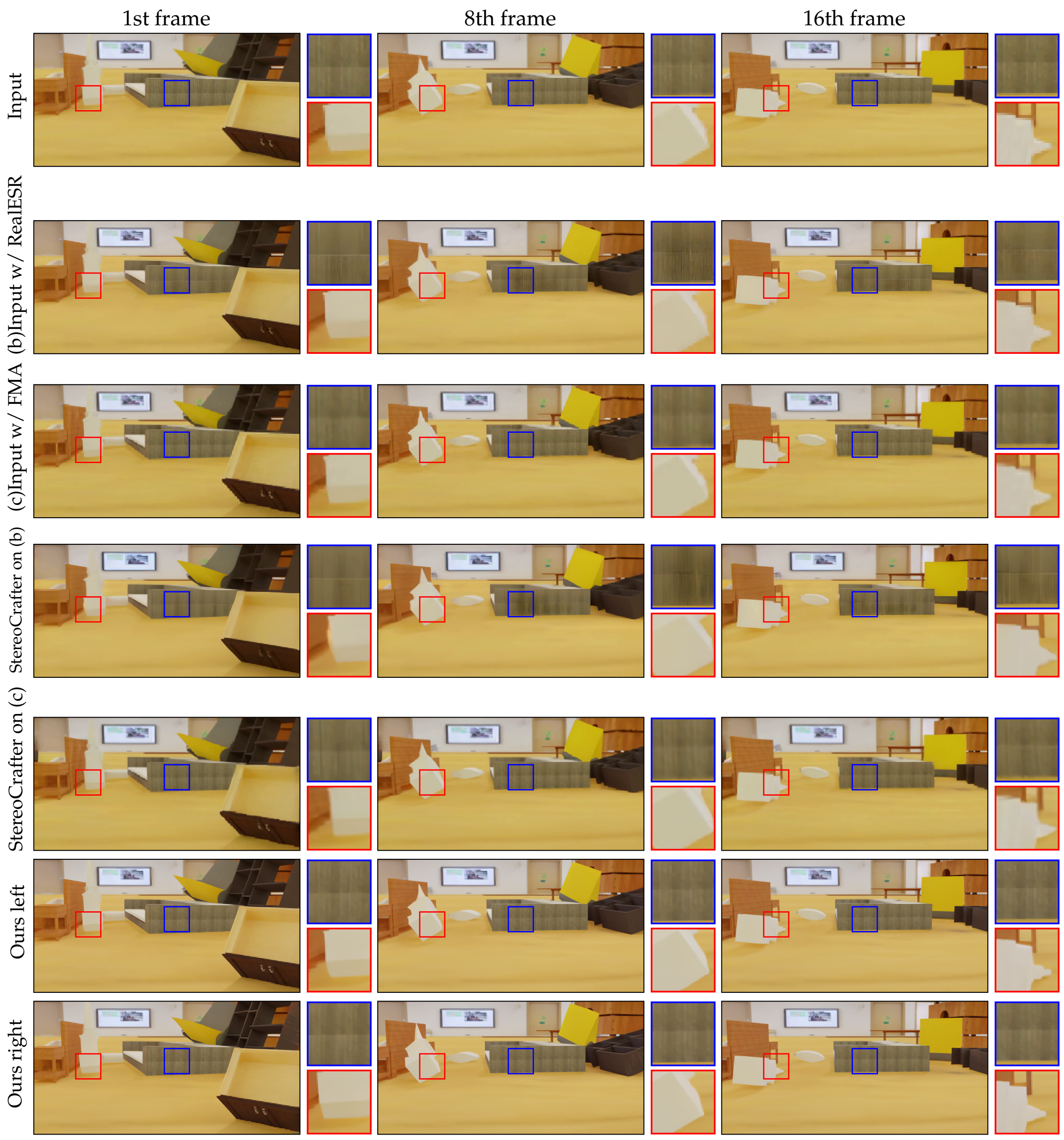


Figure 5.17: *Stereo generation and restoration comparisons between StereoCrafter [Zhao et al., 2024] with FMA-Net [Youk et al., 2024], with Real-ESRGAN [Wang et al., 2021] and Ours. Our method shows better temporal consistency and image quality than others, highlighted in the zoom-in insets. Input video is generated in Kubric [Greff et al., 2022] downsampled to* 256×128 *following Eq.* (5.10).

down-sampling, blurring, noising, and JPEG compression (following the implementation of Wang et al. [2021]) with the same random number for each video to ensure temporal consistency. In this way, we double the number of video inputs to 1,916 as training data.

Additionally, we generated 97 test videos using Kubric [Greff et al., 2022]. These test videos are used for evaluating the similarity between generated right-view videos and ground truth right-view videos, using different methods. For studying user perception, view, and temporal consistency, we collect 15 more monocular videos for testing, including synthetic videos from Kubric [Greff et al., 2022], SVD [Blattmann et al., 2023], and real-world videos from CLEVR [Johnson et al., 2017], Pixabay [Pixabay, 2025] and some self-captured videos. All test videos are not unseen during training.

Resolution. Training and testing videos are all with spatial resolution 1024×512. More specifically, videos are processed with the following equation:

$$\boldsymbol{x}' = \texttt{Up}(\texttt{Down}(\boldsymbol{x})) \tag{5.10}$$

Here we omit the superscript of $\boldsymbol{x}$, but $\boldsymbol{x}$ and $\boldsymbol{x}'$ can represent both left ($\boldsymbol{x}^{(l)}$, $\boldsymbol{x}'^{(l)}$) and right ($\boldsymbol{x}^{(r)}$, $\boldsymbol{x}'^{(r)}$) view videos consistent to the notations in Fig. 5.13. Each frame of the video $\boldsymbol{x}$ is downsampled to $W' \times H'$ implemented in PyTorch with:

$$\texttt{Interp} = \texttt{F.interpolate}(x, \text{ size} = (W', H'), \text{ mode} = \text{'area'}) \tag{5.11}$$

Note that `Down` also includes other degraded operations unrelated to resolution, like blurring, adding noise, and JPEG compression.

- During training/fine-tuning, $W' \times H' = 256 \times 128$ or 512×256 or 1024×512. `Up` is also implemented as `Interp` to turn the degraded video back to 1024×512.
- During testing, $\boldsymbol{x}_i$ is downsampled (`Down`) to 256×128, 320×160 or 360×180 and then restored back to 1024×512 via `Up` $\in$ {`Interp`, Real-ESRGAN [Wang et al., 2021], FMA-Net [Youk et al., 2024]} to get $\boldsymbol{x}'$. $\boldsymbol{x}'$ is obtained after restoration and the input to StereoCrafter [Zhao et al., 2024].

Fine-tuning. We use the Stable Video Diffusion (SVD) architecture initialized with the trained weights of StereoCrafter. Following the Pytorch [Paszke et al., 2019] implementation of Li et al. [2023b], we fine-tune the temporal transformer blocks and use the sampler from Karras et al. [2022] for training and sampling. Optimization is performed using AdamW optimizer [Loshchilov and Hutter, 2017] with a learning rate of 2e-5 and a batch size of 1. The resolutions of both input and output videos are 1024×512, as higher-resolution videos will cause out-of-memory issues. The degraded videos during training are upsampled back to 1024×512. The training takes 10,000 iterations in around 6 hours on an NVIDIA Tesla H100 GPU.

Inference time. We test the inference pipeline on an NVIDIA GeForce RTX 4090. Both our method (Ours) and StereoCrafter [Zhao et al., 2024] need to run DepthCrafter [Hu et al., 2024] with 20 diffusion sampling steps, which takes around 13 seconds per video (16 frames). For each video output with 16 frames, the inference of StereoCrafter takes around 19 seconds with 20 diffusion sampling steps. As we use the same architecture as StereoCrafter and need to run two inferences for generating consistent left-view and right-view videos, our inference time is around 38 seconds. In total, we take 51 seconds per video while StereoCrafter takes 32 seconds. While we can also run StereoCrafter to generate both left-view and right-view videos conditioned on the occlusion mask, we observe that this does not improve the view consistency of StereoCrafter, as it is not trained for that. Histogram matching can be performed in around 5 seconds per output right-view video using the input as a reference.

Table 5.4: *Quantitative evaluation on view and frame consistency between Ours, StereoCrafter [Zhao et al., 2024] and StereoDiffusion [Wang et al., 2024] from restored input using* `Interp` *(Eq.* (5.11)*), Real-ESRGAN [Wang et al., 2021], with FMA-Net [Youk et al., 2024]. We show consistent improvements over view and temporal consistency, which is also consistent with the user study in Table A.2.*

Method	$\text{LPIPS}_{\texttt{view}}(\downarrow)$	$\text{LPIPS}_{\texttt{temporal}}(\downarrow)$	$\text{CLIP}_{\texttt{view}}(\uparrow)$	$\text{CLIP}_{\texttt{temporal}}(\uparrow)$
StereoCrafter (`Up=Interp`)	0.3052	0.1300	0.9174	0.9854
StereoDiffusion (`Up=Interp`)	0.3419	0.1981	0.9214	0.9736
Stereocrafter (`Up`=Real-ESRGAN)	0.2551	0.1266	0.9285	0.9852
Stereocrafter (`Up`=FMA-Net)	0.2806	0.1327	0.9209	0.9843
Ours	**0.2393**	**0.1228**	**0.9820**	**0.9874**

Table 5.5: *User study scores on 3D Stereo Effect, Temporal Consistency across frames, and Image Quality per frame between Ours and StereoCrafter [Zhao et al., 2024] with restored input from the* `Interp` *and FMA-Net [Youk et al., 2024] methods. We show both the average score of all users and scenes, as well as the standard deviation in brackets. We show consistent improvements across different metrics, which is also consistent with the quantitative metrics in Table 5.4.*

	StereoCrafter (`Up=Interp`)	StereoCrafter (`Up`=FMA-Net)	Ours
3D Stereo Effect (↑)	3.81 (0.11)	3.68 (0.16)	**4.07** (0.10)
Temporal Consistency (across frames) (↑)	3.96 (0.05)	3.81 (0.23)	**4.28** (0.12)
Image Quality (per frame) (↑)	3.17 (0.22)	3.31 (0.19)	**4.48** (0.07)

Qualitative Comparisons

We compare our method with two recent diffusion-based methods: StereoDiffusion [Wang et al., 2024], StereoCrafter [Zhao et al., 2024], and its variants. StereoCrafter is trained to convert any monocular video to a 3D stereoscopic video. StereoDiffusion is a training-free method to convert an image to a stereo image pair using latent diffusion models [Rombach et al., 2022], which is not inherently designed for video. Official code for both methods is available for inference.

As shown in Fig. 5.16, both StereoCrafter and StereoDiffusion generate only the right-view video, which preserves most of the details from the input video while doing inpainting on the occluded region. As the input video is low-resolution, the generated right-view video is also blurry. This is highlighted in the grapes scene (1st column), the numbers on the dice (2nd column), and the edges of the book pages (3rd column). Our method improves the image quality for both left and right views.

StereoCrafter can be augmented by existing off-the-shelf video super-resolution method, such as FMA-Net [Youk et al., 2024] and Real-ESRGAN [Wang et al., 2021], following Eq. (5.10). This means the (degraded) input video can be preprocessed by FMA-Net, REAL-ESRGAN, or `Interp` (if not specified), before being fed into StereoCrafter. Real-ESRGAN is trained for image super-resolution, It does not perform temporally consistent generation, as shown in the wood texture (2nd row) of Fig. 5.17. FMA-Net is designed for video super-resolution, which takes 3 consecutive frames as input to ensure temporal consistency. However, it can still introduce inconsistency on the edges of the white object (3rd row).

Further, the upsampling quality of FMA-Net is not as sharp as ours, observed in Fig. 5.17 and appendix Fig. D.4. This can be because FMA-Net is designed for joint deblurring and super-resolution on dynamic outdoor scenes, where the architectural design and training objective are different from our method applied to stereo videos. Further, we train a single model for both stereo generation and restoration, while StereoCrafter (w/ FMA-Net) requires two independent models, more storage and resources compared to ours. Our method shows better visual quality in terms of sharpness. As we take all 16 frames as input, we also work well on temporal consistency. Lastly, we consider consistent training and inference for both left- and right-view, our generated videos are visually consistent across the two views as shown in Figures 5.16 and 5.17.

We also include additional results in appendix Fig. D.3 Fig. D.4 Fig. D.5 to show our method improved over others across different scenes, different downsampling scales, and scenes with moving cameras. More video comparison results can be found in the supplemental videos. Our method can fail to reproduce the details of specular highlight with highly reflected material as shown in appendix Fig. D.6, as our data for fine-tuning do not contain objects with complex material appearance.

Quantitative Comparisons

Generation quality. We use the 97 synthetic test videos rendered with Kubric [Greff et al., 2022] to evaluate the right-view generation quality of our method. We use Eq. (5.10) by downsampling left-view input videos $\boldsymbol{x}_i$ to 256×128 and upsampling back to 1024×512 to obtain $\boldsymbol{x}'$. LPIPS measures perceptual similarity between generated right-view videos and ground truth right-view videos. Table 5.6 shows better performance using our method compared to StereoCrafter [Zhao et al., 2024] with different restoration methods.

Table 5.6: *Comparisons between our method and StereoCrafter [Zhao et al., 2024] with different restoration methods on the quality of right-view video generation.*

Method	LPIPS (↓)
StereoCrafter (`Up=Interp`)	0.4271
StereoCrafter (`Up`=FMA-Net [Youk et al., 2024])	0.4232
Ours	**0.3422**

User study. To evaluate human perception, we compare StereoCrafter (`Up=Interp`), StereoCrafter (`Up`=FMA-Net), and Ours. The generated stereo videos of the 3 methods are shown to users using a VR headset (Meta Quest 3). There are in total 15 videos generated by each of the methods, which can be found in the supplemental videos.

We have in total of 15 computer science graduate students participating the user study, with 3 females and 12 males. Each user, with normal or corrected vision, was asked to score 5 of the videos generated by all methods based on visual inspection. The criterion includes 3D Stereo Effect, Temporal Consistency (across frames), and Image Quality (per frame). Each user scores from 1 (worst) to 5 (best) to evaluate the quality of each criterion and each method.

The results demonstrate that we clearly achieve the best score on image quality per frame. Our results on 3D stereo effect and temporal consistency across frames are also better.

View and temporal consistency. We also evaluate the view and temporal consistency using both LPIPS [Zhang et al., 2018] and CLIP [Radford et al., 2021] scores, where LPIPS is used to measure the perceptual difference, and CLIP is used for semantic similarity. Following Dai et al. [2024], we use the pretrained CLIP model [Radford et al., 2021] to extract features for both left and right views of a generated stereo video, and then calculate the feature-wise cosine similarity [Zhengwentai, 2023] to obtain the CLIP view consistency score. For the temporal consistency score, we compute the feature-wise cosine similarity between the previous frame and the current frame.

It is shown in Table 5.4 that our method outperforms other methods in terms of view consistency using LPIPS and CLIP. This can benefit from our design of training and inference with both views in a consistent manner. While we do not have a specific design for temporal consistency, our method still gets slightly better scores. Given that our evaluation is performed on a relatively small though diverse test set, we do not compute FID [Heusel et al., 2017] or FVD [Unterthiner et al., 2019]. Both are sensitive to the number of videos and require a large number of reference and generated videos to evaluate.

Ablation Study

Data augmentation. Figure 5.14 demonstrates that our idea of data augmentation with degradations during training is the key to simultaneous stereo video generation and restoration. As shown in the second column, without data augmentation, the model fails to enhance the quality of the input video. Table 5.7 also shows worse quantitative results by training without data augmentation compared to Ours (training with data degradation).

The effect of training data size. Our model performance scales with the increase of data in terms of both quantity and diversity. Table 5.7 shows fine-tuning the same model on a smaller-scale dataset of 10 and 100 stereo videos (compared to the original 958) with the same epochs but reduced object, motion, and scene diversity, leading to lower generation quality. This indicates the possibility of performance gain over more diverse test videos by fine-tuning with more diverse 3D objects, materials in the dataset.

Table 5.7: *Comparisons of training with different datasets on the quality of right-view video generation.*

Method	LPIPS (↓)
Ours (fine-tuned wo/ augmentation)	0.3664
Ours (fine-tuned w/ 10 stereo videos)	0.3878
Ours (fine-tuned w/ 100 stereo videos)	0.3446
Ours	**0.3422**

Performance on high-resolution test videos without degradation. Table 5.8 shows experiments on the 15 test videos, but with high-resolution (1024×512) videos without degradation in Eq. (5.10). This further emphasizes that our fine-tuning with data augmentation works for input videos with varying levels of degradation.

Table 5.8: *Quantitative view and frame consistency between StereoCrafter and Ours on high-resolution test videos without degradation.*

Method	LPIPS$_{\texttt{view}}$($\downarrow$)	LPIPS$_{\texttt{temporal}}$($\downarrow$)	CLIP$_{\texttt{view}}$($\uparrow$)	CLIP$_{\texttt{temporal}}$($\uparrow$)
StereoCrafter	0.2752	0.1283	0.9317	0.9860
Ours	**0.2695**	**0.1236**	**0.9682**	**0.9874**

Robustness to different disparity scaling S. Appendix Fig. D.2 shows that our generation is robust to different disparity scaling factors during inference, with similar quality output.

5.2.4 Conclusion

We have presented a novel method that can perform simultaneous stereo video generation and restoration with a single model. Our method leverages data augmentation with image degradations on synthetic data, as well as a consistent training and inference pipeline to achieve this. Our designed pipeline enables better results across a diverse set of scenes with low-resolution inputs, compared to existing methods that do not consider both generation and restoration.

Chapter 6
Conclusion

In this thesis, we explored how noise correlation can play a central role in both the synthesis and editing of point patterns, as well as the training of diffusion models for image and video generation.

First, we developed a suite of correlation-aware tools for point pattern authoring, introducing new filters and editing interfaces that allow users to directly control spatial statistics such as density and correlations. Our contributions include an example-based synthesis method (see Sec. 4.1) and an interactive framework that enables fine-grained control through image-based editing (Sec. 4.2). These tools support both artistic workflows and stochastic modeling in pattern and text design.

Second, we proposed an initial study on how the noise used in diffusion models affects generation quality. Motivated by the success of blue noise in Monte Carlo rendering, we introduced a time-varying blue noise formulation for training diffusion models (Sec. 5.1). Our results show that blue noise can improve high-frequency detail and texture fidelity in generated images, suggesting that alternatives to Gaussian noise deserve deeper investigation in generative modeling. We also studied how noise can be used for image degradation during training diffusion-based stereo video generation models. With such a degradation component, our model is more robust to input video with varying levels of degradation.

6.1 Discussion

Noise correlation for point patterns. A key insight from our experiments is that a point pattern can be characterized using image-based representations. Depending on the applications, such representations make it straightforward to design different filters, representations, and tools to re-synthesize a similar point pattern, synthesize a larger-scale pattern, and apply local edits to either density or correlation.

For example, our correlation-aware filters (Sec. 4.1) are able to extract both spatial and spectral features without any training. The features can be used for synthesizing a larger-scale pattern with a similar structure. Our decoupled representation of a point pattern using density and correlation maps (Sec. 4.2) allows us to edit both properties independently.

Several challenges remain. Decomposing a pattern into density and correlation is mathematically ill-posed and perceptually tricky: two patterns rendered with identical dot sizes but different correlation can look "denser" or "sparser" to the human eye. Resolving this ambiguity may require an additional perceptual dimension (e.g., adaptive dot sizing) and a richer model that couples density, correlation, and rendering parameters.

Noise correlation for diffusion models. Our blue noise schedule (Sec. 5.1) forces a rethink of a long-standing assumption in diffusion modeling: that Gaussian, i.i.d. noise is the “natural” choice for perturbing data. The sampling process of a standard diffusion model shows that early denoising steps recover coarse color and shape, while later steps refine details much as before. This is correlated with our idea of time-varying blue noise, where we force the model at later steps not to corrupt the low-frequency components but focus on generating high-frequency details. The benefit is noticeable as sharper details are shown across different datasets.

