A Lennard-Jones Layer for Distribution Normalization

Mulun Na *mulun.na@kaust.edu.sa*
Computational Sciences Group
KAUST

Jonathan Klein *jonathan.klein@kaust.edu.sa*
Computational Sciences Group
KAUST

Biao Zhang *biao.zhang@kaust.edu.sa*
Visual Computing Center
KAUST

Wojciech Pałubicki *wojciech.palubicki@amu.edu.pl*
Natural Phenomena Modeling Group
AMU

Sören Pirk *sp@informatik.uni-kiel.de*
Visual Computing and Artificial Intelligence Group
CAU

Dominik L. Michels *dominik.michels@kaust.edu.sa*
Computational Sciences Group
KAUST

Reviewed on OpenReview: *https://openreview.net/forum?id=imGl7xItqQ*

Abstract

We introduce the *Lennard-Jones layer* (LJL) for the equalization of the density of 2D and 3D point clouds through systematically rearranging points without destroying their overall structure (*distribution normalization*). LJL simulates a dissipative process of repulsive and weakly attractive interactions between individual points by considering the nearest neighbor of each point at a given moment in time. This pushes the particles into a potential valley, reaching a well-defined stable configuration that approximates an equidistant sampling after the stabilization process. We apply LJLs to redistribute randomly generated point clouds into a randomized uniform distribution. Moreover, LJLs are embedded in the generation process of point cloud networks by adding them at later stages of the inference process. The improvements in 3D point cloud generation utilizing LJLs are evaluated qualitatively and quantitatively. Finally, we apply LJLs to improve the point distribution of a score-based 3D point cloud denoising network. In general, we demonstrate that LJLs are effective for distribution normalization which can be applied at negligible cost without retraining the given neural network.

Source Code: Upon request, we are happy to share the source code to generate the results presented in this paper. Please contact the first or the last author of this manuscript.

Keywords: Distribution Normalization, Generative Modeling, Lennard-Jones Potential, Particle Simulation, Point Clouds.

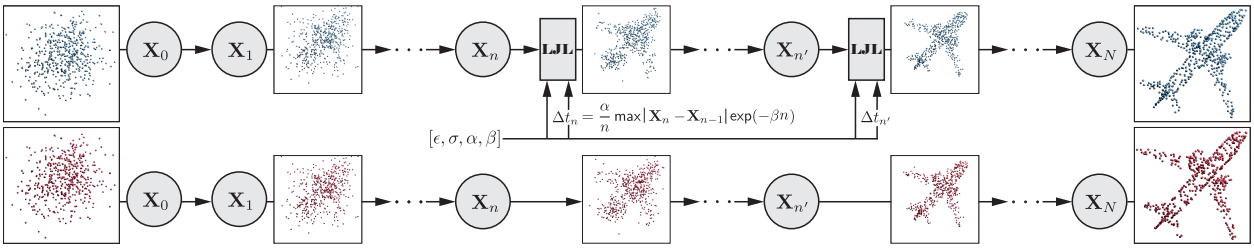


Figure 1: An overview of the integration of LJLs into the inference process of well-trained generative models. A well-trained model generates a meaningful point cloud from random noise in a sequential way (bottom row). LJLs are inserted after certain intermediate-generation steps with damping step size Δt_n and LJ potential parameters ϵ and σ (top row). Embedding LJLs in generative models can improve the generation results by normalizing the point distribution.

1 Introduction

Next to triangle meshes, point clouds are one of the most commonly used representations of 3D shapes. They consist of an unordered set of 3D points without connections and are the native output format of 3D scanners. They also map well to the structure of neural networks, either in generative or processing (e.g. denoising) tasks. A major problem with point clouds is, that their entropy heavily depends on the distribution of the points. In areas with holes, not enough information about the shape is expressed, while clustered areas waste storage without adding additional details. We introduce a so-called *Lennard-Jones layer* (LJL) for distribution normalization of point clouds. We use the term *distribution normalization* to describe the process of systematically rearranging the points' positions in order to equalize their density across the surface without changing the overall shape. More formally, *distribution normalization* describes the property of maintaining a global structure while maximizing minimum distances between points. The Lennard-Jones (LJ) potential is widely considered an archetype model describing repulsive and weakly attractive interactions of atoms or molecules. It was originally introduced by John Lennard-Jones (Jones, 1924a;b; Lennard-Jones, 1931) who is nowadays recognized as one of the founding fathers of computational chemistry. Applications of LJ-based numerical simulations can be found in different scientific communities ranging from molecular modelling (Jorgensen et al., 1996) and soft-matter physics (Al-Raeei & El-Daher, 2019) to computational biology (Hart & Istrail, 1997). For a given point cloud, we aim for a transformation from the initial distribution into a blue noise-type arrangement by simulating the temporal evolution of the associated dynamical system. Each point of the point cloud is interpreted as a particle in a dynamic scene to which the LJ potential is applied as a moving force. In each iteration of the temporal integration process, the point cloud is decomposed into a new set of independent subsystems, each containing a single pair of particles. Each of these subsystems is then simulated individually, which greatly increases stability. As we also include dissipation into our system, the point configuration of the particle system will be stabilized and form randomized uniform point distribution after a sufficient number of iterations. Learning-based point clouds related research has been explored for years, especially in the fields of generation and denoising. Existing models are solely trained to generate and denoise the point cloud without considering the overall point distribution and therefore exhibit the common problems of holes and clusters. By embedding LJLs into well-trained architectures, we are able to significantly improve their results in terms of point distribution while at the same time introducing minimal distortion of the shape as shown in Fig. 1. We also circumvent resource- and time-intensive retraining of these models, as LJLs are solely embedded in the inference process. This way, LJLs are a ready-to-use plug-in solution for point cloud distribution optimization.

2 Related Work

2.1 Molecular Dynamics

Simulating molecular dynamics is an essential tool for applications such as understanding protein folding (Creighton, 1990) or designing modern drugs (Durrant & McCammon, 2011) as well as state-of-the-art

materials (Van Der Giessen et al., 2020). While the motion of atoms and molecules can in principle be obtained by solving the time-dependent Schrödinger equation, such a quantum mechanics approach remains too computationally prohibitive to investigate large molecular systems in practice. Instead, classical molecular dynamics uses Newtonian mechanics treating the nuclei as point particles in a force field that accounts for both, their mutual as well as electronic interactions. This force field is derived from a potential energy function (e.g. the LJ potential) that is formulated either from expectation values of the quantum system or using empirical laws (Rapaport, 2004). Given the typically large number of atoms and molecules involved in molecular dynamics, an analytical solution of the resulting Newtonian mechanical system is usually out of reach. Consequently, numerical methods that evaluate the position of each nucleus at (fixed or adaptive) time intervals are used to find computational approximations to the solutions for given initial conditions (Hochbruck & Lubich, 1999). Classical molecular dynamics simulators are *LAMMPS* Molecular Dynamics Simulator (2014), *RATTLE* (Andersen, 1983) and *SHAKE* (Ryckaert et al., 1977), in which the strong covalent bonds between the nuclei are handled as rigid constraints. Modern approaches allow for the more efficient numerical simulation of large molecular structures as efficiency has further increased, either through algorithmic improvements (Michels & Desbrun, 2015) or by leveraging specialized hardware for parallel computing (Walters et al., 2008). Among others, Alharbi et al. (2017) provided a *Eurographics* tutorial on real-time rendering of molecular dynamics simulations. More recent work also addresses immersive molecular dynamics simulations in virtual (Bhatia et al., 2020; Jamieson-Binnie et al., 2020) and augmented reality (Eriksen et al., 2020).

2.2 Blue Noise Sampling

Colors of noise have been assigned to characterize different kinds of noise with respect to certain properties of their power spectra. In visual computing, blue noise is usually used more loosely to characterize noise without concentrated energy spikes and minimal low-frequency components. Due to its relevance in practical applications such as rendering, simulation, geometry processing, and machine learning, the visual computing community has devised a variety of approaches for the generation and optimization of blue noise. Several methods can maintain the Poisson-disk property, by maximizing the minimum distance between any pair of points (Cook, 1986; Lloyd, 1982; Dunbar & Humphreys, 2006; Bridson, 2007; Balzer et al., 2009; Xu et al., 2011; Yuksel, 2015; Ahmed et al., 2017). Schlömer et al. (2011) introduced farthest-point optimization(FPO) that will maximize the minimum distance of the point set by iteratively moving every point to the farthest distance from the rest of the point set. de Goes et al. (2012) successfully generated blue noise through optimal transport(BNOT). Their method has widely been accepted as the benchmark for best-quality blue noise, which was recently surpassed by Gaussian blue noise(GBN) (Ahmed et al., 2022). GBN is classified as kernel-based methods (Öztireli et al., 2010; Fattal, 2011; Ahmed & Wonka, 2021) that augment each point with a kernel to model its influence. There are several previous works that we consider as methodologically closely related to ours as the authors applied the physical-based particle simulation to synthesize blue noise. The method proposed by Jiang et al. (2015), is based on smoothed-particle hydrodynamics(SPH) fluid simulation which takes different effects caused by pressure, cohesion, and surface tension into account. This results in a less puristic process compared to our work. Both Schmaltz et al. (2010) and Wong & Wong (2017) employ electrical field models, in which electronically charged particles are forced to move towards an equilibrium state after numerical integration because of the electrostatic forces. Adaptive sampling can be achieved by assigning different amounts of electrical charge to particles.

2.3 Generative Models for 3D Point Cloud

Research related to 3D point clouds has recently gained a considerable amount of attention. Some methods aim to generate point clouds to reconstruct 3D shapes from images (Fan et al., 2016) or from shape distributions (Yang et al., 2019). Achlioptas et al. (2017) propose an autoencoder for point cloud reconstruction, while others aim to solve point cloud completion tasks based on residual networks (Xie et al., 2020), with multi-scale (Huang et al., 2020) or cascaded refinement networks (Wang et al., 2020) or with a focus on upscaling (Li et al., 2019). Some methods solve shape completion tasks by operating on point clouds without structural assumptions, such as symmetries or annotations (Yuan et al., 2018), or by predicting shapes from partial point scans (Gu et al., 2020). 3D shape generation methods either use point-to-voxel

representations (Zhou et al., 2021) or rely on shape latent variables to directly train on point clouds (Luo & Hu, 2021a; Zeng et al., 2022), by either employing normalizing flows (Rezende & Mohamed, 2015; Dinh et al., 2017) or point-voxel convolutional neural networks (Liu et al., 2019). ShapeGF (Cai et al., 2020) is an application of score-based generative models (Song & Ermon, 2019) for 3D point cloud generation, which uses estimated gradient fields for shape generation where a point cloud is viewed as samples of a point distribution on the underlying surface. Therefore, with the help of stochastic gradient ascent, sampled points are moving toward the regions near the surface. Luo & Hu (2021a) first attempt to apply denoising diffusion probabilistic modeling (DDPM) (Sohl-Dickstein et al., 2015; Ho et al., 2020) in the field of point cloud generation. Points in the point cloud are treated as particles in the thermodynamic system. During the training process, the original point cloud is corrupted by gradually adding small Gaussian noise at each forward diffusion step, while the network is trained to denoise and reverse this process.