The approach is not without trade-offs. Blue noise has no obvious analogue in the latent spaces used by modern latent diffusion models, and forcing it there can introduce undesirable inter-channel correlations. Moreover, because the blue noise covariance must be tiled to arbitrary resolutions, training cost scales less favourably than with white noise and becomes a bottleneck beyond 256^2 pixels. Future work on learned or resolution-adaptive noise could mitigate these costs while preserving the gains in detail.

In our stereo video generation project, we apply noisy degradation to train a stereo video diffusion model such that the model is robust to different levels of degraded input videos (Sec. 5.2). However, the potential of correlated noise, such as blue noise, remains largely unexplored for video generation, where temporal coherence across frames and view consistency in multi-view sequences are critical. Understanding whether spatial-temporal noise priors can enhance these aspects is an open question.

Shared Insights and Limitations. Our explorations into point pattern synthesis and image/video diffusion are built on the same simple idea: control the frequency content of the noise. Whether the signal is a cloud of points or an image grid, its spatial correlations are determined by the noise’s Fourier spectrum. This unifying view indicates that the same noise model can be repurposed to represent correlations for different representations and applications. We also empirically found that correlated noise, like blue noise, sharpened perceptual detail in both domains: it produced more uniform inter-point spacing in point patterns and cleaner high-frequency textures in image generation.

The shared drawback is speed and dimensionality. Generating or editing correlated noise takes longer than using random Gaussian (white) noise. In diffusion models, for instance, we need to first precompute and store the blue noise covariance matrix and apply it on the fly. This makes generating blue noise compared to generating random Gaussian noise. Even worse, unlike white noise, the blue noise generation step gets slower as the resolution goes up. This makes blue noise generation slower than generating random Gaussian noise. The gap widens as resolution increases, because blue noise generation time scales up while white noise remains almost unchanged. Similarly, generating point patterns with correlated noise requires a lengthy optimization step, and this step becomes even harder in higher-dimensional spaces due to the curse of dimensionality.

6.2 Future Work

There are several promising directions to extend the work presented in this thesis.

For point pattern synthesis:

- Currently, our pipeline is mainly designed for synthesizing and editing 2D point patterns. With the rapid development of 3D reconstruction and generation, it would be beneficial to extend the correlation space to take 3D point clouds into consideration. This will give users more control over 3D representation, with better reconstruction quality or controllability in 3D Vision research. Recent 3D Gaussian Splatting [Kerbl et al., 2023] is tightly related to 3D point-based representation and can be beneficial from 3D point correlations. Extending our framework to 3D would be promising in these areas.
- As real-time user interaction and feedback remain limited in our work, integrating these tools into graphics software or GPU-based workflows can significantly improve the usability of our methods.
- Inspired by the recent work of Doignies et al. [2023], which adapts diffusion models to example-based point pattern synthesis, we see a promising research avenue in extending diffusion frameworks beyond regular image grids. But unlike image generation, point pattern synthesis requires modeling sparse and highly irregular point-based structure, which is a more challenging problem. In addition, large-scale, high-quality datasets of annotated point patterns are still lacking, making it difficult to train or benchmark such models at scale. Addressing both the representational challenge and the data bottleneck will be essential for bringing the full power of diffusion models to point-based geometry.

For diffusion models:

- Our use of blue noise is an initial step; other types of structured noise (e.g., pink noise, structure-aware noise, learned noise schedules) could be explored for different types of content.
- A deeper analysis is needed to understand why and how correlated noise can affect training convergence in diffusion models. This would be important to guide further research along this direction.
- The current blue noise implementation does not scale well to high-dimensional problems and more complex data distributions. It would be promising to study whether correlated noise can be used for large-scale diffusion model pretraining to outperform existing state-of-the-art models trained with random Gaussian noise.
- Our proposed time-varying blue noise will fall back to white noise for one-step diffusion. But currently, one-step diffusion still suffers from an obvious drop of quality in the output [Liu et al., 2023b]. Few-step diffusion provides a good trade-off between one-step and many-step diffusion. In this case, our time-varying blue noise can be useful. It would be interesting to apply the idea of blue noise to distill state-of-the-art text-to-image models and see the differences compared to distillation with random Gaussian noise.
- Inspired by recent development of temporally-correlated noise [Chang et al., 2025], we can think of combining the idea of spatially-correlated noise (e.g., blue noise)

with temporal-correlated noise, to construct a new type of spatial-temporal correlated noise. Video diffusion models can benefit from both temporal consistency and spatial quality if we can incorporate spatial-temporal correlated noise into the training or fine-tuning of the model. Further, stereo video diffusion models can also benefit from spatial-temporal correlated noise, and stereo generation requires more strict view- and temporal consistency, visualized in a VR headset.

Overall, this thesis highlights the often-overlooked role of noise correlation in generative modeling of point patterns, images, and videos, showing that the correlation structure in randomness can be both controllable and beneficial. We hope these contributions inspire further research into the intersection of noise, sampling, generative modeling, and controllable graphics content generation and editing.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments on all our works. We want to thank Pierre Ecormier-Nocca for helping with object placement in Blender [Com18] and the authors Reddy et al. [2020] for making their source code and data publicly available. Renderings shown in the Sec. 4.1 use free models from Turbosquid [Tur21] under the "Editorial Use" license. We want to thank artist Bianchini Jr. for the Pablo Piccaso artwork (shown in Sec. 4.2) which we use under the Media Licence.

For the project described in Sec. 5.1, Xingchang Huang is supported by Saarbrücken Research Center for Visual Computing, Interaction and Artificial Intelligence, a joint strategic partnership between MPI and Google. Due to the agreement, we do not display the original images from the CelebA dataset [Lee et al., 2020] in Figures 5.1, 5.12, 5.6, 5.7. These images are all generated with our method.

Appendix A
Training-free correlations for point pattern synthesis

A.1 Analysis of Multi-scale Optimization

By-example point pattern synthesis methods usually require users to tune with the window size or kernel size parameters, as shown in previous state-of-the-art methods [Ma et al., 2011] [Roveri et al., 2015] [Tu et al., 2019], to get satisfying synthesis results. Similarly, we use a multi-scale optimization strategy to preserve local and non-local structures for the synthesized patterns. Here, we discuss the importance of multi-scale optimization and propose a way to tune the two hyperparameters that control the kernel size of our proposed Gabor features.

Single-scale vs. Multi-scale optimization. Fig. A.1 demonstrates that our multi-scale optimization is important for preserving both global and local structures. The value of σ is analogous to the receptive field; a higher σ value captures more global structure, while a lower σ value focuses more on local structures. As shown in the figure, optimizing with only σ_1 results in a good global structure, but locally the points do not follow the regularity in the input. While optimizing only σ_2 leads to the missing global structure. Therefore, multi-scale optimization firstly focuses on synthesizing a pattern with a good global structure and then refines the local structures as the decrease of σ value decreases.

Tuning Hyper-parameters. As discussed, a pattern can express different levels of structure, and choosing an appropriate window or kernel size is a necessary step for high-quality synthesis. In some cases, choosing the parameters incorrectly can lead to unpleasing synthesis results. Therefore, we propose a pragmatic way to tune the two hyperparameters, namely c_1 and c_2. As mentioned, most of the scenes use c_1 in 0.8 ± 0.2 and c_2 is 2.8 ± 0.2. For a new test scene, as default, we start from $c_1 = 0.6, c_2 = 2.6$. As shown in Fig. A.2, when the number of points in the exemplar is small, the feature map

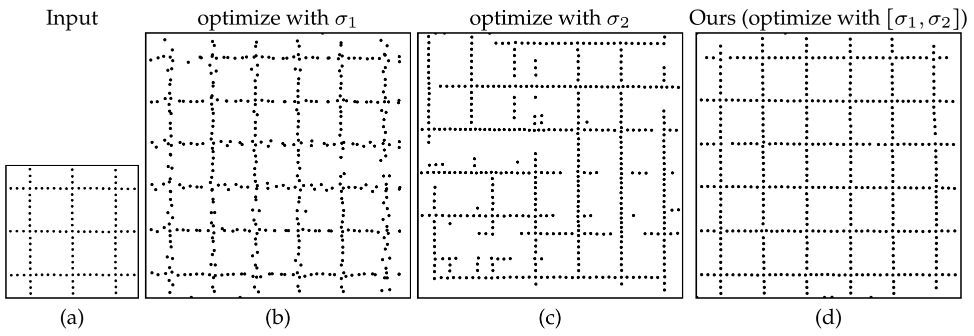


Figure A.1: *Importance of multi-scale optimization. If only optimized with a single scale, the synthesized results (b), (c) will only capture either global or local structure.*

Table A.1: *We summarize statistics and parameters used to synthesize each exemplar. M, S, and R refer to the figure in Sec. 4.1, the appendix, and the row number of each figure, respectively.*

Scene	#Classes	#Output samples	Hyper-parameters	
			c_1	c_2
Fig. 4.5 (M), (a)	1	1124	1.0	3.0
Fig. 4.5 (M), (b)	1	256	1.0	2.6
Fig. 4.5 (M), (c)	1	992	0.8	2.6
Fig. 4.6 (M), (a)	2	208	0.8	2.6
Fig. 4.6 (M), (b)	2	512	1.6	2.6
Fig. 4.6 (M), (c)	4	216	1.0	2.8
Fig. 4.8 (M), R1	1	64	2.0	3.0
Fig. 4.8 (M), R2	2	512	1.6	2.6
Fig. A.8 (S), R1	1	80	1.0	2.6
Fig. A.8 (S), R2	1	192	0.8	2.6
Fig. A.8 (S), R3	1	508	0.6	2.6
Fig. A.8 (S), R4	1	472	0.8	2.6
Fig. A.8 (S), R5	1	468	0.8	2.6
Fig. A.9 (S), R1	1	424	1.0	2.6
Fig. A.9 (S), R2	1	260	1.0	2.6
Fig. A.9 (S), R3	1	476	0.8	2.6
Fig. A.9 (S), R5	1	804	0.8	2.8
Fig. A.9 (S), R6	1	1024	1.0	2.7

of c_1 may not show the overall structure of the point pattern and lead to over-blurring features. This can result in less visually pleasing results. When we start increasing c_1 to be 0.6 and 0.8, the overall structure becomes more visible. Users can repeat this process until they are satisfied with the synthesized pattern. Fortunately, our method allows us to get satisfying synthesis results after a few trials for most of the scenes. We summarize the parameters for all scenes in Table A.1. As shown in the Table, the hyperparameters are not so different across a large variety of scenes. This further demonstrates that our method is robust to hyper-parameter settings, and users can find a good solution without too much manual effort.

A.2 Diverse Results

We show that our method inherently supports synthesizing diverse results given the same exemplar, using randomly initialized point patterns with different random seeds.

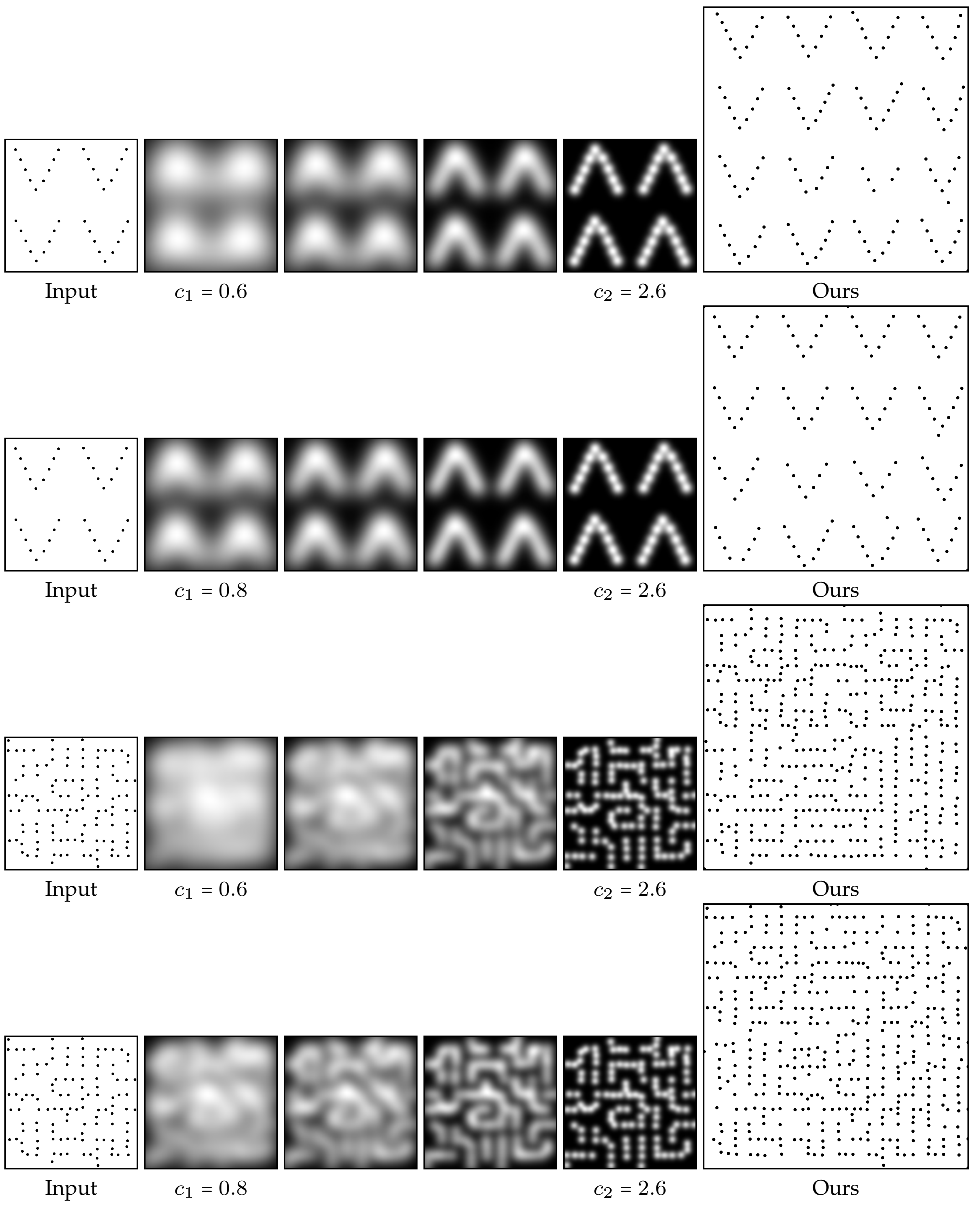


Figure A.2: *Hyper-parameter analysis. We demonstrate a pragmatic way to tune/increase the parameters c_1 and c_2 from the default values $c_1 = 0.6, c_2 = 2.6$ to get the final results.*

Fig. A.3 shows three different outputs with different random initialization given the same input. Meanwhile, the overall structure looks similar to the exemplar. This also demonstrates that our method is robust to different initializations.

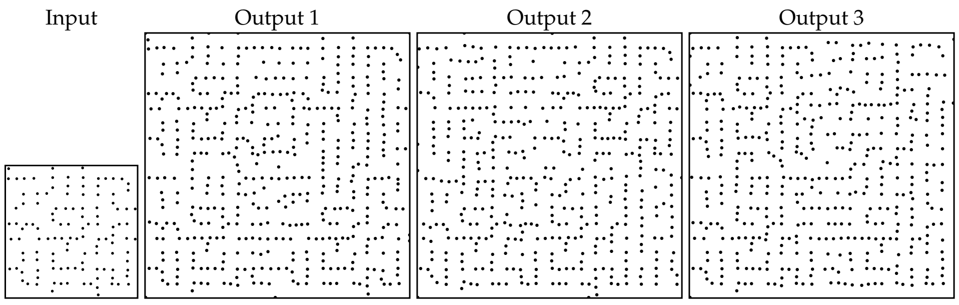


Figure A.3: *Diverse results. Given the same exemplar, our method synthesizes diverse outputs with different random initializations.*

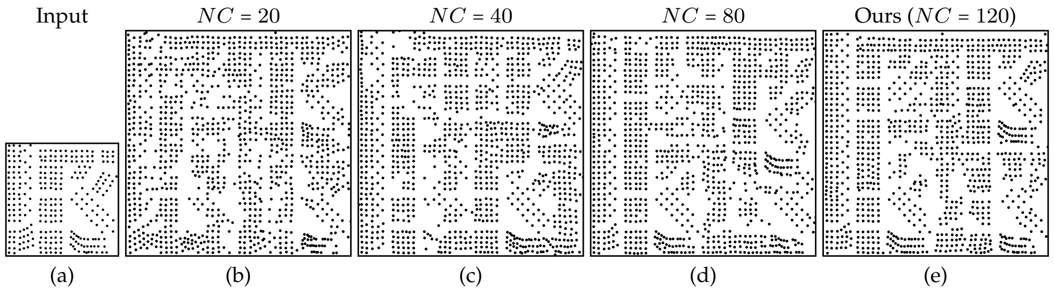


Figure A.4: *Ablation study on number of output channels (NC) in convolutional filters. Increasing NC leads to better results, and NC = 120 is a reasonable choice considering the trade-off between run-time and synthesis quality.*

A.3 Additional Results

A.3.1 Ablation Study

Number of channels. In the convolutional filtering step, we define the number of output channels NC to control the number of random filters we use. Fig. A.4 shows how the number of output channels in the convolutional layer affects the final results. We observe less accurate synthesis results with fewer filters defined by the number of output channels in the convolutional filters. Increasing the number of output channels increases the synthesis quality. We find that $NC = 120$ is a reasonable choice, as increasing the number of channels beyond 120 brings a marginal difference while being more computationally expensive.

Layer for Correlation loss computation. We also study the layer noted as l chosen for computing L_{corr}. We use the $4th$ ($l = 4$) layer for computing L_{corr}. Other options are to use output from layer $l = 1, 2, 3$. However, L_{corr} becomes more computationally expensive with higher resolution feature maps. We observe an out-of-memory issue while using the $1st$ or $2nd$ layer. Using the $3rd$ layer we get similar outputs as shown in Fig. A.5, but the run-time is on average 6 times more than our choice.

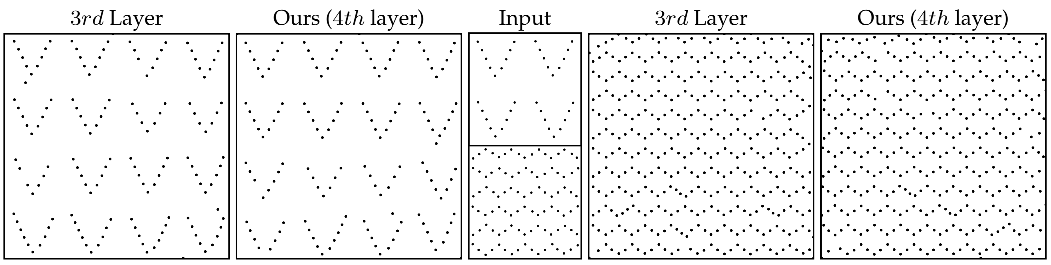


Figure A.5: *Ablation study on the layer selected for L_{corr} computation. Using the $4th$ layer output gives similar results compared with using the $3rd$ layer output, but is much less expensive due to lower resolution features.*

A.3.2 Qualitative Comparisons

Point patterns. We show more results in Fig. A.8 and Fig. A.9 for qualitative comparisons between our method and previous state-of-the-art methods. Note that these patterns are included in the user study. Among them, our method achieves the highest user scores compared with previous state-of-the-art methods on most of the scenes.

Element patterns. For discrete element-based pattern expansion, we experiment with different variants of the input exemplars and the method of Reddy et al. [2020]. Fig. A.6 shows comparisons on 2-class patterns using our method, Reddy et al. [2020] and Tu et al. [2019]. As DiffCompositing [Reddy et al., 2020] uses only the Gram loss L_{gram} in their original paper, we test 2 more variants using their methods by including the Deep Correlation loss L_{corr}. However, as shown in Fig. A.7, we do not observe obvious improvement using their method, especially on the orange pattern with clear vertical structures in the middle column. On the other hand, our method performs better in terms of local and non-local structures. For patterns with more randomized structures, our method synthesizes patterns with fewer overlaps and closer to the exemplar's structure compared with Reddy et al. [2020].