2.4 Denoising Techniques for 3D Point Cloud

Approaches for denoising point clouds aim to reduce perturbations and noise in point clouds and facilitate downstream tasks like rendering and remeshing. Similar to images, denoising for point clouds can be solved with linear (Lee, 2000) and bilateral (Fleishman et al., 2003; Digne & de Franchis, 2017) operators, or based on sparse coding (Avron et al., 2010). Although these methods can generate less noisy point clouds, they also tend to remove important details. Duan et al. (2018) leverages the 3D structure of point clouds by using tangent planes at each 3D point to compute a denoised point cloud from the weighted average of the points. Many existing approaches employ neural network architectures for point cloud denoising purposes, by building graphs or feature hierarchies (Pistilli et al., 2020; Hu et al., 2020), by employing differentiable rendering for point clouds (Yifan et al., 2019), or by classifying and rejecting outliers (Rakotosaona et al., 2020). Hermosilla et al. (2019) propose an unsupervised method for denoising point clouds, which combines a spatial locality and a bilateral appearance prior to mapping pairs of noisy objects to themselves. The majority of learning-based denoising methods directly predict the displacement from noisy point positions to the nearest positions on the underlying surface and then perform reverse displacement in one denoising step. Luo & Hu (2021b) propose an iterative method where the model is designed to estimate a score of the point distribution that can then be used to denoise point clouds after sufficient iterations of gradient ascent. More specifically, clean point clouds are considered as samples generated from the 3D distribution p on the surface, while noise is introduced by convolving p with noise mode n. The score is computed as the gradient of the log-probability function $\nabla \log(p \cdot n)$. They claim that the mode of $p \cdot n$ is the ground truth surface, which means that denoising can be realized by converging samples to this mode.

3 Lennard-Jones Layer

The LJ potential V is defined as

$$V(r) = 4\epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) , \tag{1}$$

which is a function of the distance between a pair of particles simulating the repulsive and weakly attractive interactions among them. The distance r is measured by the Euclidean metric. The first part $(\cdot)^{12}$ attributes the repulsive interaction while the second part $(\cdot)^6$ attributes the soft attraction. With the help of the parameters ϵ and σ, the LJ potential can be adjusted to different use cases. Fig. 2 illustrates the role of these parameters: The potential depth is given by the parameter $\epsilon > 0$ which determines how strong the attraction effect is, and σ is the distance at which the LJ potential is zero.

The corresponding magnitude of the LJ force can be obtained by taking the negative gradient of Eq. (1) with respect to r which is $F_{\boldsymbol{LJ}}(r) = -\nabla_{\boldsymbol{r}} V(r)$. The direction of LJ force on each particle is always along the line connecting the two particles, which point towards each other when two particles are attracted and vice versa. Equilibrium distance $r_{\boldsymbol{E}}$ is where $F_{\boldsymbol{LJ}}(r_{\boldsymbol{E}}) = 0$ and the LJ potential reaches the minimum value. For the distance $r < r_{\boldsymbol{E}}$, the repulsive part dominates, while for $r > r_{\boldsymbol{E}}$ the particles start to attract to each other. When r gets bigger, the attractive force will decrease to zero, which means that when two particles are far enough apart, they do not interact. If we assume a uniform mass distribution among all particles and normalize with the mass, the corresponding equations of motion acting on a pair of particles whose positions

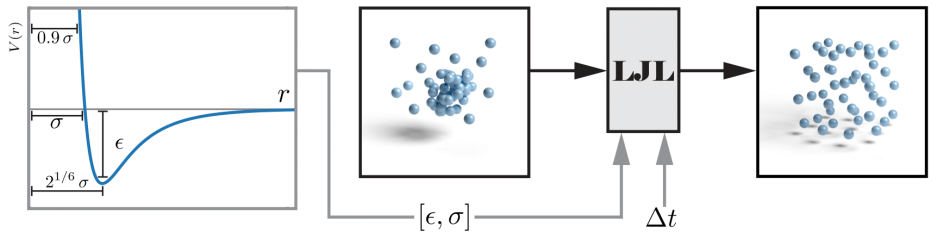


Figure 2: Illustration of the LJ potential and the LJL. The strength and position of the repulsive and weakly attractive zones in the LJ potential are controlled by the hyperparameters ϵ and σ, while Δt controls the damping step size of the LJL. Applying the LJL to a point cloud leads to a more uniform distribution of the points.

are described by $\boldsymbol{x_1}, \boldsymbol{x_2} \in \mathbb{R}^2$ or $\mathbb{R}^3$ can be obtained by

$$r = \|\boldsymbol{x_1} - \boldsymbol{x_2}\|\,, \quad \ddot{\boldsymbol{x}}_1 = F_{\boldsymbol{LJ}}(r) \cdot \frac{\boldsymbol{x_1} - \boldsymbol{x_2}}{r}\,, \quad \ddot{\boldsymbol{x}}_2 = F_{\boldsymbol{LJ}}(r) \cdot \frac{\boldsymbol{x_2} - \boldsymbol{x_1}}{r}\,.$$

Given a 2D or 3D input point cloud in Euclidean space, we aim for a transformation from a white noise- into a blue noise-type arrangement by simulating the temporal evolution of the associated dynamical system. In this regard, we use a formulation of the LJ potential where pairs of particles are grouped as a pair potential to simulate the dynamic LJL procedure. The computations within LJL are summarized in Alg. 1. Within a system containing a number of particles, the potential energy is given by the sum of LJ potential pairs. This sum shows several local minima, each associated with a low energy state corresponding to the system's stable physical configuration. It is an essential aspect of our method, that we do not compute the complete numerical solution of the resulting N-body problem (in which N denotes the number of involved particles). We only take into account the closest neighbor ($k = 1$ neighborhood) of each particle within every iteration of the temporal integration process. Note that being the closest neighbor of a particle is not necessarily a mutual property of two particles within the whole particle system. However, the whole system of particles can be decomposed into several independent subsystems by iteratively assigning the closest neighbor to each particle and, throughout the process, not considering particles which have already been assigned as a closest neighbor. Following this approach, every subsystem just contains a single pair of particles. In each iteration, different subsystems (i.e., new pairs) can be formed. As the subsystems are not interacting within each iteration, the sum of the LJ potentials has just a single (global) minimum if no changes to the nearest neighbors are present, see Appendix A.

LJL creates pairwise independent subsystems using *nearest neighbor search* (NNS) by assigning the nearest points to the current point set $\boldsymbol{X}_i$. We simulate this process by repetitively integrating forward in time using a decaying time step Δt which dampens exponentially according to

$$\Delta t(\cdot) = \alpha \,\mathsf{exp}(-\beta\,(\cdot))\,, \tag{2}$$

in which $\alpha > 0$ refers to the initial step size, and $\beta > 0$ defines the damping intensity. For each pair, the position update is computed independently by integrating the equation of motion numerically using a basic Störmer–Verlet scheme. Within this numerical integration process, we intentionally do not make use of the particles' velocities from the previous step as this corresponds to a velocity reset (setting all velocities identically to zero) causing a desired dissipation effect for the final convergence. Moreover, as $V(r)$ increases rapidly when $r < \sigma$, we clamp the LJ potential from $r = 0.9\sigma$ to $r = 100\sigma$ and employ the hyperbolic tangent as an activation function restricting the gradient $\nabla_{\boldsymbol{r}} V(r)$ to the interval $[-1, 1]$ to avoid arithmetic overflows. Following this method, we can redistribute any randomly generated point cloud into a uniform distribution by iteratively applying LJLs. This process stops as soon as a stable configuration is reached, which means that the difference between the previous and current generated point clouds is below the tolerated threshold. Consequently, this spatial rearrangement of particles prevents the formation of clusters and holes.

ALGORITHM 1: The *Lennard-Jones layer* (***LJL***). NNS denotes the *nearest neighbor search.*

Input: Point cloud $\boldsymbol{X}_i$ and point cloud $\mathsf{NNS}(\boldsymbol{X}_i)$.
Output: Point cloud $\boldsymbol{X}_{i+1}$.
$t_i \leftarrow \Delta t(i)$
$r \leftarrow \mathsf{min}\{\mathsf{max}\{\|\boldsymbol{X}_i - \mathsf{NNS}(\boldsymbol{X}_i)\|, 0.9\,\sigma\}, 100\sigma\}$
$\Delta x \leftarrow \mathsf{tanh}(-\nabla_{\boldsymbol{r}}\, V(r)) \cdot t_i^2 \,/\, 2$
$\boldsymbol{X}_{i+1} \leftarrow \boldsymbol{X}_i + \Delta x \cdot (\boldsymbol{X}_i - \mathsf{NNS}(\boldsymbol{X}_i))/\|\boldsymbol{X}_i - \mathsf{NNS}(\boldsymbol{X}_i)\|$

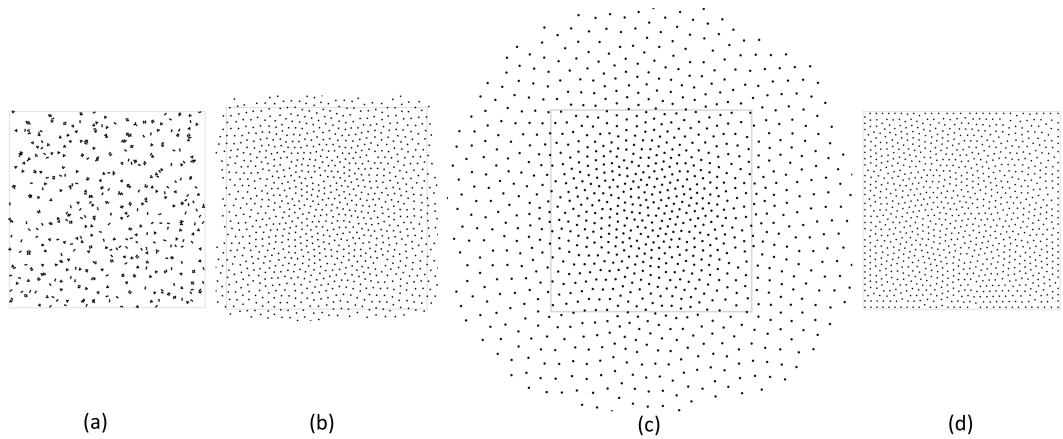


Figure 3: Influence of the parameter σ. The rectangle denotes the unit square in which 1024 points are initialized. (a) $\sigma = 0.2\sigma'$ leads to clusters; (b) $\sigma = \sigma'$ distributes points well and keeps them close to the boundary; (c) $\sigma = 10\sigma'$ spreads points out of the boundary extremely; (d) $\sigma = 10\sigma'$ with fixed boundary conditions. σ' denotes the estimated optimal value of σ.