Further, we show in Fig. A.10 that our methods not only apply to discrete elements in $2D$, but also to elements with higher-dimensional features. As shown in Fig. A.10, our method takes input point patterns with features including depth, scale, and $3D$ orientation for synthesis. This allows us to use the synthesized patterns for object placement and pattern design, not limited to $2D$ space.

A.3.3 User Study

Table 4.2 shows the numbers of average scores from 28 participating users for the 14 point patterns used for our user study. Patterns we use are from Fig. 4.5 in Sec. 4.1, Fig. A.8 and Fig. A.9 in the appendix.

Table A.2: *We perform a user study and compute an average score across 28 users. We show the average score for each pattern, where 1 is the worst and 5 is the best. Our method gets a better score for all but one pattern. M, S, and R refer to the figures in Sec. 4.1, the appendix, and the row number of the corresponding figure, respectively.*

Scene	User Scores (↑)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 5 (M), (a)	1.4286	1.9643	3.6071	**4.2857**
Fig. 5 (M), (b)	3.7143	2.6071	3.5357	**4.6071**
Fig. 5 (M), (c)	1.4643	1.3571	3.6071	**4.4286**
Fig. A.8 (S), R1	1.5714	3.0714	2.8571	**4.2857**
Fig. A.8 (S), R2	1.2857	2.3214	4.2857	**4.3214**
Fig. A.8 (S), R3	1.2857	1.9286	3.6071	**3.9286**
Fig. A.8 (S), R4	2.5714	3.0357	4.0357	**4.6071**
Fig. A.8 (S), R5	1.5000	3.9643	2.8214	**4.3929**
Fig. A.9 (S), R1	1.5714	2.5357	3.6071	**4.3929**
Fig. A.9 (S), R2	3.2143	3.0000	2.8929	**4.2500**
Fig. A.9 (S), R3	1.5357	**4.5357**	3.4643	4.3214
Fig. A.9 (S), R4	1.1071	3.3571	2.5000	**4.4643**
Fig. A.9 (S), R5	1.2143	3.2143	2.8929	**4.1071**
Fig. A.9 (S), R6	1.6071	2.7500	2.6429	**4.4643**

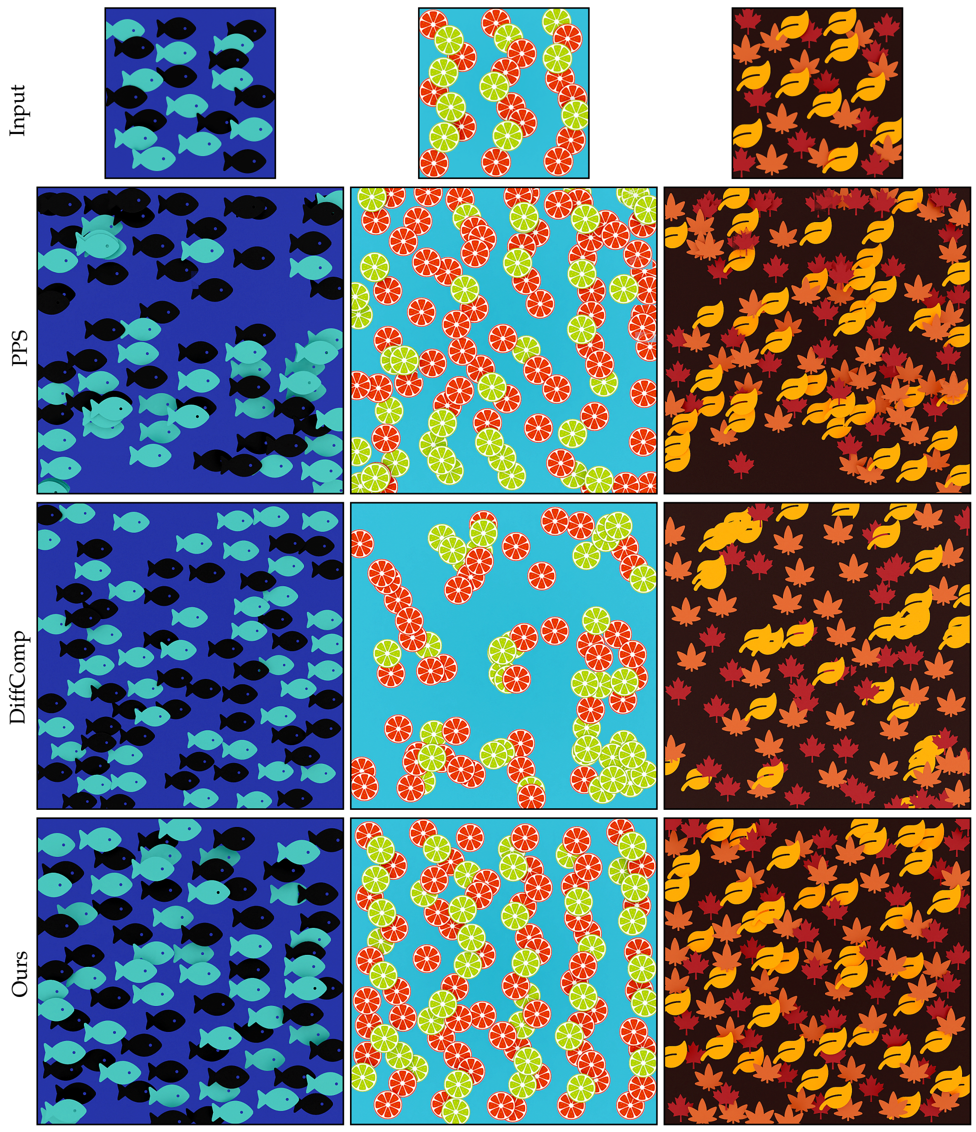


Figure A.6: *Discrete element-pattern expansion. We show 2-, 2-, and 3-class examples from left to right and comparisons between PPS [Tu et al., 2019], DiffComp [Reddy et al., 2020], and our method from top to bottom.*

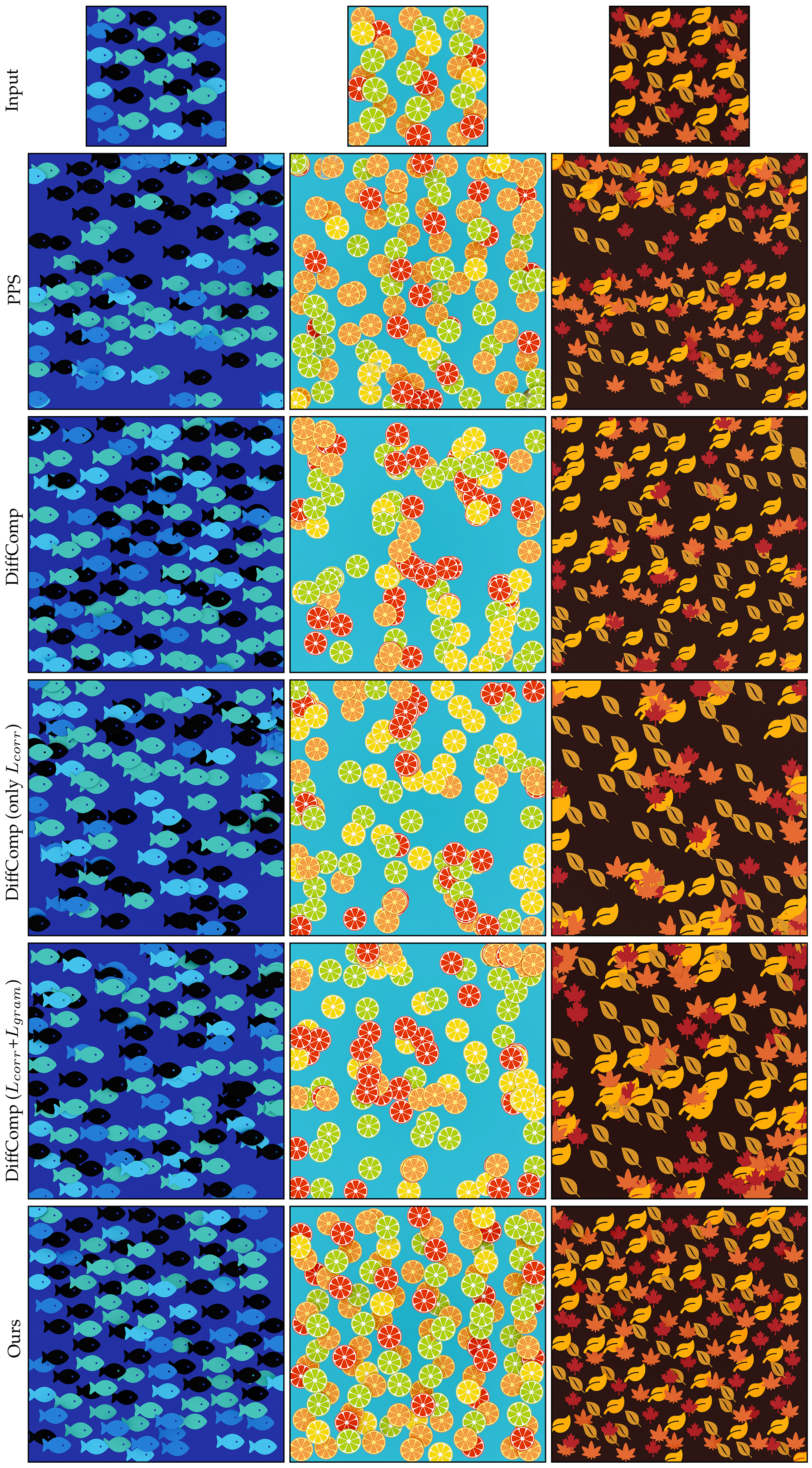


Figure A.7: *Discrete element-pattern expansion with 4-class examples. We test two more variants of DiffComp [Reddy et al., 2020] with only L_{corr} and a weighted combination of L_{corr} and L_{gram} for fair comparisons with PPS [Tu et al., 2019] and our method.*

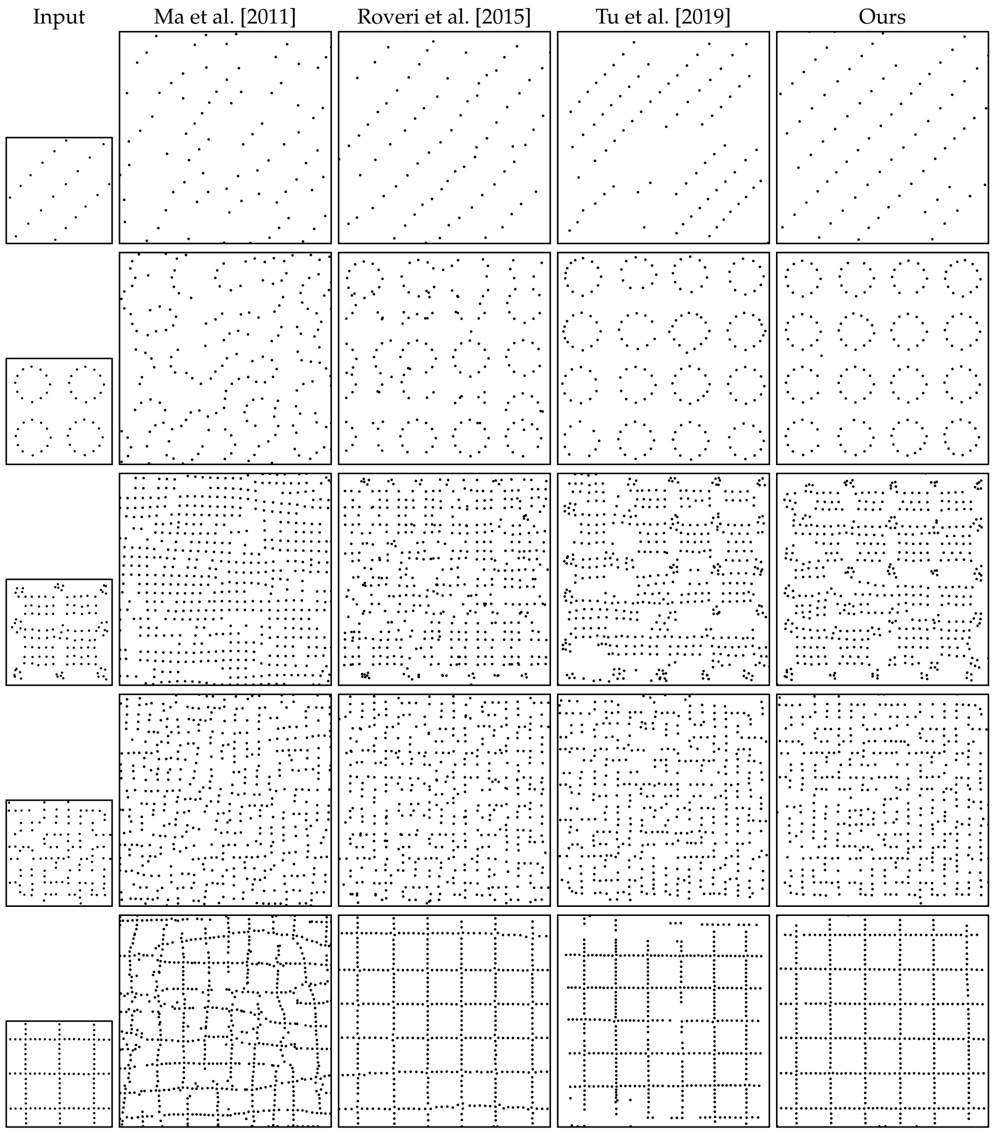


Figure A.8: *Additional comparisons between prior methods Ma et al. [2011], Roveri et al. [2015], Tu et al. [2019] and ours, respectively.*

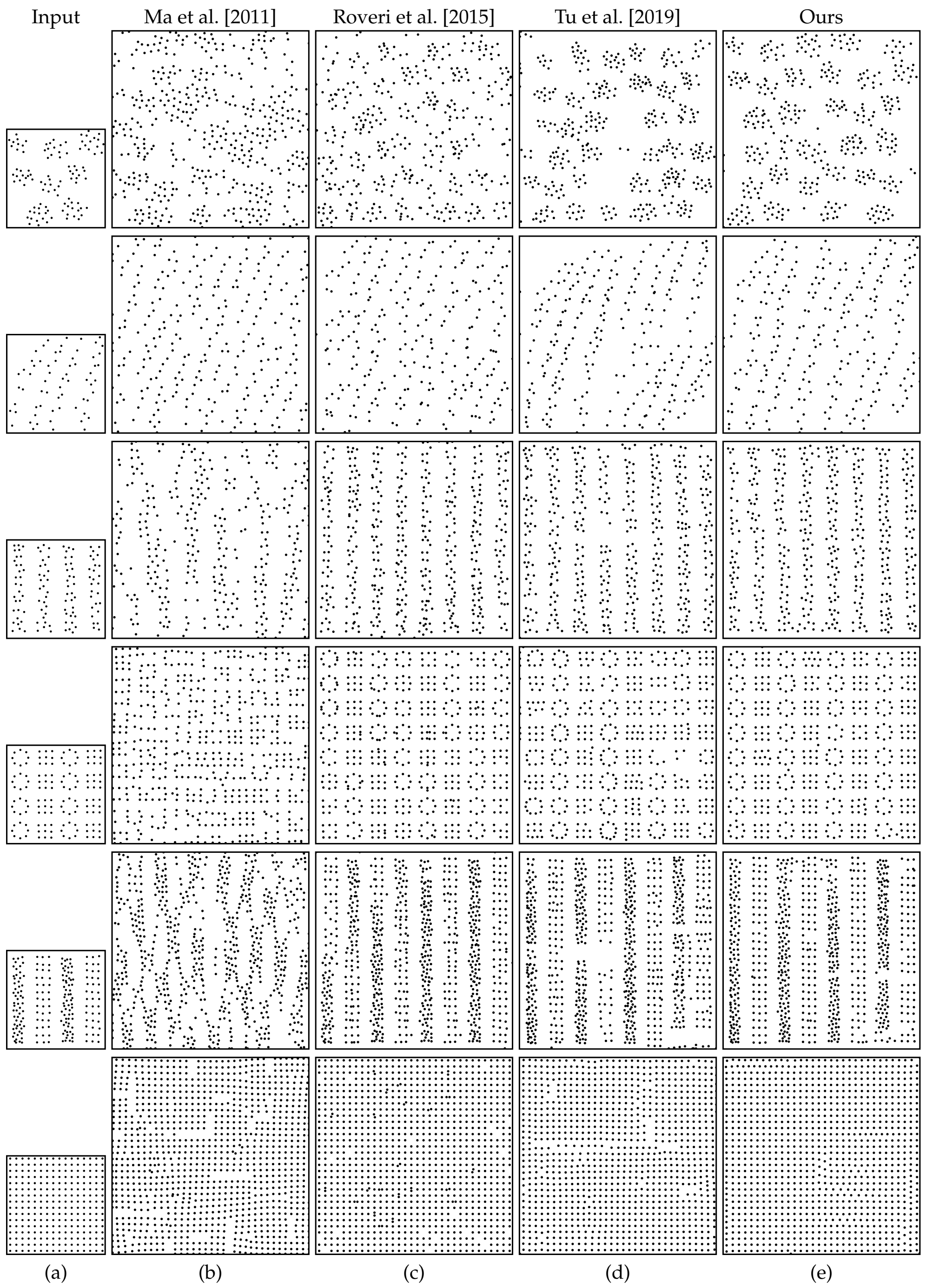


Figure A.9: *Additional comparisons between prior methods Ma et al. [2011], Roveri et al. [2015], Tu et al. [2019] and ours, respectively.*

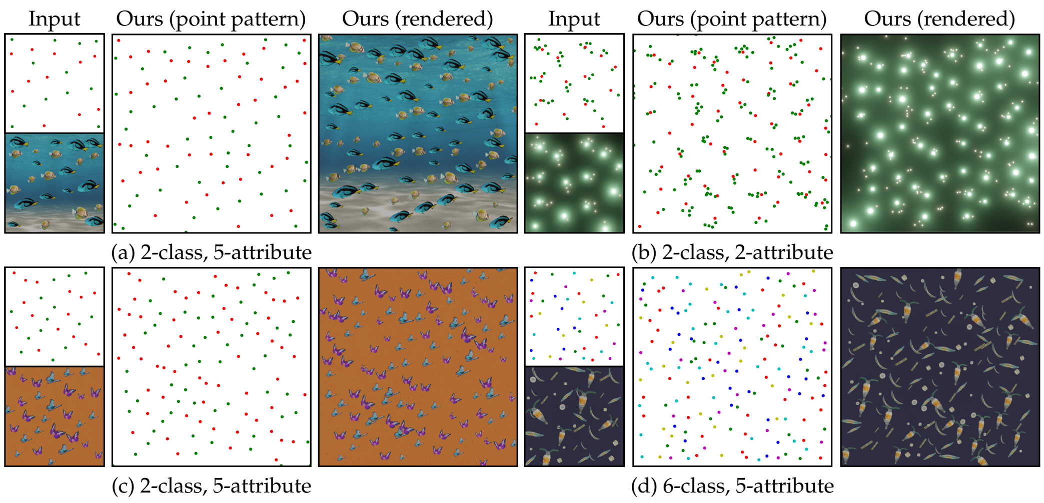


Figure A.10: *Multi-attribute and multi-class point pattern synthesis results. We show* 2*-class examples from (a) to (c) and a* 6*-class example in (d). To visualize the patterns with multiple classes and attributes, we show* 2*D point patterns along with the rendering of* 5 *attributes, including scale, depth, and* 3*D orientation.*

Table A.3: *Quantitative comparisons for point pattern synthesis results of previous methods and ours. M, S, and R refer to the figures in Sec. 4.1, appendix, and row number of the corresponding figure, respectively.*

Scene	MSE of PCFs (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 5 (M), (a)	0.2678	0.2836	0.2609	**0.2459**
Fig. 5 (M), (b)	0.3296	0.3286	0.3103	**0.3072**
Fig. 5 (M), (c)	0.6346	0.4913	0.5436	**0.3911**
Fig. A.8 (S), R1	0.7562	0.7130	0.9043	**0.6395**
Fig. A.8 (S), R2	0.4253	0.4188	0.3964	**0.3656**
Fig. A.8 (S), R3	0.3731	0.3422	**0.2799**	0.2881
Fig. A.8 (S), R4	0.2451	0.2642	0.2376	**0.2374**
Fig. A.8 (S), R5	0.5038	0.3548	**0.3455**	0.3538
Fig. A.9 (S), R1	0.5507	0.5537	0.4283	**0.4138**
Fig. A.9 (S), R2	0.2915	0.2947	0.2858	**0.2786**
Fig. A.9 (S), R3	0.2174	0.2570	**0.2403**	0.2427
Fig. A.9 (S), R4	0.2661	0.2528	0.2623	**0.2483**
Fig. A.9 (S), R5	0.4153	0.3438	**0.2872**	0.3429
Fig. A.9 (S), R6	0.2401	0.2218	**0.2199**	0.2219

Scene	Wasserstein Distance (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 5 (M), (a)	1.4483	1.1148	1.0222	**0.6896**
Fig. 5 (M), (b)	0.4858	0.5166	0.4885	**0.3352**
Fig. 5 (M), (c)	2.1869	2.2596	2.2570	**0.5287**
Fig. A.8 (S), R1	0.6292	0.6382	**0.6048**	0.7038
Fig. A.8 (S), R2	0.7266	0.5322	0.4303	**0.2813**
Fig. A.8 (S), R3	0.8372	0.7832	0.8406	**0.7158**
Fig. A.8 (S), R4	0.6775	0.6040	0.6415	**0.5068**
Fig. A.8 (S), R5	1.3144	**0.2885**	1.1597	0.3749
Fig. A.9 (S), R1	1.3111	1.1803	1.3618	**0.8991**
Fig. A.9 (S), R2	0.7385	0.7409	0.9657	**0.6828**
Fig. A.9 (S), R3	1.1793	0.6341	0.7887	**0.6230**
Fig. A.9 (S), R4	0.7412	0.5076	0.5882	**0.2367**
Fig. A.9 (S), R5	0.9996	0.8678	0.8087	**0.6586**
Fig. A.9 (S), R6	0.5531	0.3562	0.4701	**0.2345**

Scene	Chamfer Distance (↓)			
	Ma et al. [2011]	Roveri et al. [2015]	Tu et al. [2019]	Ours
Fig. 5 (M), (a)	0.0571	0.0440	0.0517	**0.0353**
Fig. 5 (M), (b)	0.0962	0.0869	0.0920	**0.0429**
Fig. 5 (M), (c)	0.0662	0.0685	0.0719	**0.0184**
Fig. A.8 (S), R1	0.2131	0.1703	0.2116	**0.1601**
Fig. A.8 (S), R2	0.1567	0.0701	0.0556	**0.0169**
Fig. A.8 (S), R3	0.0868	0.0788	0.0905	**0.0774**
Fig. A.8 (S), R4	0.0803	0.0685	0.0718	**0.0344**
Fig. A.8 (S), R5	0.1000	0.0342	0.1233	**0.0271**
Fig. A.9 (S), R1	0.1065	0.1123	0.1200	**0.0779**
Fig. A.9 (S), R2	0.1221	0.0946	0.1205	**0.0990**
Fig. A.9 (S), R3	0.1031	**0.0543**	0.0612	0.0631
Fig. A.9 (S), R4	0.0746	0.0170	0.0548	**0.0149**
Fig. A.9 (S), R5	0.0723	0.0394	0.0408	**0.0294**
Fig. A.9 (S), R6	0.0473	**0.0043**	0.0312	0.0070

Appendix B
Noise correlation embedding for point pattern editing

In this supplemental, we provide further details on the implementation (Sec. B.1), as well as additional results (Sec. B.2).

B.1 Additional Implementation Details

B.1.1 Point correlations generation

We use a mixture of Gaussians to sample the gamut of possible power spectra. We use power spectra here, as they allow us to directly work with a different range of frequencies, unlike PCFs. The value of a power spectrum bin p_b is computed as

$$p_b = \sum_{i=1}^{n_{\mathrm{GMM}}} \left(w_i \cdot \exp(-\frac{(b-\mu_i)^2}{2\sigma_i^2}) \right) + \gamma. \tag{B.1}$$

We use a mixture with, randomly, either $n_{\mathrm{GMM}} = 1$ or $n_{\mathrm{GMM}} = 2$ Gaussians and sample the parameters from the range $\gamma \in \{0,1\}$, $\mu_i \in [0,68]$, $\sigma_i \in [2,12]$, and $w_i \in [1,3]$, respectively, to cover the required range of frequencies, including blue, green, and red noises. We vary b from 0 to $m = 63$. The DC p_0 is set to 0. A sample of 100 random power spectra produced by this approach is seen in Fig. B.1.

The generated power spectra are only used to run the method of Leimkühler et al. [2019] to produce a point pattern that is used in the following steps, while the power spectrum can be discarded.

B.1.2 Optimal learning rate details

The best learning rate (LR) λ for a correlation g is the one out of 0.02, 0.01, 0.005, 0.001, 0.0005, 0.0001, and 0.00005 that, when using a fixed number of 1000 iterations to minimize

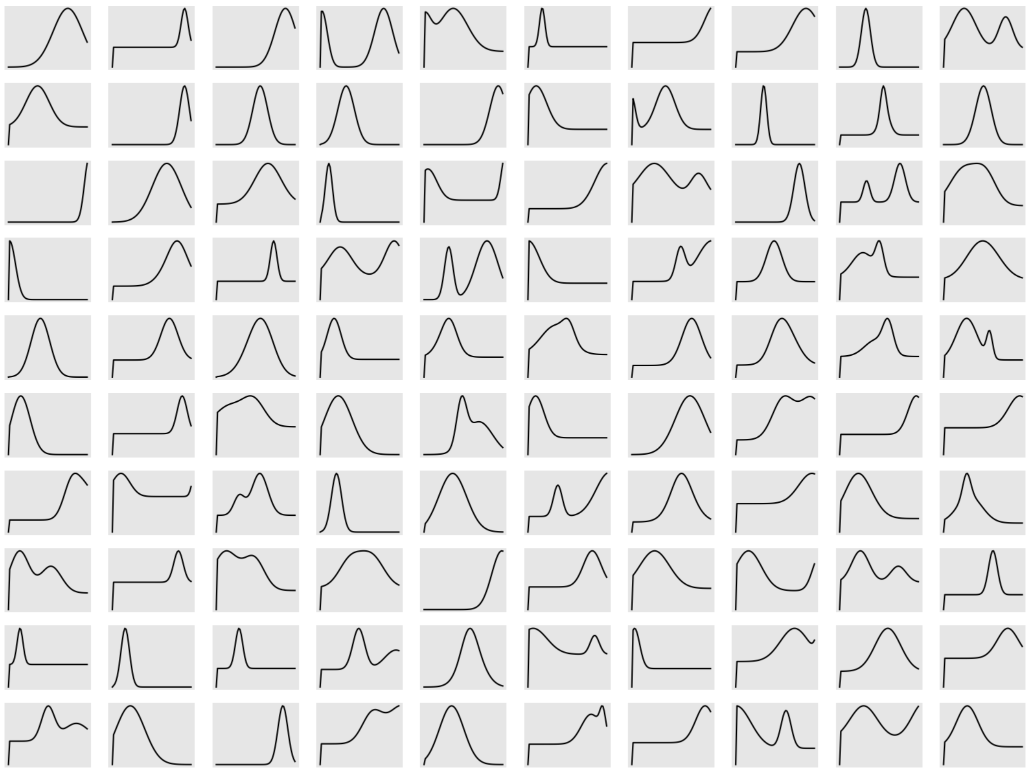


Figure B.1: *Example power spectra used to learn the perceptual embedding.*

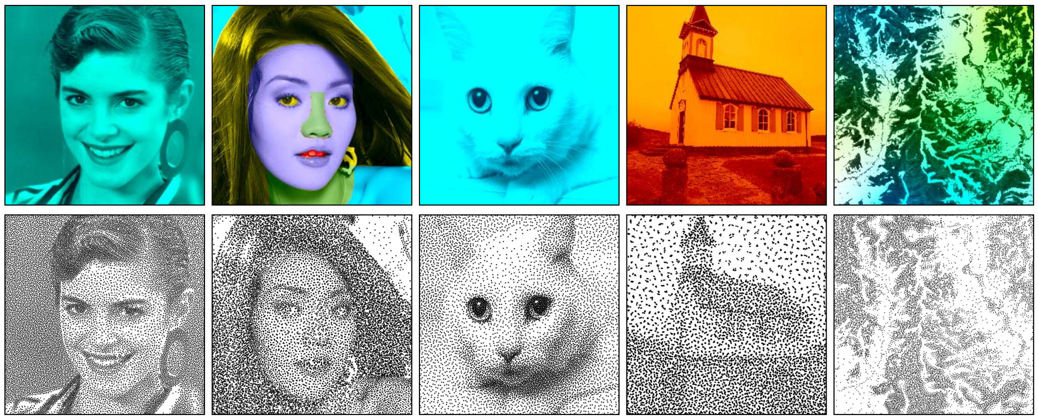


Figure B.2: *Examples of paired training data from human faces, animal faces, churches and tree cover density datasets, respectively.*

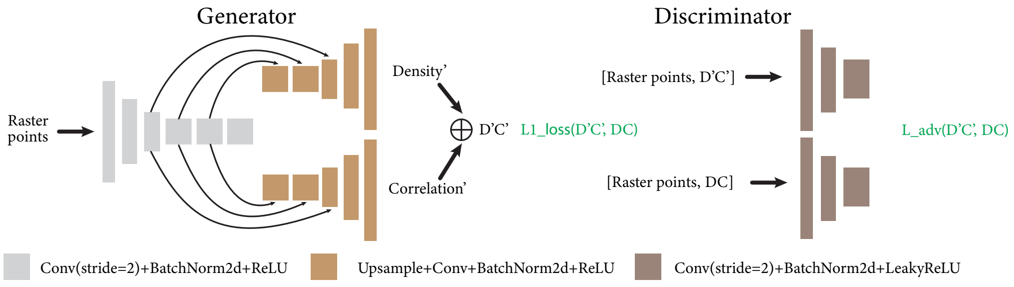


Figure B.3: *Our adapted cGAN architecture.*

PCF error of a randomly initialized pattern, leads to the lowest VGG error. We find these $\{\lambda_i\}$ in a grid search pre-process for the set of all training PCFs $\{g_i\}$.

B.1.3 Data generation for neural network-aided point pattern design

We achieve this by utilizing our proposed latent space and a large set of images to generate a dataset with varying density maps (represented by gray-scale images) and varying correlation maps (represented by different colors). All the density and correlation maps have the same resolution 256×256.

For each density and correlation map of the following datasets, our synthesis is used to generate a point pattern with spatially varying density and correlation to get pairs (a point pattern, a density, and a correlation map) for training our networks. The number of points is computed as $n = 50,000 \times \mathbb{E}_{\mathbf{y}}(\mathbf{1} - d(\mathbf{y}))$. Fig. B.2 shows examples of paired training data from different datasets.

Human faces stippling dataset. We generate the human faces stippling dataset using the face images from Lee et al. [2020]. We use 10,000 gray-scale face images as the density maps. Each of the face images is used to generate two correlation maps, one for uniform correlation with a random chroma assigned to all pixels, and another for spatially-varying correlations. To generate a spatially-varying correlation map, we utilize the facial segmentation masks, including skin, hair, and background, and generate a correlation map by assigning random chroma to each of the segments. In total, we generate 20,000 density and correlation maps.

Animal faces and outdoor churches stippling datasets. Similarly, we use the gray-scale images of animal faces [Choi et al., 2020]. and outdoor churches [Yu et al., 2015a] as density maps for the two datasets. Different from the human faces dataset [Lee et al., 2020], no segmentation masks are provided for animal faces and churches. Therefore, for each density map, we generate a uniform correlation map by randomly sampling a color and assigning it to all pixels. In total, we generate around 15,000 density and correlation maps.

Tree Cover Density dataset for point pattern expansion. We use the Tree Cover Density data [Büttner and Kosztra, 2017] as our density maps. To generate a correlation map for each density map, we generate either a uniform correlation map or a spatially-varying correlation map using anisotropic Gaussian kernels with varying locations, kernel sizes, and orientations. To generate a uniform correlation map, we randomly sample a color and assign it to all pixels. To generate a spatially-varying correlation map, the number of Gaussian kernels is randomly sampled from [2, 16]. For each of the Gaussian kernels, the location (mean) is randomly sampled in [0, 1], the orientation is randomly sampled from [-180, 180] degrees, and the kernel size (variance) for the x-, y-axis are randomly sampled from [0.15, 0.25]. The Gaussian kernels are summed with equal weight to generate a correlation map. In total, we generate around 20,000 density and correlation maps.

B.1.4 Network architecture and training

Fig. B.3 shows the details of our network architecture, adapted from the cGAN [Isola et al., 2017] framework. On the generator side, we take rasterized points as input and output the density and correlation maps in separate branches. The density and correlation predictions are concatenated in the output layer, followed by a Sigmoid function. We use a U-Net architecture with 6 downsampling and upsampling layers with skip connections. For the discriminator, we use a three-layer convolutional architecture to extract patch-based features.

Point patterns are rasterized to a resolution of 256×256 as input. Output is the three-channel image for each rasterized point pattern. Compared with the original framework [Isola et al., 2017], the major change is to use two branches for regressing density and correlation map separately for better quality. Predicted density and correlation are concatenated at the last layer, followed by a Sigmoid function to normalize output values between $[0, 1]$. The network is trained with a combination of L_1 loss and adversarial loss L_{adv} between the output and the ground truth. The total loss $L_{\text{tot}} = L_1 + 0.001 L_{\text{adv}}$ is minimized to update network weights during training. We use the ADAM optimizer [Kingma and Ba, 2014], with an initial learning rate of 0.0001 for both generator and discriminator and a batch size of 8. Learning rate decays by half after every 100 epochs. The network is trained for 400 epochs in 24 hours. With each 256×256 rasterized point image as input, the network inference time is about 0.005 seconds per frame to get the density and correlation map with a resolution of $256 \times 256 \times 3$.

B.2 Additional Results

B.2.1 Latent space

One property about our point correlation embedding space is that we locate some known point correlations to their corresponding semantic colors, including blue, green, pink/red, and step noises generated by Leimkühler et al. [2019]. The found colors can roughly match the semantic meaning, as shown in Fig. B.5. More specifically, we search for those four noises (blue, green, red, and step noises) in our embedding space using Eq. (4.7) in Sec. 4.2. The colors of their nearest neighbors are shown in the second row. The colors are then used to re-synthesize the point sets as shown in the third row.

B.2.2 Ab-initio point pattern design

In Fig. B.4, we compare our $CIELAB$ space representation to traditional approaches [Zhou et al., 2012; Öztireli and Gross, 2012]. Unlike these methods, we do not need to tailor a specific power spectrum or a PCF to represent point correlations, which requires expert knowledge from the end users. Instead, we can simply pick correlations from our correlation palette to design the point correlations and use them to synthesize point patterns. We start with a given density map in Fig. B.4(a). Figure B.4(b) and (c) uses traditional PCF representation. Figure B.4(d) and (e) show our green and blue noise stippling, which requires painting the AB-channel with the specific color (latent coordinate). green and blue noise stippling, which requires painting the AB-channel with the specific color (latent coordinate). We create a spatially-varying point pattern Fig. B.4(f) by simply assigning green color to the face and blue color to the background. Previous methods [Zhou et al., 2012; Öztireli and Gross, 2012] are not able to create point patterns with spatially-varying correlations like ours.

B.2.3 Neural network-aid point pattern design

Ablation study. Here, we study the impact of our network (trained on faces) components. Firstly, we demonstrate that our network is important in terms of density estimation. As shown in Fig. B.6 (first row), one way to estimate density from points is to perform traditional kernel density estimation. However, this can lead to blurry results, given that the number of points is finite. Our network trained on face images, on the other hand, can reconstruct a higher-quality density map. Note that no existing method can perform correlation estimation from points; both density and correlation estimation branches are important in our network. Secondly, we study the impact of using L_{adv} during training. The second row shows that training with L_{adv} can produce sharper density and correlation maps that are used to synthesize points closer to the input compared with training without L_{adv}.

Input points from existing methods. In Fig. B.7, the input points are synthesized using existing methods. We use Zhou et al. [2012] to generate the input points with CCVT profile (in the first row) and Salaün et al. [2022] to generate blue noise face stippling (in the second row). Our network reconstructs the underlying correlation and density, which can be edited to obtain new synthesized points with spatially-varying correlation. Note that our network can only faithfully reconstruct the correlations that are covered by the latent space.

B.2.4 User study

Usefulness experiment. In this experiment, we first explain the concept of density (L-channel) and correlation (AB-channel) to the users so that they are able to pick correlation from our correlation palette and density as otherwise they are able to pick color by switching between the L- and AB-channels, a built-in function of Photoshop that can natively work in LAB mode.