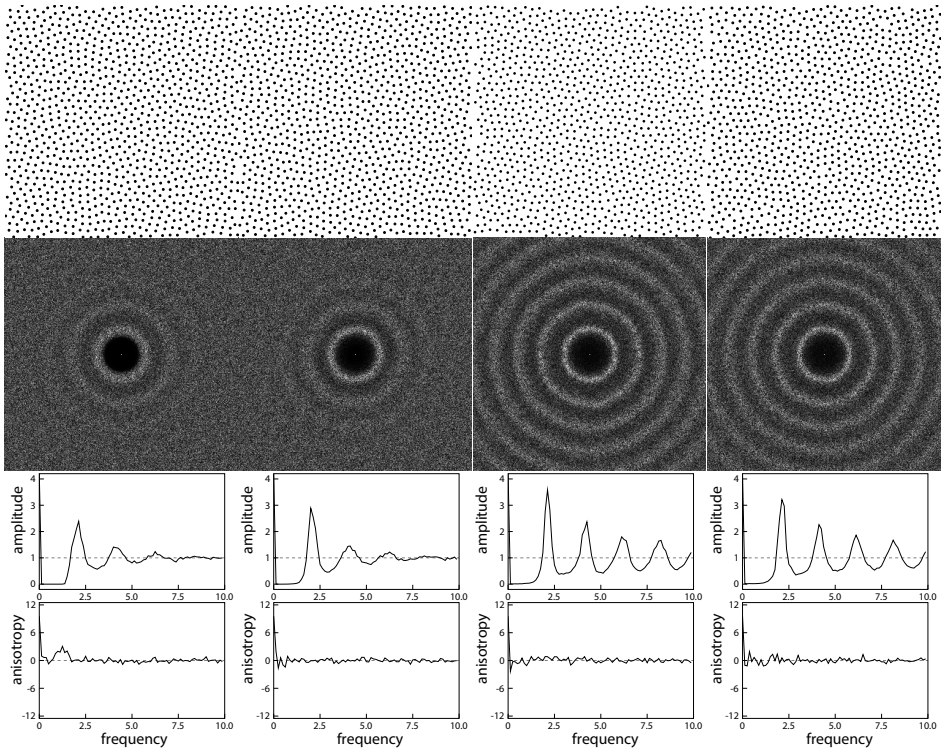
4 Applications

4.1 Numerical Examples on 2D Euclidean Plane

The LJ potential itself has two parameters: The repulsion distance σ and the attraction strength ϵ. Similar to the time step Δt, ϵ scales the gradient, however, it is applied before computing the tanh-function (see Alg. 1). In practice, we find that setting $\epsilon = 2$ results in a well-behaving LJ gradient. The repulsion distance σ corresponds to the optimal distance that all particles strive to have from their neighbors. To derive the proper value for σ, we take inspiration from the point configuration of the hexagonal lattice. The largest minimum distance r between any two points is given by $r = \sqrt{2/(\sqrt{3}\,N)}$ for N points over a unit square (Lagae & Dutré, 2005). Fig. 3 shows the result of different values of σ under identical initial conditions where points are initialized in a unit square. An appropriate σ results in a uniform point distribution that does not exceed the initial square too much. $\sigma' = \sqrt{2/(\sqrt{3}\,N)}$ denotes the estimated optimal value of σ. Small σ's lack redistribution ability over the point set as they lead to a limited effect of LJL on particles. Given a large σ, particles tend to interact with all the points nearby and are spread out of the region excessively. After exploring the effect of σ on the point distribution over the unbounded region, we introduce fixed boundary conditions where points will stop at the boundaries when they tend to exceed. In this constraint scenario, LJL is robust with respect to σ as long as it is sufficiently large. When it comes to the damping step size Δt in Eq. 2, we want LJL updates with decreasing intensity controlled by the damping parameters α and β in order to converge samples in later iterations. Empirically, we set $\alpha = 0.5$ and $\beta = 0.01$ to ensure a proper starting step size and slow damping rate. We analyze the behavior of LJLs in different situations such as considering different numbers of neighbors k and with or without attraction term in LJ potential, see Appendix B.1.

4.1.1 Blue Noise Analysis

For many applications in visual computing, blue noise patterns are desirable. Because LJLs have the ability to normalize the point distribution, we apply LJLs to redistribute samples to randomized uniform distribution



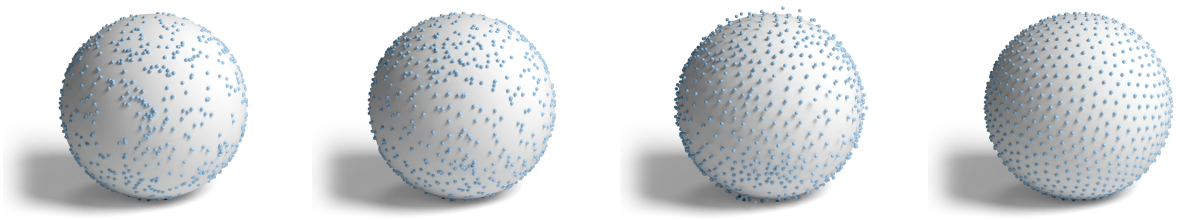
(a) GBN (b) BNOT (c) FPO (d) LJL

Figure 4: Spectral analysis of different blue noise generation methods. The rows from top to bottom display the input point set, power spectrum, radially averaged power spectrum, and anisotropy.

solely from white noise distribution and analyze the corresponding blue noise properties. Here we synthesize the blue nose distribution in a 2D unit square with periodic boundaries. Inside the region, initial random points are generated and iteratively rearranged by LJLs to generate blue noise. The spectral analysis for blue noise includes the power spectrum which averages the periodogram of distribution in the frequency domain and two corresponding 1D statistics: radial power and anisotropy. Radial power averages the power spectrum over different radii of rings and the corresponding variance over the frequency rings is anisotropy indicating regularity and directional bias. We use the point set analysis tool PSA (Schlömer & Deussen, 2011) for spectral evaluation of different 2D blue noise generation methods (1024 points) in Fig. 4. The blue noise generated by the LJL closely resembles the results produced by the FPO method.

4.2 Redistribution of Point Cloud over Mesh Surfaces

As a motivating example for real applications, we use LJLs to evenly distribute points over mesh surfaces. Initially, the point cloud is randomly generated inside a 3D cube $[-1, 1]^3$. To ensure the generality of LJL parameters in 3D cases, mesh sizes are normalized to fit into the cube. The LJL parameters $\alpha = 0.5$, $\beta = 0.01$, and $\epsilon = 2$ are the same as in the previous example. We set $\sigma = 5\sigma'$ due to the addition of the third dimension, with the factor of 5 determined through hyperparameter tuning. In the following four cases illustrated in Fig. 5, we show the importance of applying surface projection in-between LJL iterations. Without the help of LJLs, random points are directly projected on the sphere shown in Fig. 5(a). When LJL is only used once as a pre-processing step before the projection, points are located on the surface but still non-uniformly distributed (see Fig. 5(b)). When using LJL as a post-processing after the projection, additional noise is introduced and points are shifted away from the surface shown in Fig. 5(c). Due to the fact that LJLs iteratively rearrange points, in the last case, we project points to the underlying surface in each intermediate LJL step. The resulting point configuration is not only evenly distributed, but also lies on the sphere shown in Fig. 5(d). The computations involved in LJL-guided point cloud redistribution are summarized in Appendix B.2.



(a) No LJL (b) Pre-processing (c) Post-processing (d) Iterative LJL

Figure 5: Redistribution of random 3D point cloud over a unit sphere. (a) Direct projection of random points. (b) Apply LJL before the projection as pre-processing. (c) Apply LJL after the projection as post-processing. (d) Apply projection in between LJL iterations.

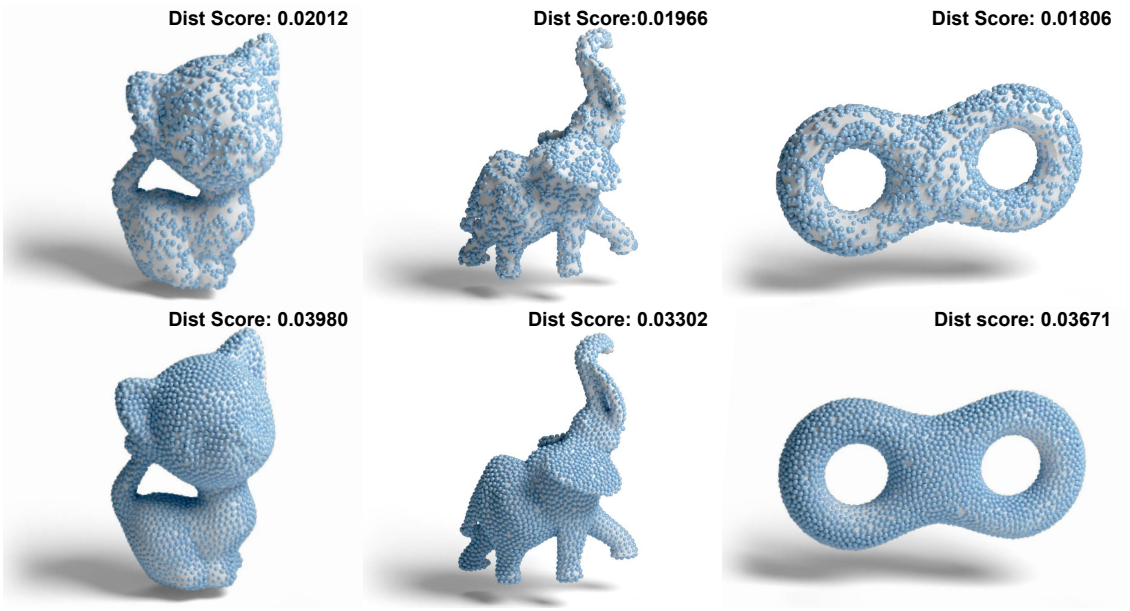


Figure 6: Projecting random 3D point clouds on mesh surfaces with (right) and without (left) LJLs (3000 sample points).