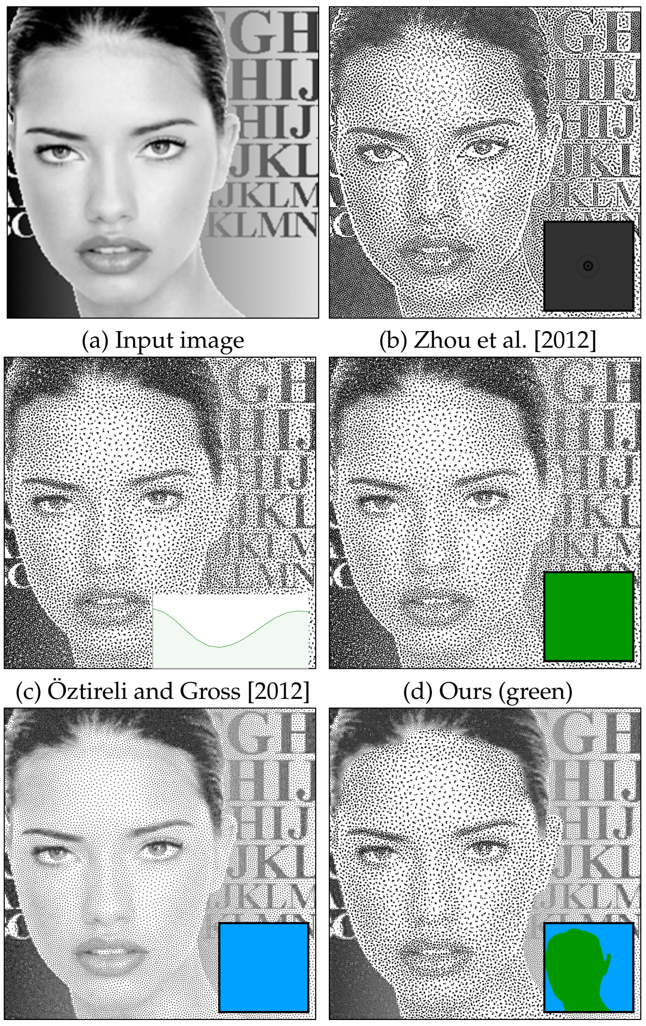


Figure B.4: *Our framework provides a straightforward way to design spatially varying point correlations by picking correlations from our correlation palette. The picked correlations are visualized as chroma in (d), (e), and (f), which automatically find the corresponding PCF from the embedded space.*

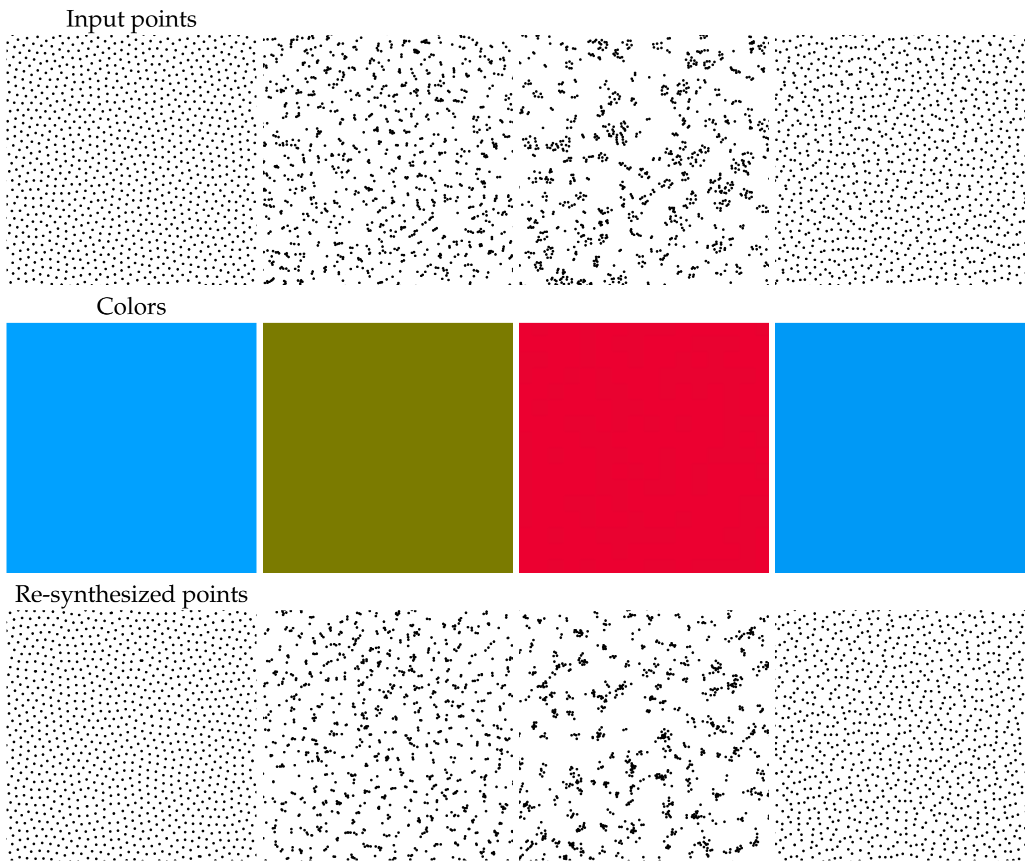


Figure B.5: *We search four known noises (blue, green, red, step noises) generated by Leimkühler et al. [2019], their nearest-neighbors in our embedding space using VGG16 [Simonyan and Zisserman, 2014] based gram metric as shown in Eq.* (4.4)*. The color of their nearest neighbors is shown in the second row. Lastly, we can use our synthesis method to realize similar point patterns.*

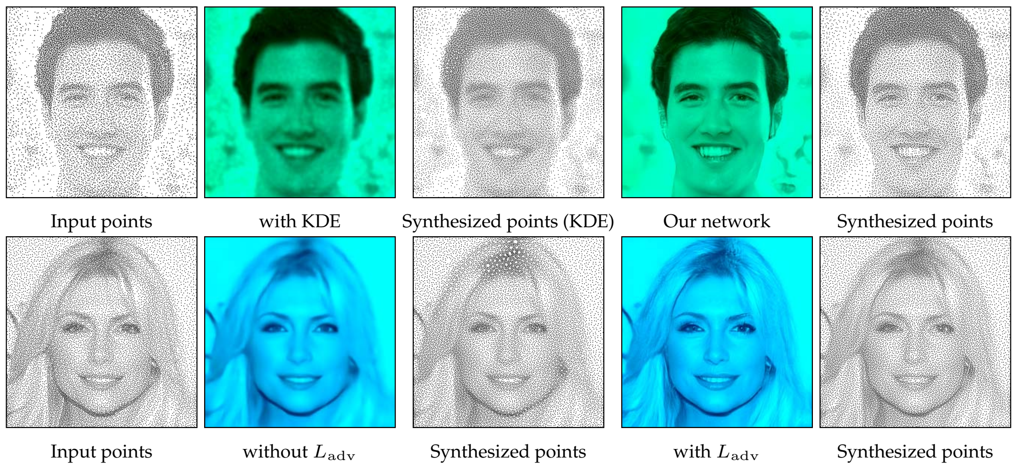


Figure B.6: *In the first row, we analyze the impact of density estimation from a given point pattern. We compare traditional kernel density estimation (KDE) with our neural network-based density reconstruction. In the second row, we study the impact of using different losses during training.*

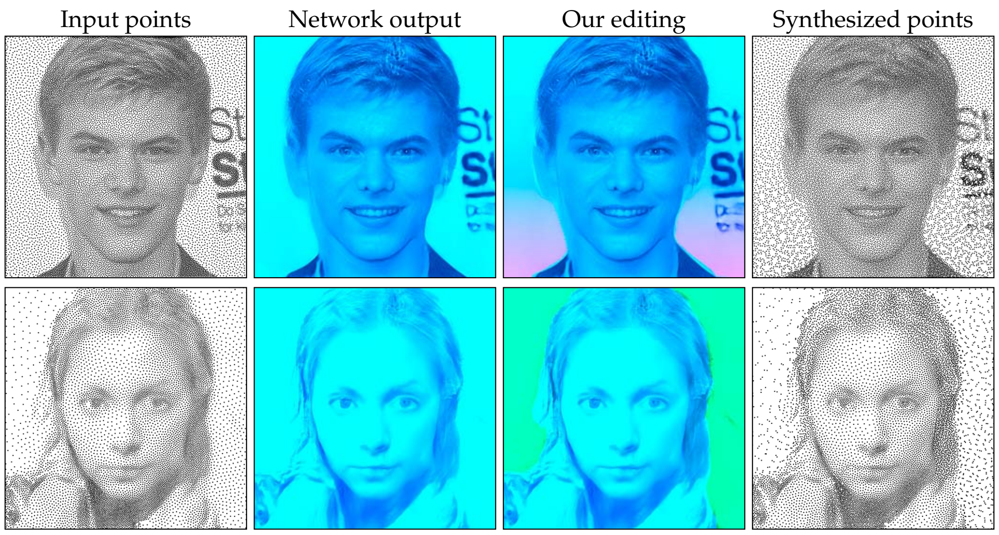


Figure B.7: *We generate the input points using Salaün et al. [2022] (first row) and Zhou et al. [2012] (second row). These point patterns go as input to our network to obtain the underlying density and correlation map (second column). We change the background correlation in both rows (third column), sharpen the density map, and change the hair correlation in the second row, to get edited point patterns (fourth column).*

Appendix C
Blue noise for diffusion models

C.1 Derivation of our backward process

The following shows the derivation of Eq. (5.7) in Sec. 5.1. The goal is to compute $\mathbf{x}_{t-1}$ using $\mathbf{x}_t$ based on the definition of the backward process. We start from:

$$\mathbf{x}_{t-1} = \alpha_{t-1}(L_{t-1}\boldsymbol{\epsilon}) + (1-\alpha_{t-1})\mathbf{x}_0 \tag{C.1}$$

Next, we need to construct $\mathbf{x}_t$ on the right-hand side (RHS) as follows:

$$\begin{aligned}\mathbf{x}_{t-1} &= \alpha_{t-1}(L_{t-1}\boldsymbol{\epsilon}) + (1-\alpha_{t-1})\mathbf{x}_0 \\ &= (\alpha_{t-1}+\alpha_t-\alpha_t)(L_{t-1}\boldsymbol{\epsilon}) + (1-\alpha_{t-1}+\alpha_t-\alpha_t)\mathbf{x}_0 \\ &= \alpha_t(L_{t-1}\boldsymbol{\epsilon}) + (\alpha_{t-1}-\alpha_t)(L_{t-1}\boldsymbol{\epsilon}) + (1-\alpha_t)\mathbf{x}_0 + (\alpha_t-\alpha_{t-1})\mathbf{x}_0\end{aligned} \tag{C.2}$$

The L_{t-1} term can be expanded as:

$$\begin{aligned}L_{t-1} &= \gamma_{t-1}L_w + (1-\gamma_{t-1})L_b \\ &= L_b + \gamma_{t-1}(L_w-L_b) \\ &= L_b + (\gamma_{t-1}+\gamma_t-\gamma_t)(L_w-L_b) \\ &= L_b + \gamma_t(L_w-L_b) + (\gamma_{t-1}-\gamma_t)(L_w-L_b)\end{aligned} \tag{C.3}$$

Based on above and $\mathbf{x}_t = \alpha_t(L_t\boldsymbol{\epsilon}) + (1-\alpha_t)\mathbf{x}_0$, we have:

$$\begin{aligned}\mathbf{x}_{t-1} &= \alpha_t(L_t\boldsymbol{\epsilon}) + \alpha_t(\gamma_{t-1}-\gamma_t)(L_w-L_b)\boldsymbol{\epsilon} + (\alpha_{t-1}-\alpha_t)(L_t\boldsymbol{\epsilon}) \\ &\quad + (\alpha_{t-1}-\alpha_t)(\gamma_{t-1}-\gamma_t)(L_w-L_b)\boldsymbol{\epsilon} + (1-\alpha_t)\mathbf{x}_0 + (\alpha_t-\alpha_{t-1})\mathbf{x}_0 \\ &= \mathbf{x}_t + (\alpha_t-\alpha_{t-1})(\mathbf{x}_0-L_t\boldsymbol{\epsilon}) + (\gamma_{t-1}-\gamma_t)(\alpha_{t-1})(L_w-L_b)\boldsymbol{\epsilon} \\ &= \mathbf{x}_t + (\alpha_t-\alpha_{t-1})(\mathbf{x}_0-L_t\boldsymbol{\epsilon}) + (\gamma_t-\gamma_{t-1})(\alpha_{t-1})(L_b-L_w)\boldsymbol{\epsilon}\end{aligned} \tag{C.4}$$

We ensure that $\alpha_t \geq \alpha_{t-1}$, $\gamma_t \geq \gamma_{t-1}$, $0 \leq \alpha_t \leq 1$, $0 \leq \gamma_t \leq 1$ for different schedulers.

The training procedure of our method is based on the derived backward process. For the terms $L_t\epsilon$, $(L_w - L_b)\epsilon$, when L_w represents the identity matrix, we do not need to actually perform the time-consuming matrix-vector multiplication for $L_w\epsilon$, as in this case $\epsilon = L_w\epsilon$. Therefore, the only term that can introduce overhead is $L_b\epsilon$. We experimentally observed that this overhead is negligible for image resolutions at 64^2, 128^2, 256^2.

Extension to DDIM Here we show that our time-varying noise model can also extend to DDIM, following the procedure shown in Heitz et al. [2023]. First, we define:

$$y_t = (1 - \beta_t)x_0 + \beta_t(L_t\epsilon) \tag{C.5}$$

where y_t, β_t are temporally introduced for easier derivations and will be replaced back by x_t, α_t later. Then, we have:

$$\begin{aligned} y_{t-1} &= (1 - \beta_{t-1})x_0 + \beta_{t-1}(L_{t-1}\epsilon) \\ &= (1 - \beta_{t-1})\frac{y_t - \beta_t(L_t\epsilon)}{1 - \beta_t} + \beta_{t-1}(L_{t-1}\epsilon) \\ &= \frac{1 - \beta_{t-1}}{1 - \beta_t}y_t - \frac{(1 - \beta_{t-1})\beta_t(L_t\epsilon)}{1 - \beta_t} + \beta_{t-1}(L_{t-1}\epsilon) \end{aligned} \tag{C.6}$$

Based on Eq. (C.3), we can expand $\beta_{t-1}(L_{t-1}\epsilon)$ as two terms:

$$\begin{aligned} \beta_{t-1}(L_{t-1}\epsilon) &= \beta_{t-1}(L_b + \gamma_t(L_w - L_b) + (\gamma_{t-1} - \gamma_t)(L_w - L_b)\epsilon \\ &= \beta_{t-1}(L_t\epsilon) + \beta_{t-1}(\gamma_{t-1} - \gamma_t)(L_w - L_b)\epsilon \end{aligned} \tag{C.7}$$

By merging the second and third terms on the right-hand side (RHS) for the expanded version of Eq. (C.6), we can get:

$$\begin{aligned} y_{t-1} &= \frac{1 - \beta_{t-1}}{1 - \beta_t}y_t - \frac{L_t\epsilon(\beta_t - \beta_{t-1})}{1 - \beta_t} + \beta_{t-1}(\gamma_{t-1} - \gamma_t)(L_w - L_b)\epsilon \\ &= \frac{1 - \beta_{t-1}}{1 - \beta_t}(y_t - L_t\epsilon) + L_t\epsilon + \beta_{t-1}(\gamma_{t-1} - \gamma_t)(L_w - L_b)\epsilon \end{aligned} \tag{C.8}$$

Next, we let $\beta_t = \frac{\sqrt{1-\overline{\alpha}_t}}{\sqrt{\overline{\alpha}_t}+\sqrt{1-\overline{\alpha}_t}}$ and $y_t = \frac{x_t}{\sqrt{\overline{\alpha}_t}+\sqrt{1-\overline{\alpha}_t}}$ where $\overline{\alpha}_t = \prod_{s=1}^t \alpha_s$. Lastly, we can derive the backward process as follows:

$$\begin{aligned} x_{t-1} &= \frac{\sqrt{\overline{\alpha}_{t-1}}}{\sqrt{\overline{\alpha}_t}}(x_t - \frac{\sqrt{\overline{\alpha}_t}\sqrt{1 - \overline{\alpha}_{t-1}} - \sqrt{\overline{\alpha}_{t-1}}\sqrt{1 - \overline{\alpha}_t}}{\sqrt{\overline{\alpha}_{t-1}}}L_t\epsilon) \\ &\quad + (\gamma_t - \gamma_{t-1})\sqrt{1 - \overline{\alpha}_{t-1}}(L_b - L_w)\epsilon \end{aligned} \tag{C.9}$$

In this case, the network needs to learn $L_t\epsilon$ and $\sqrt{1 - \overline{\alpha}_{t-1}}(L_b - L_w)\epsilon$.

C.2 Datasets, network architecture and training details

We use the following datasets for unconditional image generation: CelebA (64^2 and 128^2 resolutions, 30,000 training images) [Lee et al., 2020], AFHQ-Cat (64^2 and 128^2 resolutions, 5,153 training images) [Choi et al., 2020], and LSUN-Church (64^2 resolution, 30,000 out of 126,227 training images) [Yu et al., 2015b]. These partitions were set in order to replicate the training conditions of the methods compared here. For conditional image generation, we conduct experiments on image super-resolution using CelebA from 64^2 to 128^2, 32^2 to 128^2, and LSUN-Church from 32^2 to 128^2. For both datasets, we use 25,000 images for training and 5,000 images for evaluation.

We use the diffuser library [von Platen et al., 2022] to build the 2D U-Net Ronneberger et al. [2015] architecture with 6, 7 down- and up-sampling layers with skip connections for image resolution of 64^2, 128^2, respectively. The number of channels we use is (128, 128, 256, 256, 512, 512) for 6 layers and (128, 128, 128, 256, 256, 512, 512) for 7 layers. Self-attention [Vaswani et al., 2017] module is added in the second last of the down-sampling layer and the second of the up-sampling layer.

For network training details, we show the number of epochs and batch size used for training on different datasets for DDIM, IADB, and Ours in Table C.1. Table C.1 also includes the τ (Eq. (5.9) in Sec. 5.1) values we use for all experiments. For the CelebA (64^2) experiment, we use the exact linear scheduler for γ_t.

Table C.1: *Network training details and the choices of τ (Eq. (5.9) in Sec. 5.1) for our method. For the CelebA (64^2) experiment, we use the exact linear scheduler for γ_t. We use the latent diffusion model (LDM) [Rombach et al., 2022] for the AFHQ-Cat (512^2) experiment.*

Dataset	#Epochs	Batchsize	τ
AFHQ-Cat (64^2)	1,000	256	1,000
AFHQ-Cat (128^2)	1,000	128	0.2
AFHQ-Cat (LDM, 512^2)	1,000	256	1,000
CelebA (64^2)	1,000	256	(linear)
CelebA (128^2)	700	128	0.2
LSUN-Church (64^2)	1,000	256	1,000
CelebA ($64^2 \rightarrow 128^2$)	400	128	0.2
CelebA ($32^2 \rightarrow 128^2$)	80	64	0.2
LSUN-Church ($32^2 \rightarrow 128^2$)	80	64	0.2

C.3 Additional results

Gaussian blue noise at different resolutions. Figure C.1 visualizes Gaussian blue noise masks at different resolutions using our padding strategy. For the masks at resolutions 128^2 and 256^2, the seams between the padded 64^2 tiles are not visible, and the property of blue noise is still preserved. In addition, we show in Fig. C.2 that padding/tiling with the same 64^2 Gaussian blue noise mask results in repetitive noise patterns, as well as structural artifacts in the frequency spectra.

Early stopping tests. Though using Gaussian blue noise only leads to worse results in terms of both quantitative and qualitative evaluations, we observe that the image content appears more clear visually in the early time steps. Based on this observation, we perform a study on early stopping where we stop at a specific early time step T_e and reconstruct the final image with one step. As shown in Table C.2, early stopping at $T_e = 200$ gives much better results using Gaussian blue noise only. However, the quality does not improve with more steps but fluctuates in terms of the metrics.

Table C.2: *Early stopping tests using Gaussian noise only and Gaussian blue noise only on AFHQ-Cat*(128^2)*.* T_e *represents the time step we apply early stopping. When* $T_e = 0$*, the model falls back to the full backward process.*

	Ours (Gaussian noise only)			Ours (Gaussian blue noise only)		
T_e	FID (↓)	Precision (↑)	Recall (↑)	FID (↓)	Precision (↑)	Recall (↑)
200	24.10	0.26	0.08	**20.31**	**0.41**	**0.11**
150	**17.39**	**0.47**	**0.13**	22.42	0.33	0.12
0	**10.81**	**0.78**	**0.31**	17.61	0.59	0.18

Table C.3: *Image super-resolution metrics using IADB and Ours. Our method shows consistent improvement over IADB according to the SSIM and PSNR metrics. 2x and 4x represent super-resolution from* 64^2 *to* 128^2 *and* 32^2 *to* 128^2*, respectively.*

	IADB		Ours	
	SSIM (↑)	PSNR (↑)	SSIM (↑)	PSNR (↑)
CelebA (2×)	0.91	30.73	**0.92**	**31.56**
CelebA (4×)	0.76	24.74	**0.77**	**25.03**
LSUN-Church (4×)	0.57	19.46	**0.59**	**20.00**

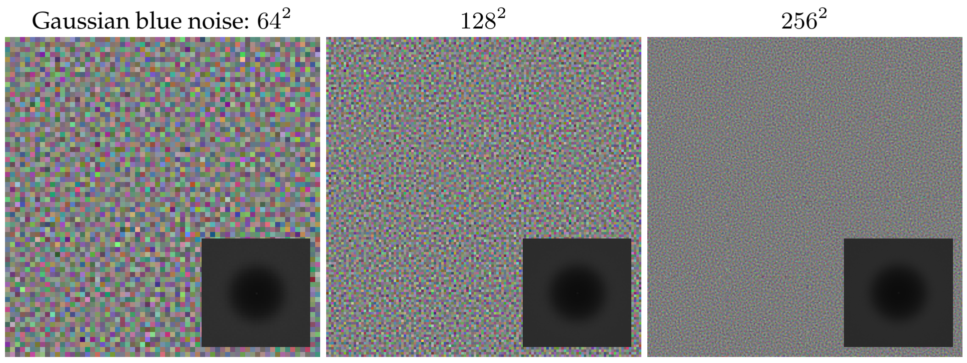


Figure C.1: *Gaussian blue noise masks at different resolutions using our padding strategy and the corresponding frequency power spectra. Padding multiple* (64^2) *tiles does not produce a visible artifact and has an unnoticeable impact on the frequency distribution.*

Nearest neighbors test. We conduct a nearest neighbors test on AFHQ-Cat (64^2) using our method to check if there exists an over-fitting issue. We test on AFHQ-Cat (64^2) as it is the dataset with the smallest training samples and the lowest resolution among our tested datasets. As shown in Fig. C.4, though the nearest neighbors (second to fifth columns) may be semantically similar to the query on the leftmost column, the generated samples are not identical to the training set samples. This means our method does not suffer from overfitting the training data.

Image super-resolution. Table C.3 provides the SSIM and PSNR metrics for the image super-resolution tasks using IADB and Ours. Our method consistently improves over IADB.

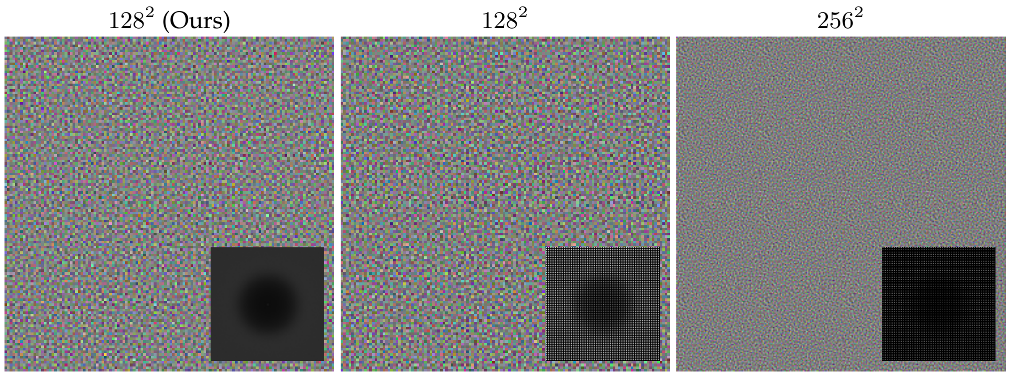


Figure C.2: *Padding/tiling with the same* 64^2 *Gaussian blue noise mask to generate higher-resolution masks results in repetitive noise patterns, as well as structural artifacts in the frequency spectra, compared to our mask shown on the left-most column.*

Image generation. We provide the full Table C.7 for quantitative comparisons, including FID, Precision, and Recall on image generation tasks. More results, comparisons, and interactive visualization can be found in the Supplemental HTML.

Extension to DDIM [Song et al., 2021b]. Detailed derivations of extending our time-varying noise model to DDIM can be found in Appendix C.1. Table C.4 shows preliminary results on AHFQ-Cat (64^2), and Ours (DDIM) gets better FID than the original DDIM.

Extension to LDM [Rombach et al., 2022]. Our time-varying noise model can also be incorporated into the latent diffusion model (LDM) [Rombach et al., 2022] for high-resolution image generation. We compare IADB and Ours used for latent diffusion in Table C.5, and the preliminary results show that Ours gets better FID than IADB. The visual comparisons can be found in Sec. 5.1.

Ablation on γ-scheduler. Table C.6 compares different parameters and functions of our γ-scheduler. For our sigmoid-based γ-scheduler, $\gamma = 0.02$ gives better results than $\gamma = 1000$, showing the importance of choosing the γ value for our γ-scheduler. Using the cosine-based function proposed by Nichol and Dhariwal [2021] resulted in worse FID, showing the importance of choosing the function for our γ-scheduler.

Ablation on noise mask size for padding/tiling. Figure C.5 shows an ablation study on padding/tiling using different Gaussian blue noise sizes (1^2, 4^2, 16^2, 32^2, 64^2) on the AFHQ-Cat (128^2) dataset. Note that 1^2 size is equivalent to using Gaussian (white) noise. We use 64^2 size as our method for all experiments. The figure shows that increasing the size of the Gaussian blue noise mask leads to lower FID than using Gaussian (white) noise in terms of mean values of multiple experiments. However, the standard deviations of using a tiled Gaussian blue noise mask can be higher than using Gaussian (white) noise.

Timing. Here, we present the timing results obtained using a single RTX 2080 NVIDIA GPU for our pipeline. To assess the average inference time for both IADB and our net-

Table C.4: *Preliminary results on comparing DDIM [Song et al., 2021b] and Ours (DDIM) on AFHQ-Cat (64^2).*

Method	DDIM	Ours (DDIM)
FID (↓)	9.82	**7.11**

Table C.5: *Preliminary results on comparing IADB and Ours in the latent diffusion model (LDM) [Rombach et al., 2022] on AFHQ-Cat (512^2).*

Method for LDM	IADB	Ours
FID (↓)	12.19	**11.45**

works, we conducted tests with a batch size of 1 and $T = 250$. The network architectures are identical, except for our network having a 6-channel output instead of 3. The average network inference time is approximately 0.020 seconds for generating a 64^2 image and around 0.023 seconds for a 128^2 image, applicable to both IADB and our method. Regarding noise generation timing, our approach takes roughly 0.0001 seconds to generate a Gaussian blue noise mask at a resolution of 64^2 and about 0.0002 seconds to generate a Gaussian noise or Gaussian blue noise at a resolution of 128^2.

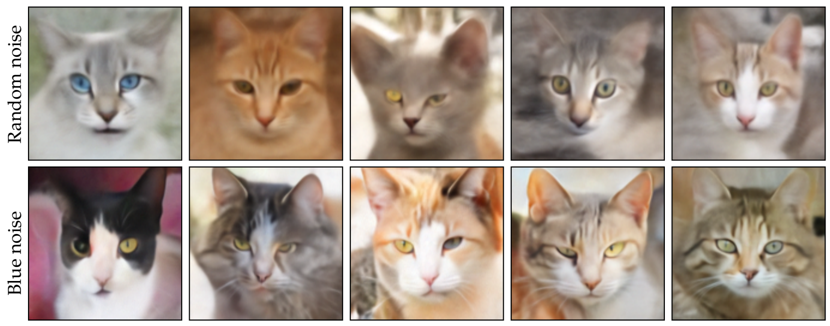


Figure C.3: *Early stopping test on AFHQ-Cat (128^2) using single noise during the training process. The backward process is stopped at $t = 200$ step. The first row shows generated results using Gaussian noise, and the second using Gaussian blue noise. While blue noise alone creates some low-frequency artifacts in the eye region, it generates sharper details than random noise.*

Table C.6: *Comparing the impact of γ-scheduler on CelebA (128^2). Our γ-scheduler with $\gamma = 0.2$ results in lower FID than $\gamma = 1000$ and cosine-based scheduler [Nichol and Dhariwal, 2021].*

γ-scheduler	cosine-based	$\gamma = 1000$	$\gamma = 0.2$
FID (↓)	37.13	29.70	**16.38**

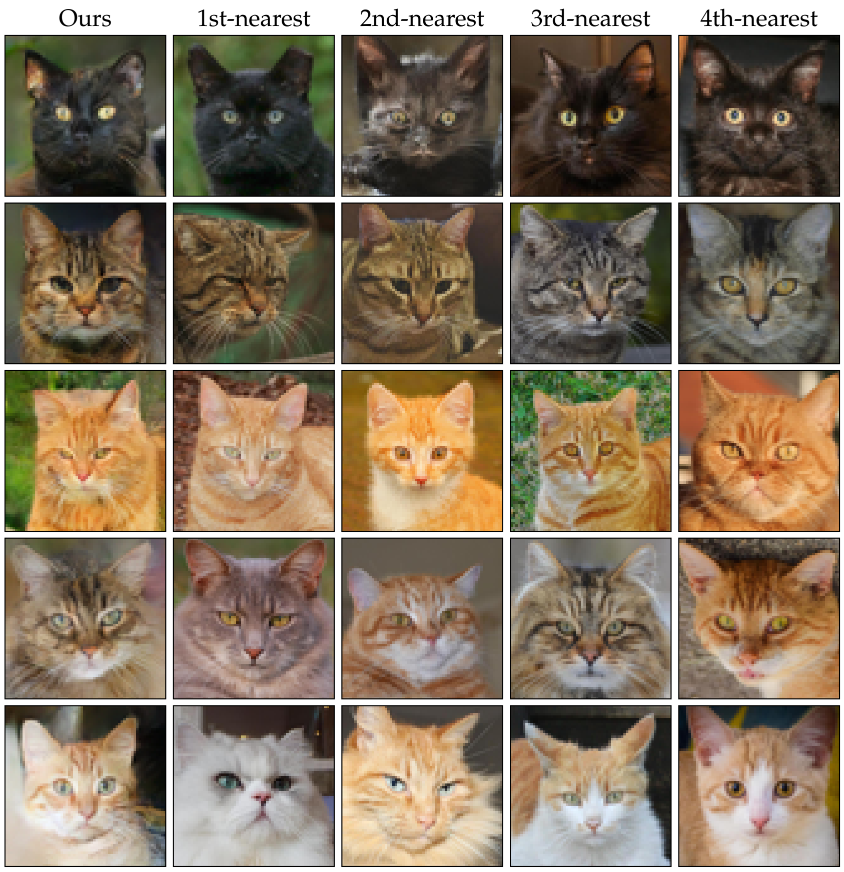


Figure C.4: *We conduct a nearest neighbors test showing our method does not overfit the training data. Generated samples of our method trained on AFHQ-Cat (64^2) are shown in the leftmost column. Training set nearest neighbors are in the remaining columns, ordered from the 1st-nearest neighbor to the 4th-nearest neighbor (based on pixel-wise mean squared distance) from left to right.*

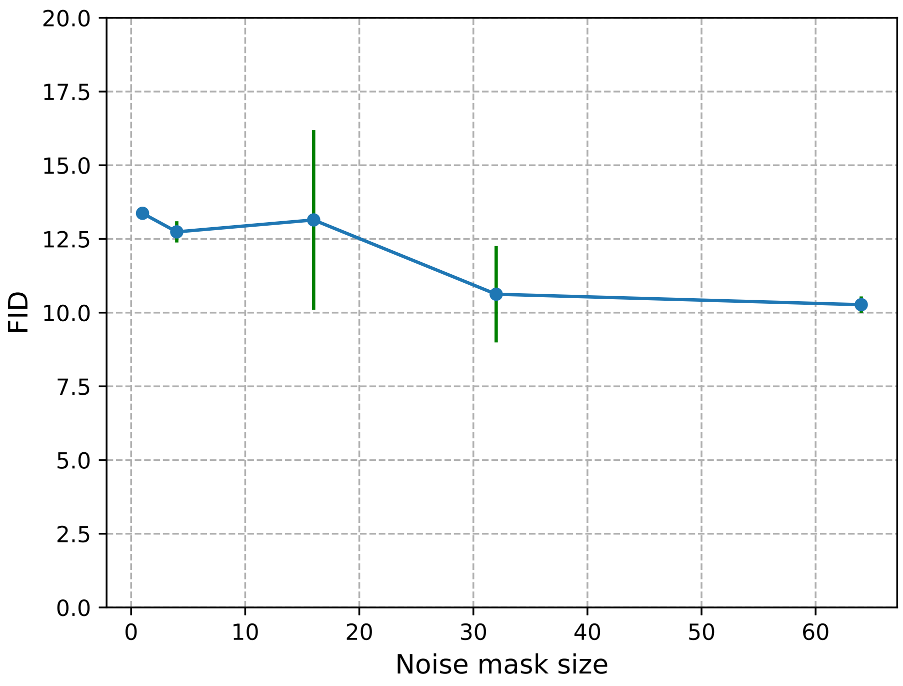


Figure C.5: *Ablation study on different Gaussian blue noise sizes used for padding/tiling* (1^2, 4^2, 16^2, 32^2, 64^2) *experimented on the AFHQ-Cat* (128^2) *dataset. Note that* 1^2 *size is equivalent to using Gaussian (white) noise only. We use* 64^2 *size as our method in all experiments.*

Table C.7: *Quantitative comparisons between DDIM [Song et al., 2021b], IADB [Heitz et al., 2023], and Ours on different datasets, including FID, Precision, and Recall metrics. Our method shows consistent improvements over IADB in terms of FID score. The best score for each metric and dataset is shown in bold, and the second best is underlined.*

	DDIM			IADB			Ours		
	FID (↓)	Precision (↑)	Recall (↑)	FID (↓)	Precision (↑)	Recall (↑)	FID (↓)	Precision (↑)	Recall (↑)
AFHQ-Cat (64^2)	9.82	**0.83**	0.32	9.18	0.72	0.37	**7.95**	0.73	**0.50**
AFHQ-Cat (128^2)	10.73	**0.84**	0.28	10.81	0.78	0.31	**9.47**	0.78	**0.34**
CelebA (64^2)	9.26	0.83	0.46	7.53	0.84	**0.53**	**7.05**	**0.85**	0.52
CelebA (128^2)	**11.92**	**0.81**	**0.48**	20.71	0.75	0.45	16.38	**0.81**	0.42
LSUN-Church (64^2)	16.46	0.74	0.41	13.12	0.71	0.47	**10.16**	**0.76**	**0.48**

Appendix D
Diffusion stereo video generation and restoration

D.1 Additional results

Data augmentation examples. We show in Fig. B.2 that we apply degradations as data augmentation, including blurring, downsampling, adding Gaussian noise, and JPEG compression. The degradation is applied with the same random seed in a temporally-consistent manner to all the frames of the input left-view video.

Robustness to different disparity scaling S. During inference, our method is robust to different disparity scaling S values. Figure D.2 shows that our generated left-view and right-view videos have similar quality under different values of S.

More comparisons for stereo video generation. We show in Fig. D.3 more comparisons regarding stereo video generation. We can observe the same behavior that StereoCrafter [Zhao et al., 2024] and StereoDiffusion [Wang et al., 2024] fail to improve the video quality, while our method is designed for both generation and restoration. Our method shows better visual quality, highlighted in the zoom-in insets.

More comparisons for stereo video generation and restoration. Figure D.4 shows more comparisons regarding stereo video generation and restoration. In this scene, both Real-ESRGAN [Wang et al., 2021] and FMA-Net [Youk et al., 2024] struggle to preserve temporal consistency, e.g., on the textures cropped in the red boxes. Our method also generates sharper edge details around the cloth than other methods.

Moving camera. Though our fine-tuning data does not contain scenes with large camera motion, due to the pretrained network, our method can still generate consistent

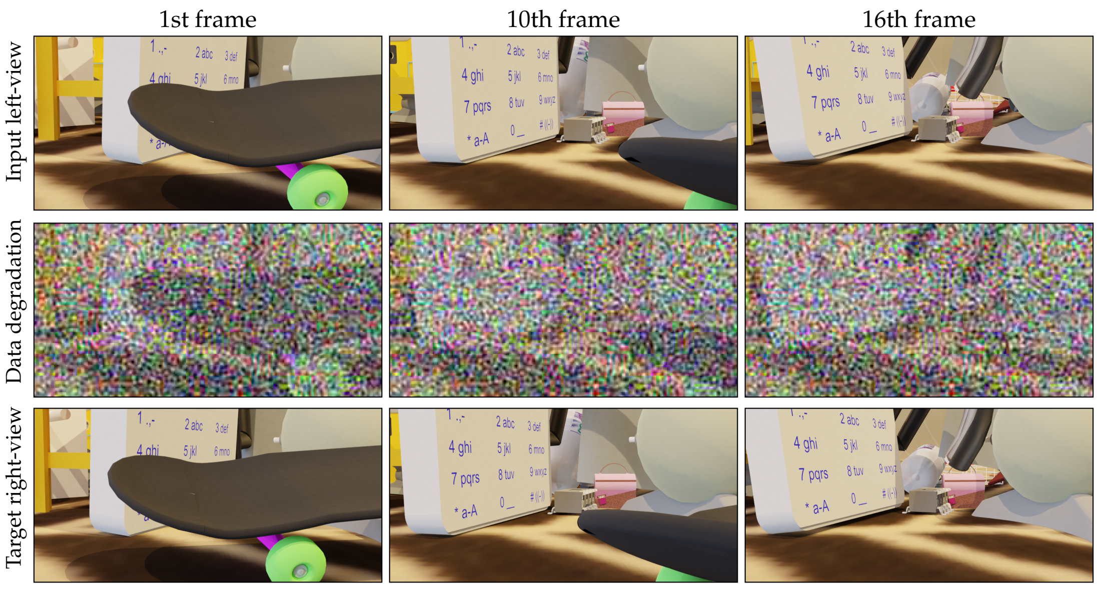


Figure D.1: *Training data augmentation with degradation of blurring, downsampling, adding Gaussian noise, and JPEG compression in a temporally consistent manner with a fixed random seed.*

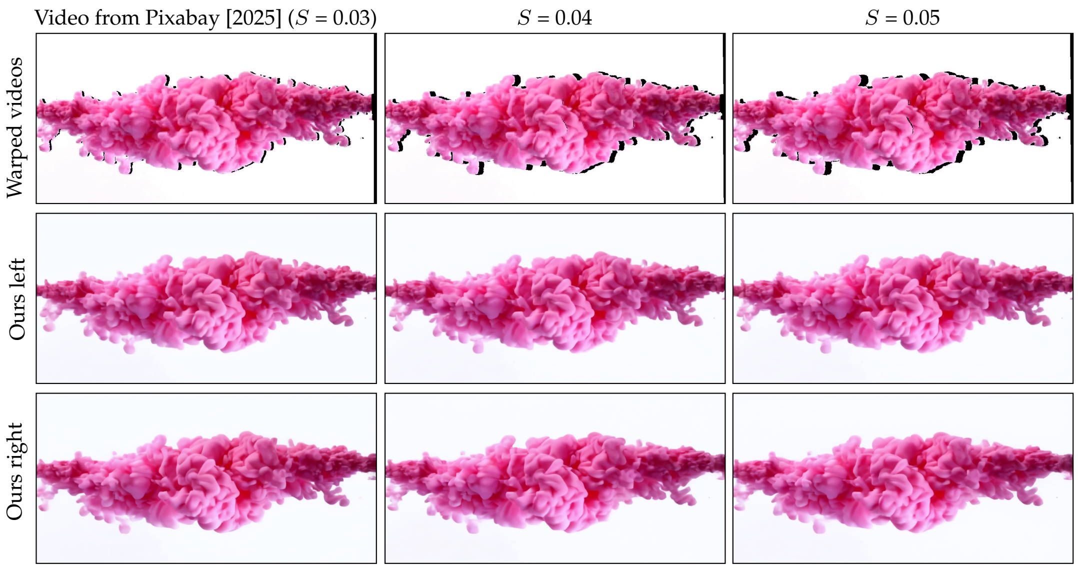


Figure D.2: *Our method is robust to different disparity scaling values S.*

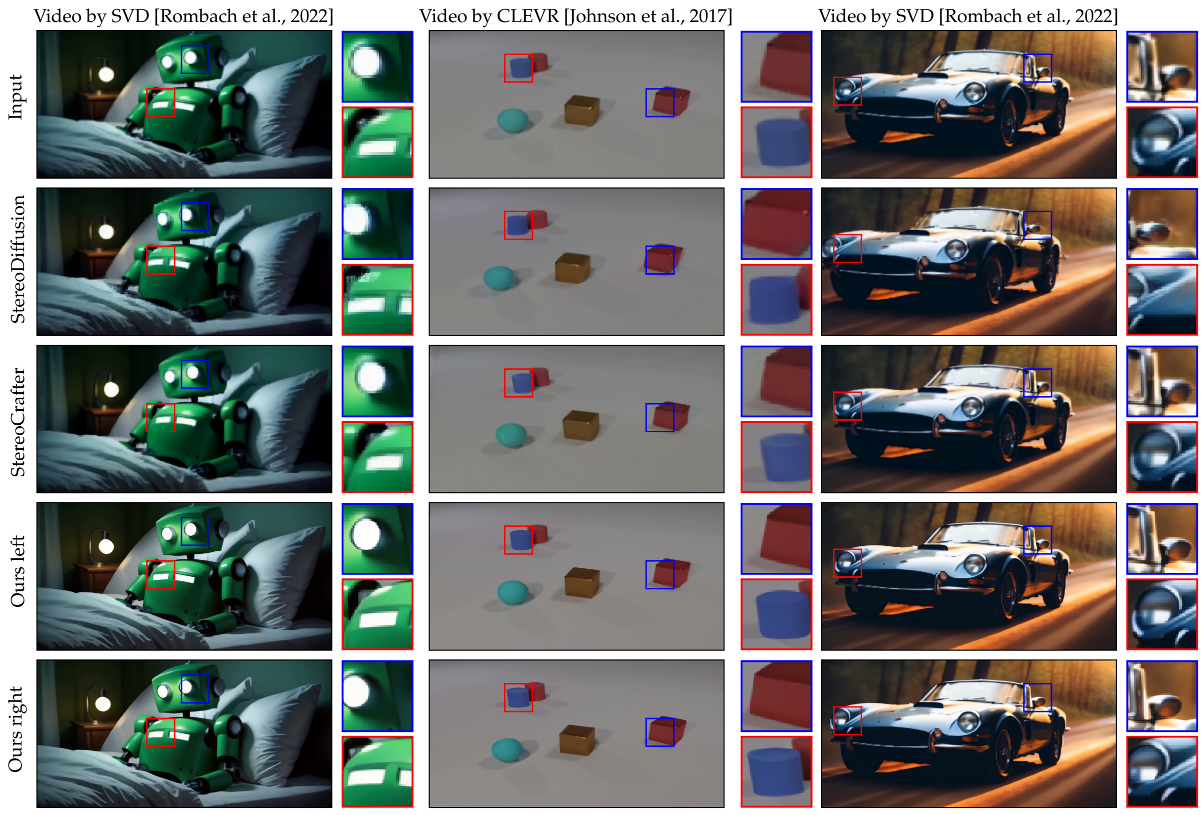

Video by SVD [Rombach et al., 2022]
Video by CLEVR [Johnson et al., 2017]
Video by SVD [Rombach et al., 2022]
Input
StereoDiffusion
StereoCrafter
Ours left
Ours right