For geometric analysis, we use the mean value of Euclidean distances from each point $\boldsymbol{X_i}$ to its nearest neighbor $\mathsf{NNS}_{\frac{\pi}{4}}$ denoted as *DistanceScore* to evaluate the point distribution of N points shown in Eq. 3. The nearest neighbors are those who satisfy the intersection angles of their normals less than $\frac{\pi}{4}$ to ensure they are on the same side of the surface. A higher distance score indicates the uniform point distribution and fewer holes and clusters in the results. The score is computed by

$$DistanceScore = \frac{1}{N}\sum_{i=0}^{N} \left\| \boldsymbol{X_i} - \mathsf{NNS}_{\frac{\pi}{4}}(\boldsymbol{X_i}) \right\| . \tag{3}$$

The results of LJL-guided redistribution demonstrate significant improvements in point distribution, shown in Fig. 6.

4.3 LJL-embedded Deep Neural Networks

Clusters and holes are consuming the limited number of points without contributing additional information to the result. In the following two applications, LJLs are plugged into the 3D point cloud generation and

denoising networks that work in an iterative way (e.g. Langevin dynamics). With the help of LJLs, we can enforce points to converge to a normalized distribution by embedding LJLs into refinement steps (generation and denoising). The evaluation metric *NoiseScore* is defined as the mean value of the minimum Euclidean distances from the point $\boldsymbol{X_i}$ to the underlying surface $\boldsymbol{M}$, i.e.,

$$NoiseScore = \frac{1}{N}\sum_{i=0}^{N}\min_{x\in M}\|\boldsymbol{X_i} - x\|\,, \tag{4}$$

where $\boldsymbol{x}$ is any point on the surface. We use the mean value of Euclidean distances from each point $\boldsymbol{X_i}$ to its nearest neighbor NNS (Schlömer et al., 2011) denoted as *DistanceScore* to evaluate the point distribution of N points in Eq. 5. We want LJL-embedded results to have as small as possible noise score increments while having large *DistanceScore* increments.

$$DistanceScore = \frac{1}{N}\sum_{i=0}^{N}\|\boldsymbol{X_i} - \mathsf{NNS}(\boldsymbol{X_i})\|\,. \tag{5}$$

4.3.1 LJL-embedded Generative Model for 3D Point Cloud

In this section, we show that LJLs can be applied to improve the point distribution of point cloud generative models ShapeGF (Cai et al., 2020) and DDPM (Luo & Hu, 2021a) without the need to retrain them. It has been shown that the aforementioned two types of probabilistic generative models are deeply correlated (Song et al., 2021). The inference process of both models can be summarized as an iterative refinement process shown in the bottom row of Fig. 1. Initial point cloud $\boldsymbol{X_0}$ is sampled from Gaussian noise $N(0, I)$. The recursive generation process produces each intermediate point cloud $\boldsymbol{X_n}$ according to the previously generated point cloud $\boldsymbol{X_{n-1}}$. Even though there is randomness introduced in each iteration to avoid local minima, the generation process depends heavily on the initial point positions. Every point moves independently without considering the neighboring points, thus generated point clouds tend to form clusters and holes.

The top row of Fig. 1 provides an overview of the integration of LJLs into the inference process of well-trained generative models. Since the sampling procedure of these generative models is performed recursively, one can insert LJLs into certain intermediate steps and guide the generated point cloud to have a normalized distribution. The generator and LJLs have different goals: the former aims to converge all points to a predicted surface ignoring their neighboring distribution, which is also can be seen as a denoising process. The latter aims to prevent the formation of clusters and holes to improve the overall distribution. It is therefore important to observe whether LJLs are inhibiting the quality of the generation process. The trade-off between distribution improvement and surface distortion needs to be measured.

To find LJL parameters that achieve a favorable trade-off, we apply LJLs in the well-trained autoencoder-decoder models. In Fig. 7, given a point cloud $\boldsymbol{Y}$, the autoencoder $\boldsymbol{E}_\theta$ encodes it into latent code Z, then the decoder $\boldsymbol{D}_\theta$ which is the generator will utilize iterative sampling algorithms to reconstruct the point cloud conditioned on Z. We fix the encoder $\boldsymbol{E}_\theta$ and insert LJLs solely in decoder $\boldsymbol{D}_\theta$. To determine proper parameters for LJLs in generation tasks, the reconstructed results need to have a good point distribution and preserve the original shape structures according to distance and noise scores. The computation related to this task is summarized in Alg. 2.

During the generation process, LJLs are activated from *starting steps* (SS) till *ending steps* (T'). This implies that in each valid generation step, the LJL undergoes multiple iterations until convergence. A systematic parameter searching for SS is shown in Appendix C.2.1. It is not necessarily advantageous to perform LJLs at every iteration, since LJLs slow down the convergence of the generation in the beginning steps. LJL-embedding starts in the second half of the generation steps. We also circumvent embedding LJLs in the last few steps to ensure that no extra noise is introduced. Point clouds are normalized into the cube $[-1, 1]^3$, and we set $\epsilon = 2$ and $\sigma = 5\sigma'$ due to the additional third dimension. The decaying time step Δt in i-th generation step is set as

$$\Delta t(i) = \frac{\alpha}{i}\mathsf{max}\|\boldsymbol{X}_i - \boldsymbol{X}_{i-1}\|\mathsf{exp}(-\beta\, i)\,,$$

where $\Delta t(i)$ is scaled by the maximum difference between the current $\boldsymbol{X}_i$ and the previous $\boldsymbol{X}_{i-1}$ point cloud, such that each position modification ΔX caused by LJLs maintains similar scale. Furthermore, $\Delta t(i)$

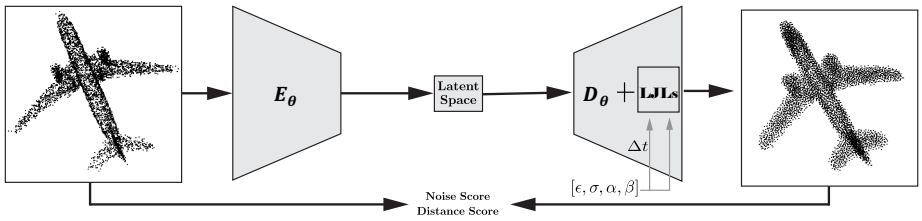


Figure 7: A pipeline for searching LJL parameters through a well-trained autoencoder-decoder.

ALGORITHM 2: *LJL parameter searching via autoencoder $\boldsymbol{E}_\theta$ and decoder $\boldsymbol{D}_\theta$.*

Input: Point cloud $\boldsymbol{Y}$, starting SS and ending $T' < T$ steps.
Output: Reconstructed point cloud $\boldsymbol{X}_T$.

$Z \leftarrow \boldsymbol{E}_\theta(\boldsymbol{Y})$
$\boldsymbol{X}_0 \sim N(0, I)$
FOR $t \leftarrow 1...T$
  IF $SS \leq t \leq T'$:
    $\boldsymbol{X}_t \leftarrow \boldsymbol{D}_\theta(\boldsymbol{X}_{t-1}, Z)$
    $\boldsymbol{X}_t \leftarrow \boldsymbol{LJLs}(\boldsymbol{X}_t, \mathsf{NNS}(\boldsymbol{X}_t))$
  ELSE:
    $\boldsymbol{X}_t \leftarrow \boldsymbol{D}_\theta(\boldsymbol{X}_{t-1}, Z)$
RETURN $\boldsymbol{X}_T$

is divided by the iteration number to diminish the redistribution effect as generation steps increase. In order to determine α and β, we perform a systematic parameter searching based on distance and noise scores, see Appendix C.2.2. We found that $\alpha = 2.5$ and $\beta = 0.01$ meet the requirements of less noise and better distribution.

Incorporating the optimal parameters, we tested LJL-embedded generative models on ShapeNet (Chang et al., 2015) shown in Fig. 8. We evaluate the chair and airplane datasets using DDPM model, while the car dataset is evaluated with ShapeGF model. There is a 99.2% increase in the *DistanceScore* and a 7.8% increase in the noise score for the car dataset. The point distribution is improved by 42.8% while the noise score is only increased by 2.8% for airplanes. Finally, we get a 207.9% increase in *DistanceScore*, and 19.1% increase in the noise score for the chair dataset, shown in Table 1. More details about LJL parameter settings and generation results can be found in Appendix C.2.