Figure D.3: *Stereo generation comparisons between StereoDiffusion [Wang et al., 2024], StereoCrafter [Zhao et al., 2024] and Ours. We show that our method clearly outperforms other methods visually in terms of image quality, highlighted in the zoom-in insets. Input videos are from SVD [Blattmann et al., 2023] and CLEVR [Johnson et al., 2017] downsampled to* 256 × 128, 360 × 180, 360 × 180, *respectively.*

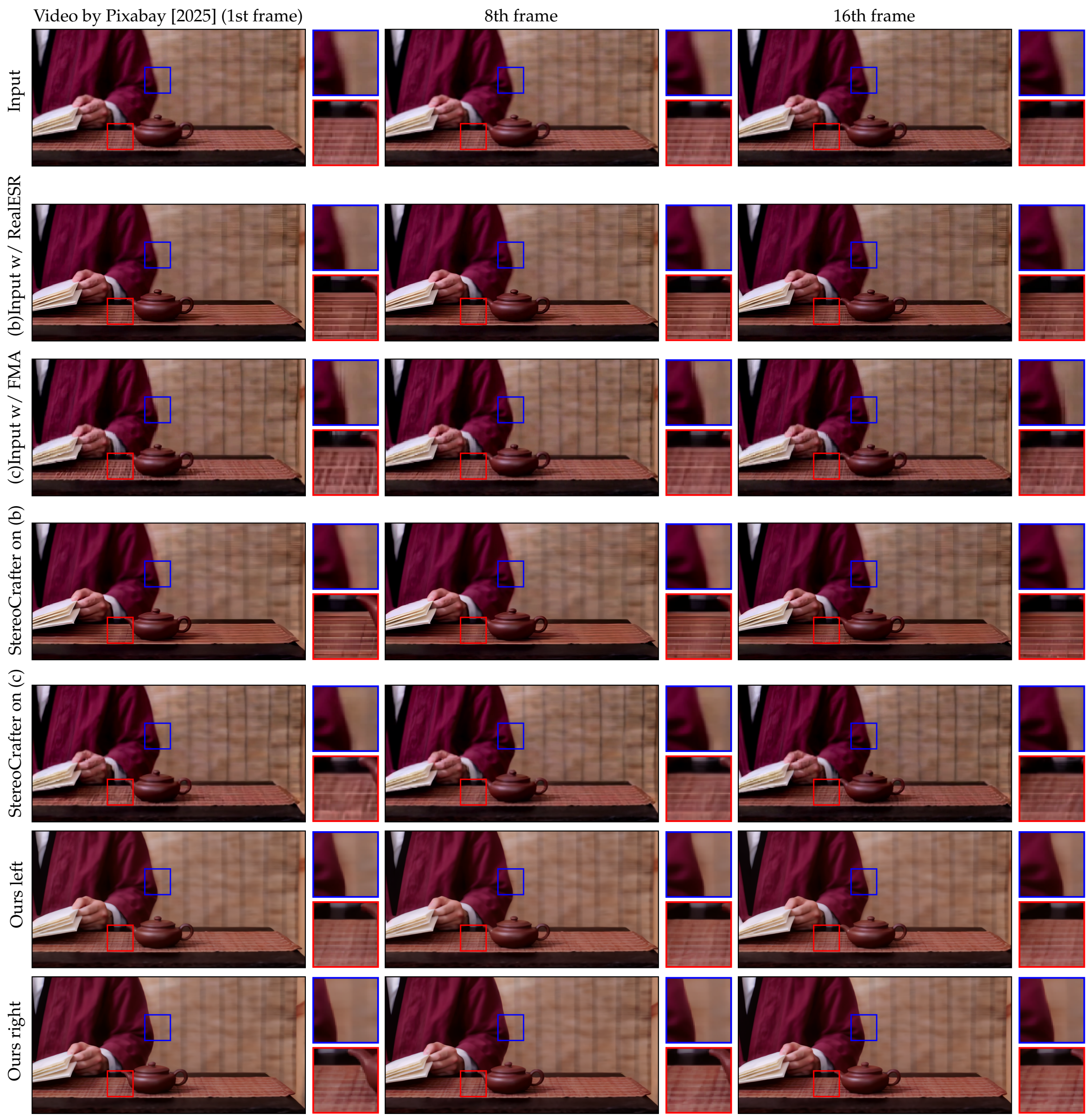


Figure D.4: *Stereo generation and restoration comparisons between StereoCrafter with FMA-Net [Youk et al., 2024], with Real-ESRGAN [Wang et al., 2021], and Ours. We highlight the zoom-in insets that our method generated more temporally-consistent results around the moving textures and the edge of the cloth. The input video is from Pixabay [2025] downsampled to* 320×160.

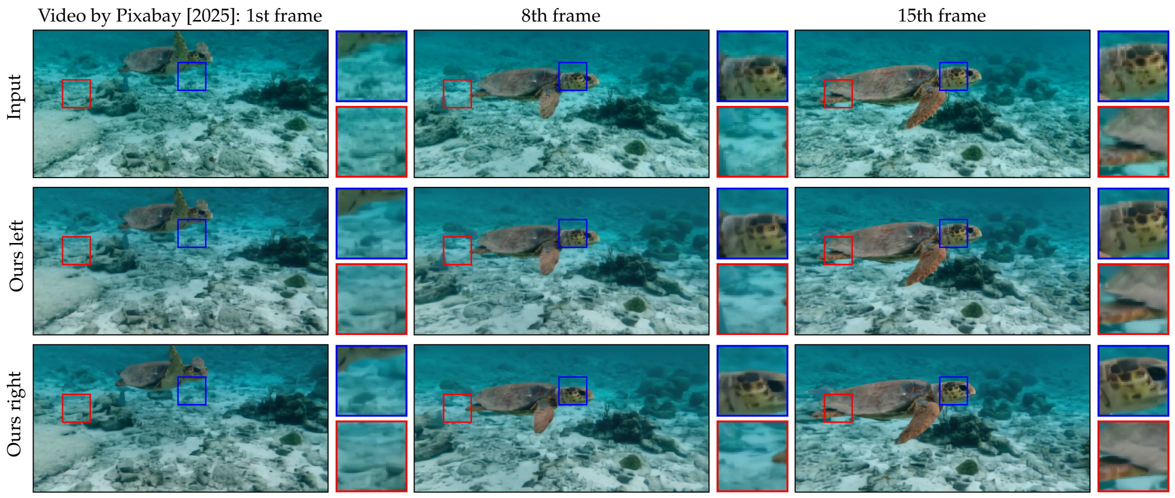


Figure D.5: *This example shows that our method can also generate consistent results for videos with both object and camera movements, though each of the training videos is generated with a fixed stereo camera position. The input video is from Pixabay [2025] downsampled to* 320×160.

results for scenes with both object and camera motions, as shown in Fig. D.5.

Failure case. As our dataset only contains simple shapes and materials from the ShapeNet [Chang et al., 2015] dataset. Our trained model does not generalize well to scenes with highly detailed geometry or appearance, such as the example in Fig. D.6. We believe this can be further addressed by considering more complex geometry and material in our dataset.

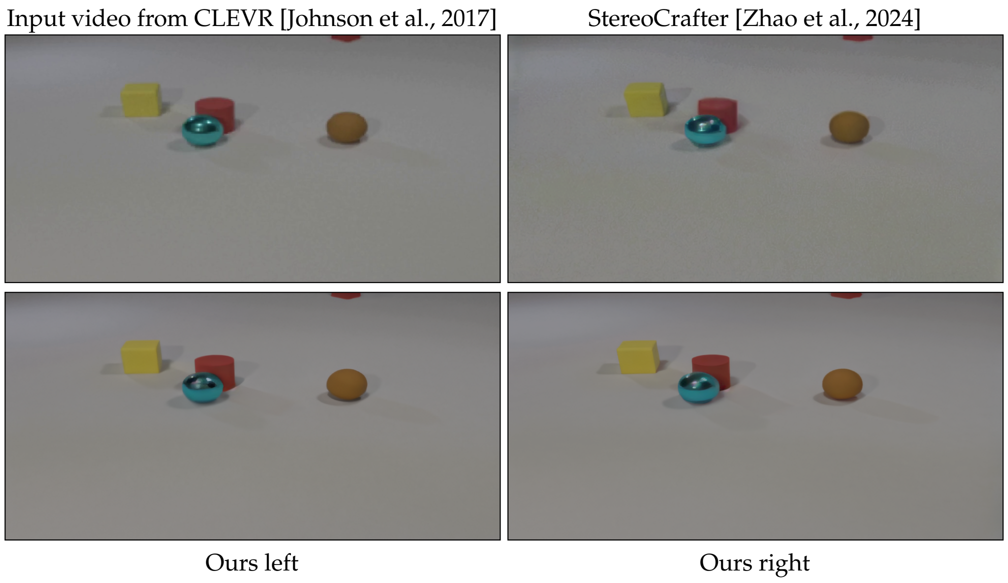

Input video from CLEVR [Johnson et al., 2017]
StereoCrafter [Zhao et al., 2024]
Ours left
Ours right

Figure D.6: *Our method can fail to reproduce the details of specular highlights with highly reflected material, such as the ball in the scene, as our data for fine-tuning does not contain objects with complex material appearance. The input video is from Johnson et al. [2017] downsampled to* 360×180.

Bibliography

David Accurso. 10 tips for stippling art, 2021. URL `https://www.davidaccurso.com/blog/2021/8/4/10-tips-for-stippling-art`. Accessed: 2024-04-29.

Marcin Kluczek, Bogdan Zagajewski, and Tomasz Zwijacz-Kozica. Mountain tree species mapping using sentinel-2, planetscope, and airborne hyspex hyperspectral imagery. *Remote Sensing*, 15(3):844, 2023.

Boyu Wang, Huidong Liu, Dimitris Samaras, and Minh Hoai Nguyen. Distribution matching for crowd counting. *Advances in neural information processing systems*, 33: 1595–1607, 2020.

free-vector. Texture pattern vector. `https://all-free-download.com/free-vector/download/texture_pattern_02_vector_179458.html`, 2024. Accessed: 2024-04-29.

Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1033–1038. IEEE, 1999.

Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Seminal graphics papers: pushing the boundaries, volume 2*, pages 571–576. 2023.

Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, 2000.

Xingchang Huang, Pooran Memari, Hans-Peter Seidel, and Gurprit Singh. Point-pattern synthesis using gabor and random filters. In *Computer Graphics Forum*, volume 41, pages 169–179, 2022.

Xingchang Huang, Tobias Ritschel, Hans-Peter Seidel, Pooran Memari, and Gurprit Singh. Patternshop: editing point patterns by image manipulation. *ACM Transactions on Graphics (TOG)*, 42(4):1–14, 2023.

Riccardo Roveri, A Cengiz Öztireli, and Markus Gross. General point sampling with adaptive density and correlations. In *Computer Graphics Forum*, volume 36, pages 107–117. Wiley Online Library, 2017.

Peihan Tu, Dani Lischinski, and Hui Huang. Point pattern synthesis via irregular convolution. In *Computer Graphics Forum*, volume 38, pages 109–122. Wiley Online Library, 2019.

Xingchang Huang, Corentin Salaun, Cristina Vasconcelos, Christian Theobalt, Cengiz Oztireli, and Gurprit Singh. Blue noise for diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

Pierre Ecormier-Nocca, Pooran Memari, James Gain, and Marie-Paule Cani. Accurate synthesis of multi-class disk distributions. In *Computer Graphics Forum*, volume 38, pages 157–168. Wiley Online Library, 2019.

Ares Lagae and Philip Dutré. Poisson sphere distributions. In *Vision, Modeling, and Visualization*, pages 373–379. Akademische Verlagsgesellschaft Aka GmbH, 2006.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015a.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Omry Sendik and Daniel Cohen-Or. Deep correlations for texture synthesis. *ACM Transactions on Graphics (ToG)*, 36(5):1–15, 2017.

Pradyumna Reddy, Paul Guerrero, Matt Fisher, Wilmot Li, and Niloy J Mitra. Discovering pattern structure using differentiable compositing. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

Juan Camilo Orduz. Sampling from a multivariate normal distribution, March 2019. URL `https://juanitorduz.github.io/multivariate_normal/`. Blog post.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 262–270. Curran Associates, Inc., 2015b.

Ivan Ustyuzhaninov, Wieland Brendel, Leon Gatys, and Matthias Bethge. What does it take to generate natural textures? 2016.

Greg Turk. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 347–354, 2001.

Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. *ACM Transactions on Graphics (TOG)*, 30(4):1–10, 2011.

A Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

Chongyang Ma, Li-Yi Wei, Sylvain Lefebvre, and Xin Tong. Dynamic element textures. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

Riccardo Roveri, A Cengiz Öztireli, Sebastian Martin, Barbara Solenthaler, and Markus Gross. Example based repetitive structure synthesis. In *Computer Graphics Forum*, volume 34, pages 39–52. Wiley Online Library, 2015.

Peihan Tu, Li-Yi Wei, Koji Yatani, Takeo Igarashi, and Matthias Zwicker. Continuous curve textures. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

Pascal Guehl, Rémi Allegre, J-M Dischler, Bedrich Benes, and Eric Galin. Semi-procedural textures using point process texture basis functions. In *Computer Graphics Forum*, volume 39, pages 159–171. Wiley Online Library, 2020.

Lena Gieseke, Paul Asente, Radomír Měch, Bedrich Benes, and Martin Fuchs. A survey of control mechanisms for creative pattern generation. In *Computer Graphics Forum*, volume 40, pages 585–609. Wiley Online Library, 2021.

David J Heeger and James R Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, 1995.

Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

Pierre-Edouard Landes, Bruno Galerne, and Thomas Hurtut. A shape-aware model for discrete texture synthesis. In *Computer Graphics Forum*, volume 32, pages 67–76. Wiley Online Library, 2013.

Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. *Computer Graphics Forum*, 25(3):663–671, 2006. doi: https://doi.org/10.1111/j.1467-8659.2006.00986.x.

Thomas Hurtut, P-E Landes, Joëlle Thollot, Yann Gousseau, Remy Drouillhet, and J-F Coeurjolly. Appearance-guided synthesis of element arrangements by example. In *Proceedings of the 7th International Symposium on Non-photorealistic Animation and Rendering*, pages 51–60, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5099–5108. Curran Associates, Inc., 2017b.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

Thomas Leimkühler, Gurprit Singh, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel. Deep point correlation design. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356562.

Kun He, Yan Wang, and John Hopcroft. A powerful generative model using random weights for the deep image representation. *Advances in Neural Information Processing Systems*, 29, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.

Kfir Aberman, Jing Liao, Mingyi Shi, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. Neural best-buddies: Sparse cross-domain correspondence. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

Robert Ulichney. *Digital Halftoning*. MIT Press, 1987.

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

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

Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. In *SIGGRAPH*, page 69–78, 1985.

Frédo Durand. A frequency analysis of Monte-Carlo and other numerical integration schemes. Technical Report TR-2011-052, MIT CSAIL, 2011.

Gurprit Singh, Cengiz Oztireli, Abdalla G.M. Ahmed, David Coeurjolly, Kartic Subr, Oliver Deussen, Victor Ostromoukhov, Ravi Ramamoorthi, and Wojciech Jarosz. Analysis of sample correlations for monte carlo rendering. *Comp. Graph Form. (Proc. EGSR)*, 38(2), 2019.

Yifan Xu, Tianqi Fan, Yi Yuan, and Gurprit Singh. Ladybird: Quasi-monte carlo sampling for deep implicit field based 3d reconstruction with symmetry. *ECCV*, pages 248–263, 2020.

Cheng Zhang, Cengiz Öztireli, Stephan Mandt, and Giampiero Salvi. Active mini-batch sampling using repulsive point processes. 2019.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3), 2006.

Bernhard Reinert, Tobias Ritschel, and Hans-Peter Seidel. Interactive by-example design of artistic packing layouts. *ACM Transactions on Graphics*, 32(6), 2013. ISSN 0730-0301.

Oliver Deussen, Stefan Hiller, Cornelius Van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. In *Computer Graphics Forum*, volume 19, pages 41–50, 2000.

Adrian Secord. Weighted Voronoi stippling. In *Proc. NPAR*, 2002.

Christoph Schulz, Kin Chung Kwan, Michael Becher, Daniel Baumgartner, Guido Reina, Oliver Deussen, and Daniel Weiskopf. Multi-class inverted stippling. *ACM Transactions on Graphics*, 40(6), 2021. ISSN 0730-0301.

Domingo Martín, Germán Arroyo, Alejandro Rodríguez, and Tobias Isenberg. A survey of digital stippling. *Computers & Graphics*, 67:24–44, 2017. ISSN 0097-8493.

Sung Ye Kim, Ross Maciejewski, Tobias Isenberg, William M. Andrews, Wei Chen, Mario Costa Sousa, and David S. Ebert. Stippling by example. In *Proc. NPAR*, page 41–50, 2009.

Oliver Deussen and Tobias Isenberg. Halftoning and stippling. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, pages 45–61. Springer London, 2013.

Paul Rosin and John Collomosse. *Image and Video-Based Artistic Stylisation*. Springer Publishing Company, Incorporated, 2012.

Daniel L. Lau, Gonzalo R. Arce, and Neal C. Gallagher. Digital halftoning by means of green-noise masks. *J OSA*, 16(7):1575–1586, 1999.

Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. Point sampling with general noise spectrum. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012.

Ares Lagae and Philip Dutre. A comparison of methods for generating poisson disk distributions. *Computer Graphics Forum*, 27(1), 2008.

Li-Yi Wei and Rui Wang. Differential domain analysis for non-uniform sampling. *ACM Transactions on Graphics*, 30(4), 2011.