4.3.2 LJL-embedded Denoising Model for 3D Point Cloud

Given a noisy point cloud $\boldsymbol{X^0} = \{\boldsymbol{x}_i^N\}$ which is normalized into the cube $[-1, 1]^3$, the score-based model (Luo & Hu, 2021b) utilizes stochastic gradient ascent to denoise it step by step. In each intermediate step, the denoising network $\boldsymbol{S}_\theta(\boldsymbol{X})$ predicts the gradient of the log-probability function (score) which is a vector field depicting vectors pointing from each point position towards the estimated underlying surface. The gradient ascent denoising process will iteratively update each point position to converge to the predicted surface, where we can obtain the denoised result $\boldsymbol{X}^T$. The denoising step can be described as follows:

$$\boldsymbol{X}^{(t)} = \boldsymbol{X}^{(t-1)} + \gamma_t \boldsymbol{S}_\theta(\boldsymbol{X}^{(t-1)}), \;\; t = 1, \ldots, T\,,$$

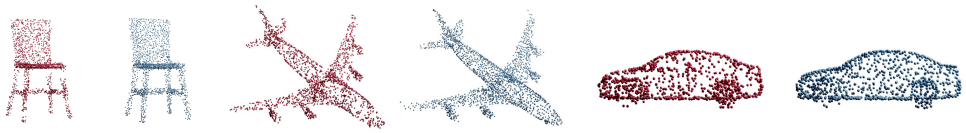


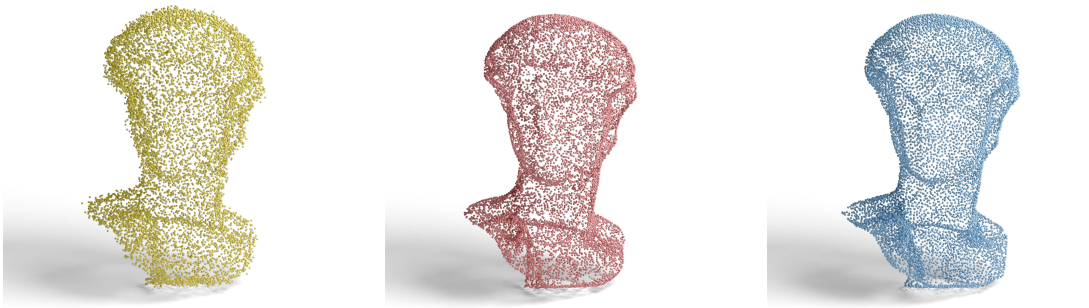
Figure 8: Comparison of generation-only (red) and LJL-embedded generation (blue).

Table 1: Evaluation of LJL-embedded shape generation by measuring the introduced error (*NoiseScore*) and improvement in distribution (*DistanceScore*). Smaller noise and larger distance scores are better.

	Noise Score ↓	**Distance Score** ↑
Airplane	+ 2.8%	+ 42.8%
Chair	+19.1%	+207.9%
Car	+ 7.8%	+ 99.2%

Table 2: Evaluation of LJL-embedded denoising for two different levels of initial noise after different amounts of iterations of denoising steps. Smaller noise and larger distance scores are better.

	Noise Score ↓		**Distance Score** ↑	
Noise Level:	Low	High	Low	High
30 Iterations:	+0.09%	+2.63%	+ 8.79%	+10.71%
60 Iterations:	-0.42%	+1.37%	+13.04%	+16.85%



(a) Noisy input. (b) Denoise-only. (c) LJL-embedded denoising.

Figure 9: Comparison between denoising without and with LJLs.

where γ_t is denoising step size for t-th iteration. LJLs are inserted after each intermediate denoising step except the last few iterations to avoid extra perturbation to the denoised results. LJL parameter settings and evaluation metrics used in this task are inherited from previous generation tasks, where $\sigma = 5\sigma'$ and $\epsilon = 2$. We keep $\beta = 0.01$ and choose $\alpha = 0.3$ which will minimize the ratio of the noise and distance score increment rates, see Appendix C.1.1. The only difference is that the position update scale ($\max \|\boldsymbol{X}_i - \boldsymbol{X}_{i-1}\|$) for the denoising step is different from that in the generation task. The denoising results then do not only converge to the predicted surface with less distortion but are also distributed uniformly as shown in Fig. 9.

We continue to determine appropriate denoising iterations and the performance of denoising models (with and without LJLs) on different noise scales (more and less) shown in Fig. 10. We apply isotropic Gaussian noise with a higher noise scale of 3% and a lower noise scale of 1% of the shape's bounding sphere radius as the standard deviation. This is because denoising tends to progressively smooth the surface and wash out the details. This phenomenon is shown in Fig. 11.

It is clear that no matter the noise scales, for the denoise-only model, the noise score increases when the number of iterations goes up. While for the LJL-embedded denoising model, the noise score remains steady for the first 60 denoising iterations and the distance scores of both models decrease when iteration times increase regardless of noise scales. We further evaluate the LJL-embedded denoising model on the test set from Luo & Hu (2021b) shown in Table 2. By combining LJLs with the denoising network, we find that it improves *DistanceScore* with negligible increments of *NoiseScore*. Moreover, we boost the performance of existing denoising models without extra training. More LJL-embedded denoising results and evaluations are shown in Appendix C.1.

5 Conclusion

In this contribution, we have extended the concept of the LJ potential describing pairwise repulsive and weakly attractive interactions of atoms and molecules to a variety of tasks that require equalizing the density of a set of points without destroying the overall structure. It has been shown that LJLs are conceptually and

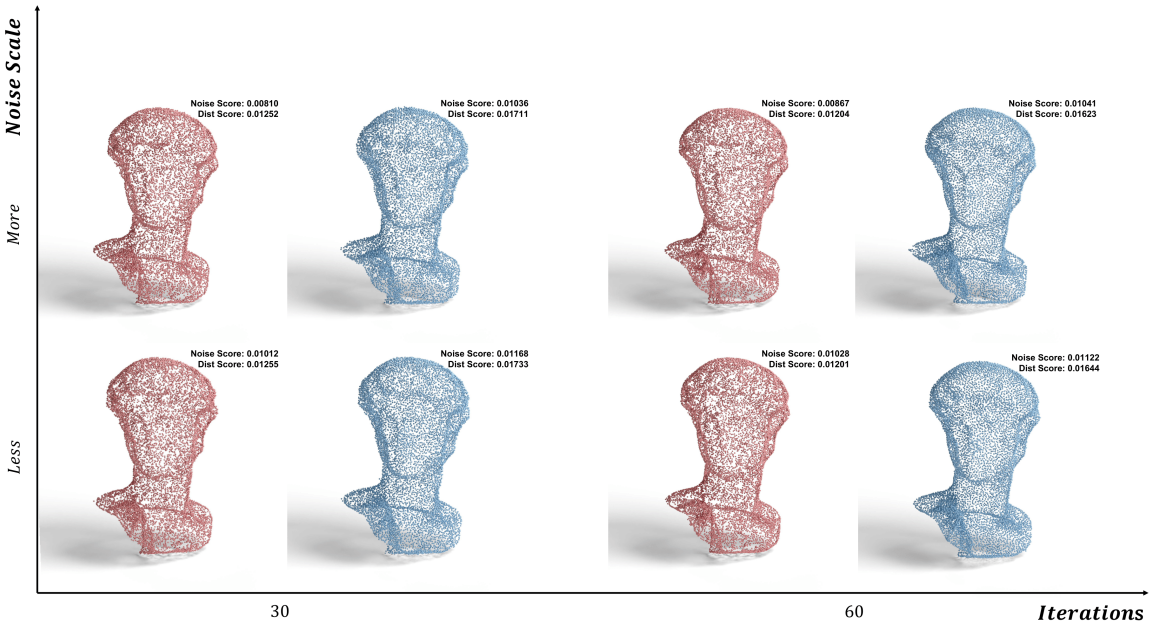


Figure 10: Comparison of denoise-only (red) and LJL-embedded denoising (blue) for 30/60 denoising iterations and different input noise scales.

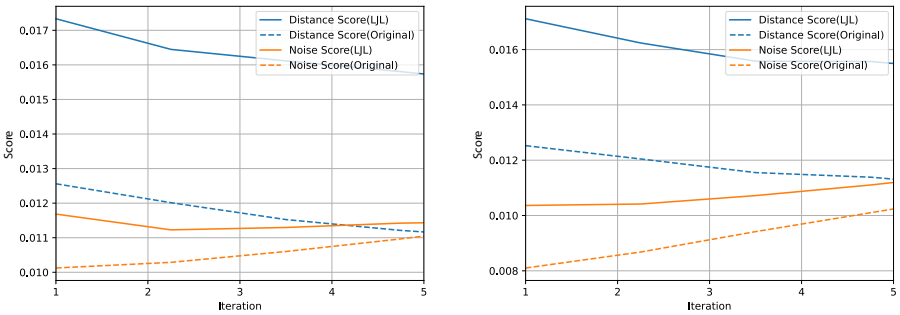


Figure 11: (a) Denoising results of lower noise scale point clouds with respect to increasing denoising iterations. (b) Denoising results of higher noise scale point clouds with respect to increasing denoising iterations.

practically capable of generating high-quality blue noise distributions. To demonstrate the benefit of applying LJLs in the context of the 3D point cloud generation task, we incorporated LJLs into the generative models ShapeGF (Cai et al., 2020) and DDPM (Luo & Hu, 2021a) aiming for the generation of point clouds with an improved point distribution. By embedding LJLs in certain intermediate-generation steps, the clusters and holes are decreased significantly because LJLs increase the chances of points converging to cover the entire shape. Finally, we combine LJLs with a score-based point cloud denoising network (Luo & Hu, 2021b) such that the noisy point clouds are not only moved towards the predicted surfaces but also maintain the uniform distribution. Consequently, this rearrangement prevents the formation of clusters and holes. Since adding LJLs in these cases does not require retraining the network, the cost of applying LJLs is negligible.

The presented work offers several avenues for future work. From an algorithmic perspective, the k-nearest neighbor classification performance could be improved through supervised metric learning methods such as neighborhood components analysis and large margin nearest neighbor. The distribution normalization on surfaces could be addressed by means of considering tangential bundles. By doing this, we force particles not to leave tangential manifolds during LJL interactions and in a perfect case to move solely along the geodesics of the implied surfaces. This would require that we do not distort the underlying surfaces while performing distribution normalization. Geodesical distances between points on surfaces also refer to a uniform sampling, which seems more adequate than Euclidean distances of embedding spaces. The reconstruction of geodesics from point clouds has been explored by Crane et al. (2013). From an analytical point of view, the presented LJL for distribution normalization has a discrete, piecewise character as it takes only the closest neighbors into account which change from iteration to iteration. Most common approaches in theoretical mechanics aim for a continuous description, e.g., based on Hamiltonians and their analytical solution or numerical integration. It is theoretically not yet clear if it is possible to obtain a blue noise-type arrangement of particles utilizing pure Hamiltonian mechanics. For future work, we would like to further investigate this question. On a slightly different trajectory, we would like to explore the possibilities of relativistically moving particles. If such particles move within a medium with a suitable refraction index, then the power density of radiation grows linearly with the frequency. This effect is well studied in the literature and known as Vavilov-Cherenkov radiation (Cherenkov, 1934; Jackson, 1999).

Acknowledgements

This work has been supported by KAUST through the baseline funding of the Computational Sciences Group. The authors are grateful to Abdalla G. M. Ahmed for sharing Gaussian blue noise data and to Dmitry A. Lyakhov for helpful discussions about properties of the Lennard-Jones potential. The reviewers' valuable comments are gratefully acknowledged.

References

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J. Guibas. Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning*, 2017.

Abdalla G. M. Ahmed and Peter Wonka. Optimizing dyadic nets. *ACM Transactions on Graphics (TOG)*, 40(4):1–17, 2021.

Abdalla G. M. Ahmed, Till Niese, Hui Huang, and Oliver Deussen. An adaptive point sampler on a regular lattice. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Abdalla G. M. Ahmed, Jing Ren, and Peter Wonka. Gaussian blue noise. *ACM Trans. Graph.*, 41(6), nov 2022. ISSN 0730-0301. doi: 10.1145/3550454.3555519.

Marwan Al-Raeei and Moustafa Sayem El-Daher. Analytical formula of heat capacity in soft matter materials using lennard-jones potential. *Chemical Physics Letters*, 734:136729, 2019.

Naif Alharbi, Matthieu Chavent, and Robert S. Laramee. Real-Time Rendering of Molecular Dynamics Simulation Data: A Tutorial. In Tao Ruan Wan and Franck Vidal (eds.), *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association, 2017. ISBN 978-3-03868-050-5. doi: 10.2312/cgvc.20171277.

Hans C. Andersen. Rattle: A velocity version of the Shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52:24–34, 11 1983.

Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. $\ell 1$-sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph.*, 29(5), nov 2010. ISSN 0730-0301. doi: 10.1145/1857907.1857911.

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.

Nitesh Bhatia, Erich A. Müller, and Omar Matar. A gpu accelerated lennard-jones system for immersive molecular dynamics simulations in virtual reality. In *Virtual, Augmented and Mixed Reality. Industrial and Everyday Life Applications: 12th International Conference, VAMR 2020.*, pp. 19–34, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-49697-5. doi: 10.1007/978-3-030-49698-2_2.

Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches*, 10(1):1, 2007.

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Pavel Cherenkov. Visible emission of clean liquids by action of gamma radiation. *Proceedings of the USSR Academy of Sciences*, 2:451–454, 1934.

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

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):1–11, 2013.

Thomas E. Creighton. Protein folding. *Biochemical journal*, 270(1):1, 1990.

Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. Blue noise through optimal transport. *ACM Trans. Graph. (SIGGRAPH Asia)*, 31, 2012.

Julie Digne and Carlo de Franchis. The Bilateral Filter for Point Clouds. *Image Processing On Line*, 7: 278–287, 2017.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

Chaojing Duan, Siheng Chen, and Jelena Kovacevic. Weighted multi-projection: 3d point cloud denoising with tangent planes. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 725–729, 2018. doi: 10.1109/GlobalSIP.2018.8646331.

Daniel Dunbar and Greg Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Transactions on Graphics (TOG)*, 25(3):503–508, 2006.

Jacob D. Durrant and J. Andrew McCammon. Molecular dynamics simulations and drug discovery. *BMC biology*, 9(1):1–9, 2011.

Kristina Eriksen, Bjarne E Nielsen, and Michael Pittelkow. Visualizing 3d molecular structures using an augmented reality app, 2020.

Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2463–2471, 2016.

Raanan Fattal. Blue-noise point sampling using kernel density model. *ACM Trans. Graph.*, 30(4), jul 2011. ISSN 0730-0301. doi: 10.1145/2010324.1964943.

Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, 22 (3):950–953, jul 2003. ISSN 0730-0301. doi: 10.1145/882262.882368.

Jiayuan Gu, Wei-Chiu Ma, Sivabalan Manivasagam, Wenyuan Zeng, Zihao Wang, Yuwen Xiong, Hao Su, and Raquel Urtasun. Weakly-supervised 3d shape completion in the wild. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, pp. 283–299, Cham, 2020. Springer. ISBN 978-3-030-58558-7.

William E. Hart and Sorin Istrail. Robust proofs of np-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4(1):1–22, 1997.

Pedro Hermosilla, Tobias Ritschel, and Timo Ropinski. Total denoising: Unsupervised learning of 3d point cloud cleaning. *International Conference on Computer Vision 2019 (ICCV19)*, 2019.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

Marlis Hochbruck and Christian Lubich. A bunch of time integrators for quantum/classical molecular dynamics. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. Mark, S. Reich, and R. D. Skeel (eds.), *Algorithms for Macromolecular Modelling*, pp. 421–432. Springer, 1999.

Wei Hu, Xiang Gao, Gene Cheung, and Zongming Guo. Feature graph learning for 3d point cloud denoising. *IEEE Transactions on Signal Processing*, 68:2841–2856, 2020. doi: 10.1109/TSP.2020.2978617.

Zitian Huang, Yikuan Yu, Jiawen Xu, Feng Ni, and Xinyi Le. Pf-net: Point fractal network for 3d point cloud completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

John David Jackson. *Classical electrodynamics*. American Association of Physics Teachers, 1999.

Alexander D. Jamieson-Binnie, Michael B. O'Connor, Jonathan Barnoud, Mark D. Wonnacott, Simon J. Bennie, and David R. Glowacki. Narupa IMD: A VR-Enabled Multiplayer Framework for Streaming Interactive Molecular Simulations. In *ACM SIGGRAPH 2020 Immersive Pavilion*, SIGGRAPH '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379687. doi: 10.1145/3388536.3407891.

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. Blue noise sampling using an sph-based method. *ACM Trans. Graph.*, 34(6), nov 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818102.

John Edward Jones. On the determination of molecular fields. – i. from the variation of the viscosity of a gas with temperature. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):441–462, 1924a.

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

William L. Jorgensen, David S. Maxwell, and Julian Tirado-Rives. Development and testing of the opls all-atom force field on conformational energetics and properties of organic liquids. *Journal of the American Chemical Society*, 118(45):11225–11236, 1996. doi: 10.1021/ja9621760.

Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM Trans. Graph.*, 24:1442–1461, 10 2005. doi: 10.1145/1095878.1095888.

LAMMPS Molecular Dynamics Simulator, May 2014. URL `http://lammps.sandia.gov/`.

In-Kwon Lee. Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, 17(2): 161–177, 2000. ISSN 0167-8396. doi: https://doi.org/10.1016/S0167-8396(99)00044-8.

John E. Lennard-Jones. Cohesion. *Proceedings of the Physical Society*, 43(5):461, sep 1931. doi: 10.1088/0959-5309/43/5/301.

Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-gan: A point cloud upsampling adversarial network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. *Point-Voxel CNN for Efficient 3D Deep Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2836–2844, 2021a.

Shitong Luo and Wei Hu. Score-based point cloud denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4583–4592, 2021b.

Dominik L. Michels and Mathieu Desbrun. A semi-analytical approach to molecular dynamics. *Journal of Computational Physics*, 303, 10 2015. doi: 10.1016/j.jcp.2015.10.009.

Cengiz Öztireli, Marc Alexa, and Markus Gross. Spectral sampling of manifolds. *ACM Trans. Graph.*, 29 (6), dec 2010. ISSN 0730-0301. doi: 10.1145/1882261.1866190.

Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Learning graph-convolutional representationsfor point cloud denoising. In *The European Conference on Computer Vision (ECCV)*, 2020.

Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J. Mitra, and Maks Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 39(1):185–203, 2020.

Dennis C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2004. ISBN 9780521825689.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1530–1538. JMLR.org, 2015.

Jean-Paul Ryckaert, Giovanni Ciccotti, and Herman J.C Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977. ISSN 0021-9991.

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

Thomas Schlömer, Daniel Heck, and Oliver Deussen. Farthest-Point Optimized Point Sets with Maximized Minimum Distance. In Carsten Dachsbacher, William Mark, and Jacopo Pantaleoni (eds.), *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*. ACM, 2011. ISBN 978-1-4503-0896-0. doi: 10.1145/2018323.2018345.

Christian Schmaltz, Pascal Gwosdek, Andrés Bruhn, and Joachim Weickert. Electrostatic halftoning. *Computer Graphics Forum*, 29, 2010.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *CoRR*, abs/1907.05600, 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 *International Conference on Learning Representations*, 2021.

Erik Van Der Giessen, Peter A Schultz, Nicolas Bertin, Vasily V Bulatov, Wei Cai, Gábor Csányi, Stephen M Foiles, Marc GD Geers, Carlos González, Markus Hütter, et al. Roadmap on multiscale materials modeling. *Modelling and Simulation in Materials Science and Engineering*, 28(4):043001, 2020.

John Paul Walters, Vidyananth Balu, Vipin Chaudhary, David A Kofke, and Andrew Schultz. Accelerating molecular dynamics simulations with gpus. In *PDCCS*, pp. 44–49, 2008.

Xiaogang Wang, Marcelo H. Ang Jr. , and Gim Hee Lee. Cascaded refinement network for point cloud completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Kin-Ming Wong and Tien-Tsin Wong. Blue noise sampling using an n-body simulation-based method. *Vis. Comput.*, 33(6–8):823–832, jun 2017. ISSN 0178-2789. doi: 10.1007/s00371-017-1382-9.

Haozhe Xie, Hongxun Yao, Shangchen Zhou, Jiageng Mao, Shengping Zhang, and Wenxiu Sun. Grnet: Gridding residual network for dense point cloud completion. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX*, pp. 365–381, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58544-0. doi: 10.1007/978-3-030-58545-7_21.

Yin Xu, Ligang Liu, Craig Gotsman, and Steven J. Gortler. Capacity-constrained delaunay triangulation for point distributions. *Computers & Graphics*, 35(3):510–516, 2011. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2011.03.031. Shape Modeling International (SMI) Conference 2011.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Trans. Graph.*, 38(6), nov 2019. ISSN 0730-0301.

Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pp. 728–737, 2018. doi: 10.1109/3DV.2018.00088.

Cem Yuksel. Sample elimination for generating poisson disk sample sets. In *Computer Graphics Forum*, volume 34, pp. 25–32. Wiley Online Library, 2015.

Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion, 2021.