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

Stuart Lloyd. Least squares quantization in PCM. *IEEE Trans Inform. Theory*, 28(2), 1982.

Michael Balzer, Thomas Schlömer, and Oliver Deussen. Capacity-constrained point distributions: A variant of lloyd's method. *ACM Transactions on Graphics (TOG)*, 28(3): 1–8, 2009.

Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal voronoi tessellation – energy smoothness and fast computation. *ACM Transactions on Graphics*, 28(4), 2009.

Christian Schmaltz, Pascal Gwosdek, Andres Bruhn, and Joachim Weickert. Electrostatic halftoning. *Computer Graphics Forum*, 2010.

Raanan Fattal. Blue-noise point sampling using kernel density model. *ACM Transactions on Graphics*, 30(4), 2011.

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

Daniel Heck, Thomas Schlömer, and Oliver Deussen. Blue noise sampling with controlled aliasing. *ACM Transactions on Graphics (TOG)*, 32(3):1–12, 2013.

Bhavya Kailkhura, Jayaraman J Thiagarajan, Peer-Timo Bremer, and Pramod K Varshney. Stair blue noise sampling. *ACM Transactions on Graphics*, 35(6), 2016.

Hongxing Qin, Yi Chen, Jinlong He, and Baoquan Chen. Wasserstein blue noise sampling. *ACM Transactions on Graphics*, 36(5), 2017. ISSN 0730-0301.

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics*, 23(3), 2004.

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

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

Abdalla GM Ahmed, Hélène Perrier, David Coeurjolly, Victor Ostromoukhov, Jianwei Guo, Dong-Ming Yan, Hui Huang, and Oliver Deussen. Low-discrepancy blue noise sampling. *ACM Transactions on Graphics*, 35(6), 2016.

Abdalla GM Ahmed, Jianwei Guo, Dong-Ming Yan, Jean-Yves Franceschia, Xiaopeng Zhang, and Oliver Deussen. A simple push-pull algorithm for blue-noise sampling. 23(12), 2017.

Tiancheng Sun, Jonathan T. Barron, Yun-Ta Tsai, Zexiang Xu, Xueming Yu, Graham Fyffe, Christoph Rhemann, Jay Busch, Paul Debevec, and Ravi Ramamoorthi. Single image portrait relighting. *ACM Transactions on Graphics*, 38(4), 2019. ISSN 0730-0301.

Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *ICCV*, 2019.

Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proc. SIGGRAPH*, page 417–424, 2000.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, pages 2414–2423, 2016.

Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH*, page 341–346, 2001.

Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics*, 37(4), 2018. ISSN 0730-0301.

Fabio Pellacini and Jason Lawrence. Appwand: Editing measured materials using appearance-driven optimization. *ACM Transactions on Graphics*, 26(3):54–64, 2007. ISSN 0730-0301.

Xiaobo An, Xin Tong, Jonathan D. Denning, and Fabio Pellacini. Appwarp: Retargeting measured materials by appearance-space warping. *ACM Transactions on Graphics*, 30 (6):1–10, 2011. ISSN 0730-0301.

Francesco Di Renzo, Claudio Calabrese, and Fabio Pellacini. Appim: Linear spaces for image-based appearance editing. *ACM Transactions on Graphics*, 33(6), 2014. ISSN 0730-0301.

Adrian Jarabo, Belen Masia, Adrien Bousseau, Fabio Pellacini, and Diego Gutierrez. How do people edit light fields? *ACM Transactions on Graphics*, 33(4), 2014. ISSN 0730-0301.

Šárka Sochorová and Ondřej Jamriška. Practical pigment mixing for digital painting. *ACM Transactions on Graphics*, 40(6):1–11, 2021.

Fabio Pellacini. envylight: An interface for editing natural illumination. 2010.

Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics*, 34(4):1–11, 2015.

Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. Autocomplete element fields. In *Proc. CHI*, pages 1–13, 2020.

Paul Guerrero, Gilbert Bernstein, Wilmot Li, and Niloy J. Mitra. Patex: Exploring pattern variations. *ACM Transactions on Graphics*, 35(4), 2016. ISSN 0730-0301.

Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *ICCV*, pages 4432–4441, 2019.

John Duncan and Glyn W Humphreys. Visual search and stimulus similarity. *Psychological review*, 96(3):433, 1989.

Mark D Fairchild. *Color appearance models*. John Wiley & Sons, 2013.

Chuong H Nguyen, Tobias Ritschel, and Hans-Peter Seidel. Data-driven color manifolds. *ACM Transactions on Graphics*, 34(2):1–9, 2015.

Norbert Lindow, Daniel Baum, and Hans-Christian Hege. Perceptually linear parameter variations. In *Computer Graphics Forum*, volume 31, pages 535–544, 2012.

Louis CW Pols, LJ Th Van der Kamp, and Reinier Plomp. Perceptual and physical space of vowel sounds. *J ASA*, 46(2B):458–467, 1969.

Josh Wills, Sameer Agarwal, David Kriegman, and Serge Belongie. Toward a perceptual space for gloss. *ACM Transactions on Graphics*, 28(4):1–15, 2009.

Philipp Henzler, Niloy J Mitra, , and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *CVPR*, 2019.

Michal Piovarči, David I.W. Levin, Danny Kaufman, and Piotr Didyk. Perception-aware modeling and fabrication of digital drawing tools. 37(4), 2018.

Iliyan Georgiev and Marcos Fajardo. Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342827. doi: 10.1145/2897839.2927430. URL `https://doi.org/10.1145/2897839.2927430`.

Robert Ulichney. Void-and-cluster method for dither array generation. In *Electronic imaging*, 1993. URL `https://api.semanticscholar.org/CorpusID:120266955`.

Eric Heitz and Laurent Belcour. Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames. *Computer Graphics Forum*, 2019. ISSN 1467-8659. doi: 10.1111/cgf.13778.

Vassillen Chizhov, Iliyan Georgiev, Karol Myszkowski, and Gurprit Singh. Perceptual error optimization for monte carlo rendering. *ACM Transactions on Graphics*, 41(3), mar 2022. ISSN 0730-0301. doi: 10.1145/3504002. URL https://doi.org/10.1145/3504002.

Corentin Salaün, Iliyan Georgiev, Hans-Peter Seidel, and Gurprit Singh. Scalable multi-class sampling via filtered sliced optimal transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 41(6), 2022. doi: 10.1145/3550454.3555484.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=PxTIG12RRHS.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021b. URL https://openreview.net/forum?id=St1giarCHLP.

Eric Heitz, Laurent Belcour, and Thomas Chambon. Iterative α-(de) blending: A minimalist deterministic diffusion model. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–8, 2023.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=FjNys5c7VyY.

Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion models. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Ryan Po, Wang Yifan, Vladislav Golyanik, Kfir Aberman, Jonathan T Barron, Amit H Bermano, Eric Ryan Chan, Tali Dekel, Aleksander Holynski, Angjoo Kanazawa, et al. State of the art on diffusion models for visual computing. *arXiv preprint arXiv:2310.07204*, 2023.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023a. URL https://openreview.net/pdf?id=XVjTT1nw5z.

Xingchao Liu, Xiwen Zhang, Jianzhu Ma, Jian Peng, et al. Instaflow: One step is enough for high-quality diffusion-based text-to-image generation. In *The Twelfth International Conference on Learning Representations*, 2023b.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=TIdIXIpzhoI`.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35:26565–26577, 2022.

Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. *arXiv preprint arXiv:2312.02696*, 2023.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 32211–32252. PMLR, 2023. URL `https://proceedings.mlr.press/v202/song23a.html`.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

Alexia Jolicoeur-Martineau, Kilian Fatras, Ke Li, and Tal Kachman. Diffusion models with location-scale noise. *arXiv preprint arXiv:2304.05907*, 2023.

Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *Advances in Neural Information Processing Systems*, 36, 2024.

Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/pdf?id=4PJUBT9f2Ol`.

Hao Phung, Quan Dao, and Anh Tran. Wavelet diffusion models are fast and scalable image generators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10199–10208, 2023.

Sijie Zhao, Wenbo Hu, Xiaodong Cun, Yong Zhang, Xiaoyu Li, Zhe Kong, Xiangjun Gao, Muyao Niu, and Ying Shan. Stereocrafter: Diffusion-based generation of long and high-fidelity stereoscopic 3d from monocular videos. *arXiv preprint arXiv:2409.07447*, 2024.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4): 139–1, 2023.

Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. Dynibar: Neural dynamic image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4273–4284, 2023a.

Yu-Lun Liu, Chen Gao, Andreas Meuleman, Hung-Yu Tseng, Ayush Saraf, Changil Kim, Yung-Yu Chuang, Johannes Kopf, and Jia-Bin Huang. Robust dynamic radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13–23, 2023c.

Lezhong Wang, Jeppe Revall Frisvad, Mark Bo Jensen, and Siavash Arjomand Bigdeli. Stereodiffusion: Training-free stereo image generation using latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7416–7425, 2024.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022.

Peng Dai, Feitong Tan, Qiangeng Xu, David Futschik, Ruofei Du, Sean Fanello, Xiaojuan Qi, and Yinda Zhang. Svg: 3d stereoscopic video generation via denoising frame matrix. *arXiv preprint arXiv:2407.00367*, 2024.

Zhen Li, Cheng-Ze Lu, Jianhua Qin, Chun-Le Guo, and Ming-Ming Cheng. Towards an end-to-end framework for flow-guided video inpainting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17562–17571, 2022.

Shangchen Zhou, Chongyi Li, Kelvin C.K Chan, and Chen Change Loy. ProPainter: Improving propagation and transformer for video inpainting. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2023.

Immersity ai: The ai platform converting images and videos into 3d. `https://www.immersity.ai/`.

Owl3d: Ai-powered 2d to 3d conversion software. `https://www.owl3d.com/`.

Jiale Zhang, Qianxi Jia, Yang Liu, Wei Zhang, Wei Wei, and Xin Tian. Spatialme: Stereo video conversion using depth-warping and blend-inpainting. *arXiv preprint arXiv:2412.11512*, 2024.

Lukas Mehl, Andrés Bruhn, Markus Gross, and Christopher Schroers. Stereo conversion with disparity-aware warping, compositing and inpainting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4260–4269, 2024.

Jian Shi, Zhenyu Li, and Peter Wonka. Immersepro: End-to-end stereo video synthesis via implicit disparity learning. *arXiv preprint arXiv:2410.00262*, 2024a.

Ruiqi Gao, Aleksander Holynski, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul Srinivasan, Jonathan T Barron, and Ben Poole. Cat3d: Create anything in 3d with multi-view diffusion models. *arXiv preprint arXiv:2405.10314*, 2024.

Yiming Xie, Chun-Han Yao, Vikram Voleti, Huaizu Jiang, and Varun Jampani. Sv4d: Dynamic 3d content generation with multi-frame and multi-view consistency. *arXiv preprint arXiv:2407.17470*, 2024.

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.

Basile Van Hoorick, Rundi Wu, Ege Ozguroglu, Kyle Sargent, Ruoshi Liu, Pavel Tokmakov, Achal Dave, Changxi Zheng, and Carl Vondrick. Generative camera dolly: Extreme monocular dynamic novel view synthesis. In *European Conference on Computer Vision*, pages 313–331. Springer, 2024.

Rundi Wu, Ruiqi Gao, Ben Poole, Alex Trevithick, Changxi Zheng, Jonathan T Barron, and Aleksander Holynski. Cat4d: Create anything in 4d with multi-view video diffusion models. *arXiv preprint arXiv:2411.18613*, 2024.

Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1905–1914, 2021.

Jingyun Liang, Yuchen Fan, Xiaoyu Xiang, Rakesh Ranjan, Eddy Ilg, Simon Green, Jiezhang Cao, Kai Zhang, Radu Timofte, and Luc V Gool. Recurrent video restoration transformer with guided deformable attention. *Advances in Neural Information Processing Systems*, 35:378–393, 2022.

Jingyun Liang, Jiezhang Cao, Yuchen Fan, Kai Zhang, Rakesh Ranjan, Yawei Li, Radu Timofte, and Luc Van Gool. Vrt: A video restoration transformer. *IEEE Transactions on Image Processing*, 2024.

Xinqi Lin, Jingwen He, Ziyan Chen, Zhaoyang Lyu, Bo Dai, Fanghua Yu, Yu Qiao, Wanli Ouyang, and Chao Dong. Diffbir: Toward blind image restoration with generative diffusion prior. In *European Conference on Computer Vision*, pages 430–448. Springer, 2024.

Chang-Han Yeh, Chin-Yang Lin, Zhixiang Wang, Chi-Wei Hsiao, Ting-Hsuan Chen, Hau-Shiang Shiu, and Yu-Lun Liu. Diffir2vr-zero: Zero-shot video restoration with diffusion-based image restoration models. *arXiv preprint arXiv:2407.01519*, 2024.

Shangchen Zhou, Peiqing Yang, Jianyi Wang, Yihang Luo, and Chen Change Loy. Upscale-a-video: Temporal-consistent diffusion model for real-world video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2535–2545, 2024.

Kelvin CK Chan, Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. Basicvsr: The search for essential components in video super-resolution and beyond. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4947–4956, 2021.

Kelvin CK Chan, Shangchen Zhou, Xiangyu Xu, and Chen Change Loy. Basicvsr++: Improving video super-resolution with enhanced propagation and alignment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5972–5981, 2022.

Geunhyuk Youk, Jihyong Oh, and Munchurl Kim. Fma-net: Flow-guided dynamic filtering and iterative feature refinement with multi-attention for joint video super-resolution and deblurring. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 44–55, 2024.

John-Peter Lewis. Algorithms for solid noise synthesis. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1989.

Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, 1998.

Robert A Ulichney. Dithering with blue noise. *Proc. IEEE*, 76(1), 1988.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Kwang Moo Yi, Eduard Trulls, Yuki Ono, Vincent Lepetit, Mathieu Salzmann, and Pascal Fua. Learning to find good correspondences. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2666–2674, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. A sliced wasserstein loss for neural texture synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9412–9420, 2021.

Baptiste Nicolet, Pierre Ecormier-Nocca, Pooran Memari, and Marie-Paule Cani. Pair correlation functions with free-form boundaries for distribution inpainting and decomposition. In *Eurographics 2020 short paper proceedings*, 2020.

Steve Strassmann. Hairy brushes. *SIGGRAPH*, 20(4):225–232, 1986.

Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian CNOurless, and David H Salesin. Image analogies. In *SIGGRAPH*, pages 327–340, 2001.

Jiating Chen, Xiaoyin Ge, Li-Yi Wei, Bin Wang, Yusu Wang, Huamin Wang, Yun Fei, Kang-Lai Qian, Jun-Hai Yong, and Wenping Wang. Bilateral blue noise sampling. *ACM Transactions on Graphics*, 32(6):1–11, 2013.

Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Ann. Review Neuroscience*, 24(1), 2001.

Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. AK Peters/crc Press, 2001.

Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics*, 23(3):664–672, 2004.

Hans Knutsson and C-F Westin. Normalized and differential convolution. In *CVPR*, pages 515–523, 1993.

Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Transactions on Graphics*, 26(3):103–113, 2007.

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

ClipDrop. Relight api, 2023. URL `https://clipdrop.co/relight`.

Marc Spicker, Franz Hahn, Thomas Lindemeier, Dietmar Saupe, and Oliver Deussen. Quantifying visual abstraction quality for stipple drawings. In *Proc. NPAR*, 2017.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *CVPR*, 2020.

Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015a.

Corentin Salaün, Iliyan Georgiev, Hans-Peter Seidel, and Gurprit Singh. Scalable multi-class sampling via filtered sliced optimal transport. 41(6), 2022. ISSN 0730-0301.

György Büttner and Barbara Kosztra. Clc2018 technical guidelines. Technical report, European Environment Agency, 2017.

Konrad Kapp, James Gain, Eric Guérin, Eric Galin, and Adrien Peytavie. Data-driven authoring of large-scale ecosystems. *ACM Transactions on Graphics*, 39(6):1–14, 2020.

Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Trans. Graph (Proc. SIGGRAPH Asia)*, 37(5), 2018.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Vikram Voleti, Christopher Pal, and Adam Oberman. Score-based denoising diffusion with non-isotropic gaussian noise models. *arXiv preprint arXiv:2210.12254*, 2022.

Robert Ulichney. The void-and-cluster method for dither array generation. *SPIE MILESTONE SERIES MS*, 154:183–194, 1999.

Abdalla GM Ahmed, Jing Ren, and Peter Wonka. Gaussian blue noise. *ACM Transactions on Graphics (TOG)*, 41(6):1–15, 2022.

Thomas Kollig and Alexander Keller. Efficient multidimensional sampling. In *Computer Graphics Forum*, volume 21, pages 557–563. Wiley Online Library, 2002.

Ting Chen. On the importance of noise scheduling for diffusion models. *arXiv preprint arXiv:2301.10972*, 2023.

Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015b.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. `https://github.com/huggingface/diffusers`, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in Neural Information Processing Systems*, 32, 2019.

George Stein, Jesse Cresswell, Rasa Hosseinzadeh, Yi Sui, Brendan Ross, Valentin Villecroze, Zhaoyan Liu, Anthony L Caterini, Eric Taylor, and Gabriel Loaiza-Ganem. Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. Modelscope text-to-video technical report. *arXiv preprint arXiv:2308.06571*, 2023.

Chen Hou, Guoqiang Wei, Yan Zeng, and Zhibo Chen. Training-free camera control for video generation. *arXiv preprint arXiv:2406.10126*, 2024.

Jian Shi, Qian Wang, Zhenyu Li, and Peter Wonka. Stereocrafter-zero: Zero-shot stereo video generation with noisy restart. *arXiv preprint arXiv:2411.14295*, 2024b.

Qiao Jin, Xiaodong Chen, Wu Liu, Tao Mei, and Yongdong Zhang. T-svg: Text-driven stereoscopic video generation. *arXiv preprint arXiv:2412.09323*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

Pixabay. Pixabay license summary. `https://pixabay.com/service/license-summary/`, 2025. Accessed: 2025-03-03.

Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3749–3761, 2022.

Blender Foundation. Blender. `https://www.blender.org`, 2023. Version 3.4.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Yannick Hold-Geoffroy, Akshaya Athawale, and Jean-François Lalonde. Deep sky modeling for single image outdoor lighting estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6927–6935, 2019.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023.

Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3061–3070, 2015.

Guorun Yang, Xiao Song, Chaoqin Huang, Zhidong Deng, Jianping Shi, and Bolei Zhou. Drivingstereo: A large-scale dataset for stereo matching in autonomous driving scenarios. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 899–908, 2019.

Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12*, pages 611–625. Springer, 2012.

Wenbo Hu, Xiangjun Gao, Xiaoyu Li, Sijie Zhao, Xiaodong Cun, Yong Zhang, Long Quan, and Ying Shan. Depthcrafter: Generating consistent long depth sequences for open-world videos. *arXiv preprint arXiv:2409.02095*, 2024.

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL `https://doi.org/10.7717/peerj.453`.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pengxiang Li, Zhili Liu, Kai Chen, Lanqing Hong, Yunzhi Zhuge, Dit-Yan Yeung, Huchuan Lu, and Xu Jia. Trackdiffusion: Multi-object tracking data generation via diffusion models. *arXiv preprint arXiv:2312.00651*, 2023b.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

SUN Zhengwentai. clip-score: CLIP Score for PyTorch. `https://github.com/taited/clip-score`, March 2023. Version 0.1.1.

Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. Fvd: A new metric for video generation. 2019.

Bastien Doignies, Nicolas Bonneel, David Coeurjolly, Julie Digne, Loïs Paulin, Jean-Claude Iehl, and Victor Ostromoukhov. Example-based sampling with diffusion models. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11, 2023.

Pascal Chang, Jingwei Tang, Markus Gross, and Vinicius C Azevedo. How i warped your noise: a temporally-correlated noise prior for diffusion models. *arXiv preprint arXiv:2504.03072*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.