A Convergence Properties of the Lennard-Jones Layer

The LJL's output is computed by repetitively integrating forward in time while damping the motion of each particle over time. As a consequence of the damping effect, we observe that almost no changes to the nearest neighbors are present at the later stage of the process. Assuming that there are no changes to the nearest neighbors, we can theoretically prove the convergence of the LJL to the global potential minimum as outlined in the following theorem.

Theorem: The LJL's output is uniquely defined matching the single (global) minimum of the corresponding potential under the assumption that there are no changes of the nearest neighbors throughout the process.

Proof: Without loss of generality, let us consider a set which is composed of an even number of N particles. It is processed by the LJL, i.e., the dynamical process of each pair of particles is governed by the LJ potential defined in Eq. (1). The total potential $\mathcal{V}$ controlling the process carried out by the LJL is given as the sum of the LJ potentials of the $N/2$ pairs, i.e.,

$$\mathcal{V}(r_1, \ldots, r_{N/2}) = \sum_{i=1}^{N/2} V_i \quad \text{with} \quad V_i := 4\epsilon \left(\left(\frac{\sigma}{r_i} \right)^{12} - \left(\frac{\sigma}{r_i} \right)^{6} \right) , \tag{6}$$

in which r_i, $i = 1, \ldots, N/2$, denotes the Euclidean distance of the two particles composing the i-th pair.

The derivatives V_j' of Eq. (6) with respect to r_j, $j = 1, \ldots N/2$, are given by

$$V_j'(r_j) := \frac{\mathsf{d}}{\mathsf{d}r_j} \mathcal{V}(r_1, \ldots, r_{N/2}) = 24\epsilon \left(\frac{\sigma^6}{r_j^7} - 2\,\frac{\sigma^{12}}{r_j^{13}} \right) . \tag{7}$$

From $V_j'(r_j) \stackrel{!}{=} 0$, we obtain the candidate $r_j^* = \sqrt[6]{2}\,\sigma$ for a local extremum or saddle particle. The second partial derivative test results in $V_j''(r_j^*) > 0$ for which reason, the potential $\mathcal{V}$ has a local minimum at $(r_1^*, \ldots, r_{N/2}^*) = \sqrt[6]{2}\,(\sigma, \ldots, \sigma)$ with $\mathcal{V}(r_1^*, \ldots, r_{N/2}^*) = -(\epsilon, \ldots, \epsilon)$. This minimum is further the only local extremum and the global minimum of $\mathcal{V}$ which we can easily see by considering $\lim_{r_j \to 0^+} V_j(r_j) = \infty$ and $\lim_{r_j \to \infty} V_j(r_j) = 0$, $j = 1, \ldots N/2$.

In other words, the potential energy of each pair has only a single energy valley and as each pair is independent by construction, the LJL converges to the unique minimum of the potential $\mathcal{V}$.

□

B Parameter Configuration of the Lennard-Jones Layer

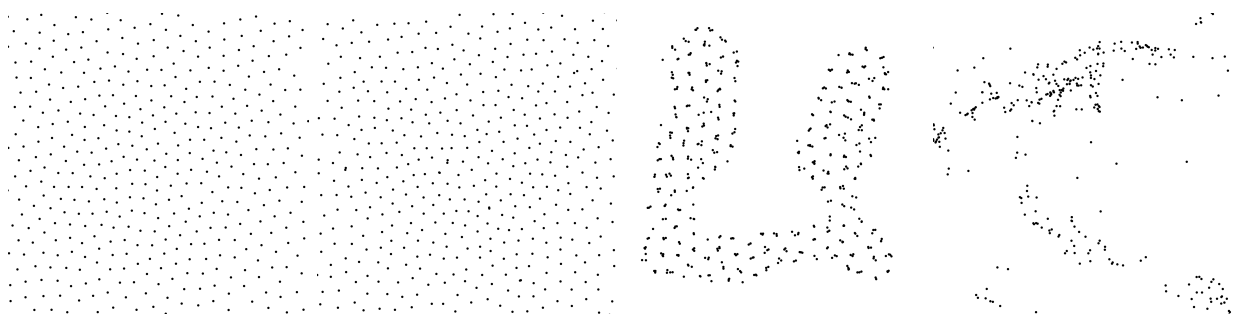
We investigate the behavior of LJL in detail by evaluating the point configuration of point sets on 2D Euclidean plane and 3D mesh surfaces and propose strategies for LJL parameter searching for practical applications.

B.1 Numerical Examples on 2D Euclidean Plane

Before applying LJLs to practical applications, we now analyze the behavior of LJLs in different situations (i.e., different numbers of neighbors k, with or without attraction term in LJ potential). We set LJL parameters $\alpha = 0.5$, $\beta = 0.01$, $\epsilon = 2$, and $\sigma = 5\sigma'$. All different cases are initialized with uncorrelated randomly distributed points inside a unit square region.

B.1.1 k-NN

The interaction acting on individual particles according to LJ potential can be summed with a varying number of nearest neighbors. Our theoretical finding guarantees convergence only for the consideration of a single nearest neighbor. The experiment shown in Fig. 12 indicates point configurations of LJLs considering a different number of nearest neighbors k. Cases with more than one neighbor increase the difficulty of reaching uniform point distribution.



(a) $k = 1$ (b) $k = 2$ (c) $k = 10$ (d) $k = 499$

Figure 12: LJL evaluated with a varying amount of nearest neighbors (500 points in total).

B.1.2 The Impact of the Attraction Term

We continue to discover the impact of attraction term in LJ potential. Two experiments, shown in Fig. 13 run in identical settings except for the appearance of the attraction term. Intuitively, the repulsive term alone should achieve an effect of redistribution. Without attraction term, however, particles are easily pulled out of the initial boundaries and spread out further. LJLs with the attraction term can enable the overall control of the point set and prevent particles from spreading out, which leads to a compact and equal distribution.

B.2 Redistribution of Point Cloud over Mesh Surfaces

In Fig. 14, two particles A and B form a cluster when they are directly projected on the surface (A' and B'). With the help of LJL rearrangement, particles are moved to new positions (A_{LJ} and B_{LJ}) and further projected back on the surface with proper distance (A'_{LJ} and B'_{LJ}). In this case, the redistributed points are not only located on the surface but also have blue noise properties. The computations involved in LJL-guided point cloud redistribution are summarized in Algorithm 3, where we start with a random 3D point cloud $\boldsymbol{X}_0$ initialized inside the cube $[-1, 1]^3$ and a normalized mesh surface $\boldsymbol{M}$. In order to select nearest neighbors on the same side of

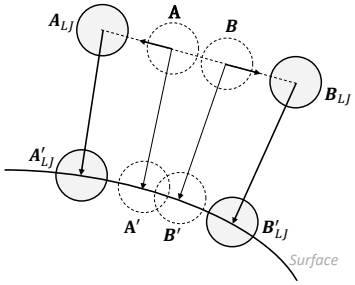


Figure 14: An illustration of how LJLs perform redistribution of clustered particles.

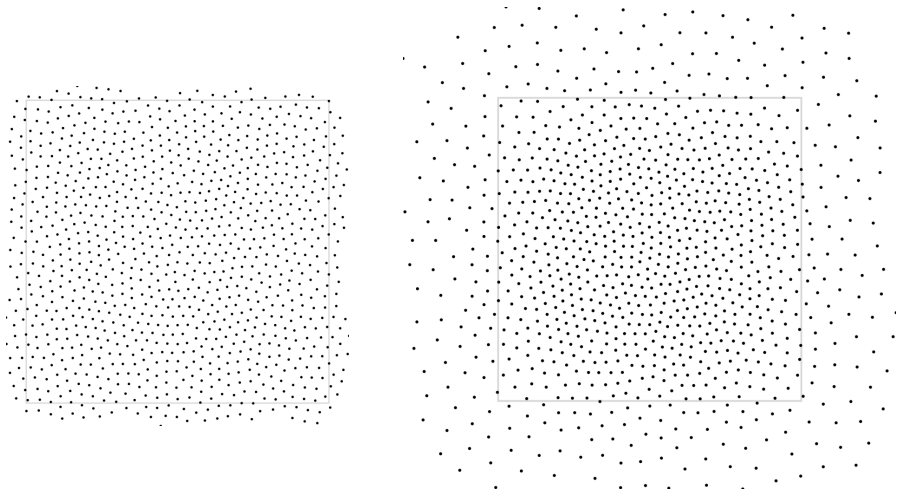


Figure 13: LJL with activated (left) and deactivated (right) attraction term.

the surface, LJLs only act on pairs that satisfy the condition that the intersection angle between their normals is less than $\frac{\pi}{4}$. In each LJL iteration, the redistributed points are projected back onto the surface. After a sufficient amount of LJL iterations, random point clouds can be rearranged to a normalized distribution and efficiently avoid the formation of clusters and holes.

ALGORITHM 3: *Distribution normalization of 3D point cloud over the mesh surface* $\boldsymbol{M}$.

Input: Point cloud $\boldsymbol{X}_0$ and tolerated threshold **tol**.
Output: Point cloud $\boldsymbol{X}_n$.

$n = 0$
repeat
  $\boldsymbol{N}_n \leftarrow \boldsymbol{Normal}(\boldsymbol{X}_n)$
  $\boldsymbol{N}'_n \leftarrow \boldsymbol{Normal}(\mathsf{NNS}(\boldsymbol{X}_n))$
  if $\boldsymbol{Angle}(\boldsymbol{N}_n, \boldsymbol{N}'_n) < \frac{\pi}{4}$:
    $\boldsymbol{X}_{n+1} \leftarrow \boldsymbol{LJL}(\boldsymbol{X}_n, \mathsf{NNS}(\boldsymbol{X}_n))$
  else:
    $\boldsymbol{X}_{n+1} \leftarrow \boldsymbol{X}_n$
  $\boldsymbol{X}_{n+1} \leftarrow \boldsymbol{Project\text{-}to\text{-}Mesh}(\boldsymbol{X}_{n+1})$
  $\Delta X \leftarrow \|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\|$
  $n \leftarrow n + 1$
until $\Delta X <$ **tol**

C LJL-embedded Deep Neural Networks

In this section, we show more details of LJL-embedded networks' parameter searching and results. We test all models on NVIDIA GeForce RTX 3090 GPU if not specified otherwise. Two well-trained point cloud generative models used in the papers are from ShapeGF (Cai et al., 2020) and DDPM (Luo & Hu, 2021a). The well-trained point cloud denoising model is from Luo & Hu (2021b).

C.1 LJL-embedded Denoising Model for 3D Point Cloud

C.1.1 Systematic Parameter Searching for α

We keep $\beta = 0.01$ inherited from the generation task and search for α by finding the minimum ratio of *noise and distance score increment rate*, shown in Fig. 15. We choose $\alpha = 0.3$ for the denoising task.

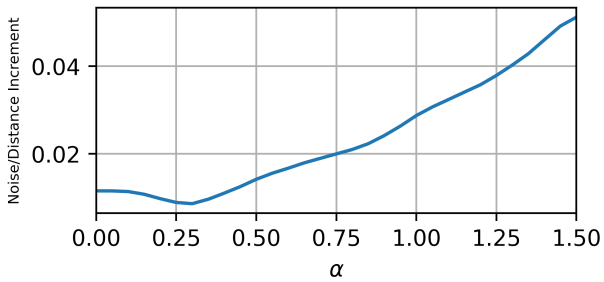


Figure 15: The ratio of *noise and distance score increment rate*. Parameter searching for optimal α by finding the global minima at $\alpha = 0.3$.

C.1.2 More LJL-embedded Denoising Results

LJL-embedded denoising results with different denoising iterations and noise scales are shown in Fig. 16.

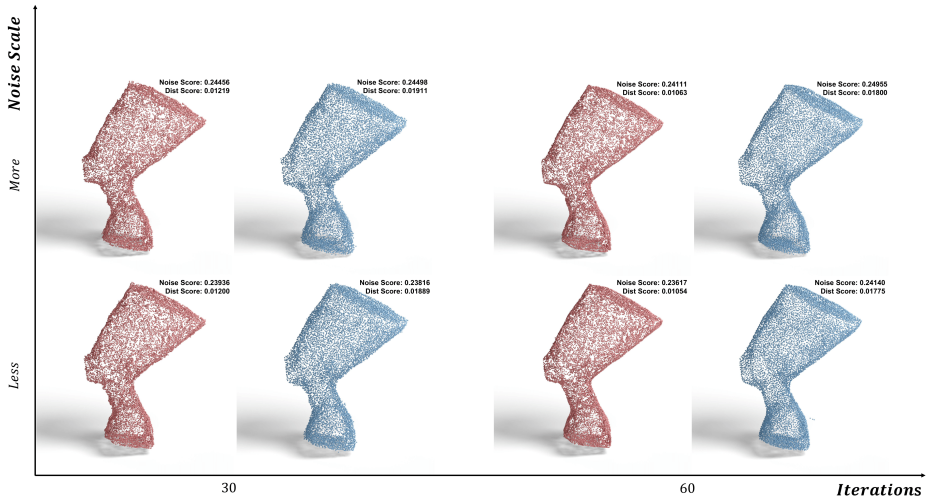


Figure 16: Comparison of denoise-only (red) and LJL-embedded denoising (blue) for 30/60 denoising iterations and different input noise scales.

C.2 LJL-embedded Generative Model for 3D Point Cloud

C.2.1 Systematic Parameter Searching for *Starting Steps*

Assuming that it takes $T = 100$ steps to generate point clouds, we continue to search for a good *starting steps* SS to insert LJLs shown in Fig. 17. $SS = 0$ means LJLs are applied from the very beginning, whereas $SS = 101$ means generation without LJLs. In order to balance the generation speed and LJL normalization effects, we choose $SS = 60$, meaning embedding should start in the second half of the generation steps.

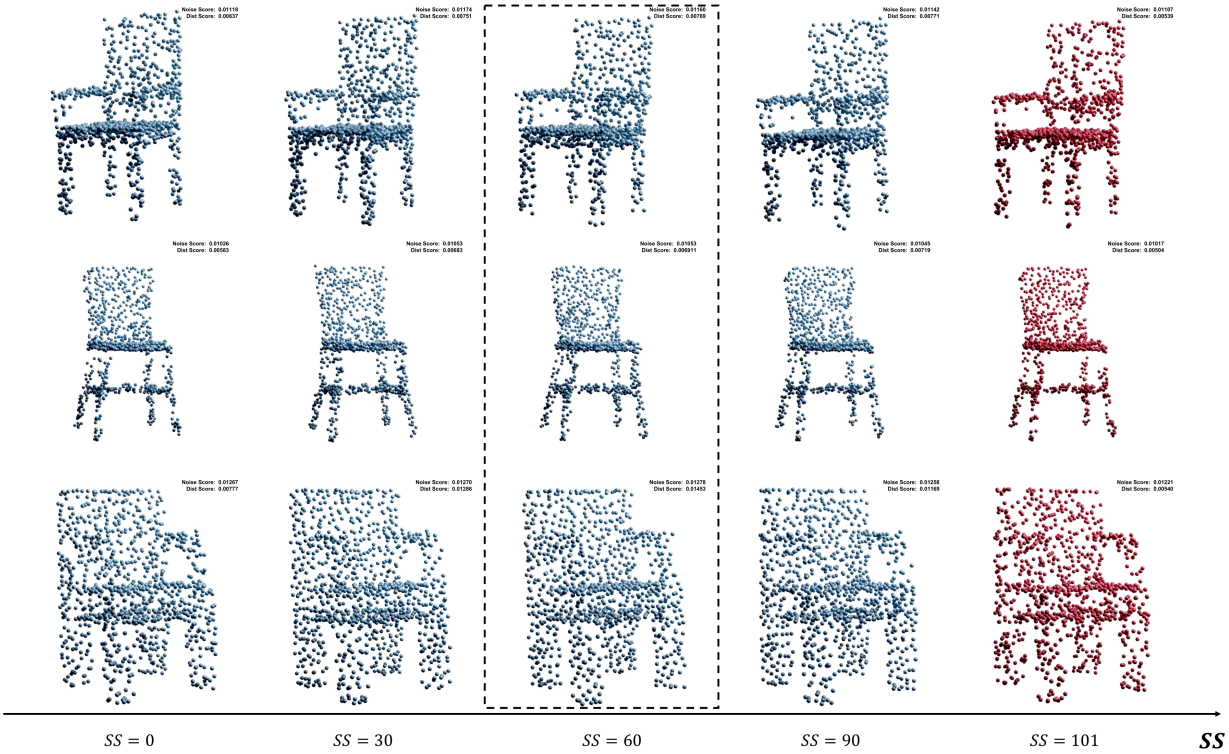


Figure 17: The optimal starting step (SS) to embed LJLs in the generation process (i.e. T=100). We select $SS = 60$ in the actual generation task.

C.2.2 Systematic Parameter Searching for α and β

In order to determine α and β, we perform a systematic parameter searching based on *distance* and *noise* scores, shown in Fig. 18. We found that $\alpha = 2.5$ and $\beta = 0.01$ meet the requirements of less noise and better distribution.

C.2.3 Point Cloud Generation Process with Different Numbers of Points

We especially focus on the task of using as few points as possible in generation tasks. For use cases with a large number of points, inefficient point distributions are less of a concern since even for low-density regions, there are sufficient points to describe the underlying surface. Fig. 19 shows how LJLs behave in the generation process with different numbers of points N. Note that our choice of $\sigma = 5\sigma'$ implicitly accounts for the change in N. In the case of generation tasks with fewer points, the LJL-embedded generator can redistribute the limited number of points as even as possible while maintaining the global structure. Combining LJLs with generative models enables points to have the ability to converge to the different parts of the shape.

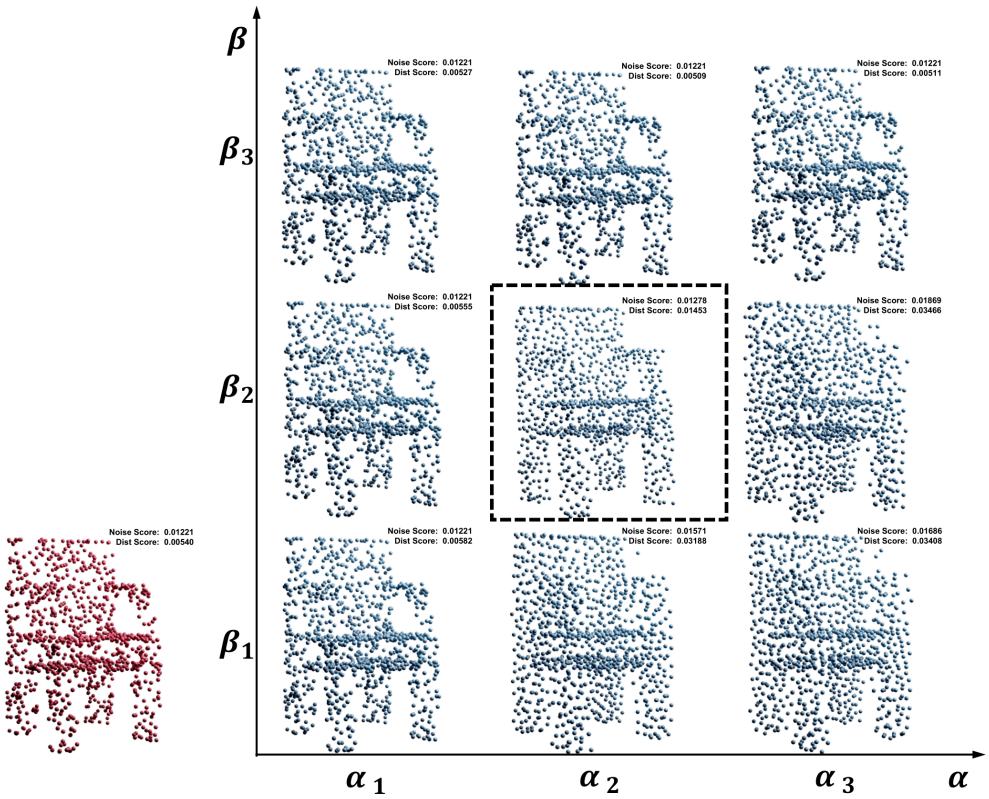


Figure 18: Parameter searching for decaying factors α and β, where $\alpha_1 = 0.1$, $\alpha_2 = 2.5$, $\alpha_3 = 5.0$ and $\beta_1 = 0.001$, $\beta_2 = 0.01$, $\beta_3 = 0.1$. The optimal parameters are $\alpha = 2.5$ and $\beta = 0.01$.

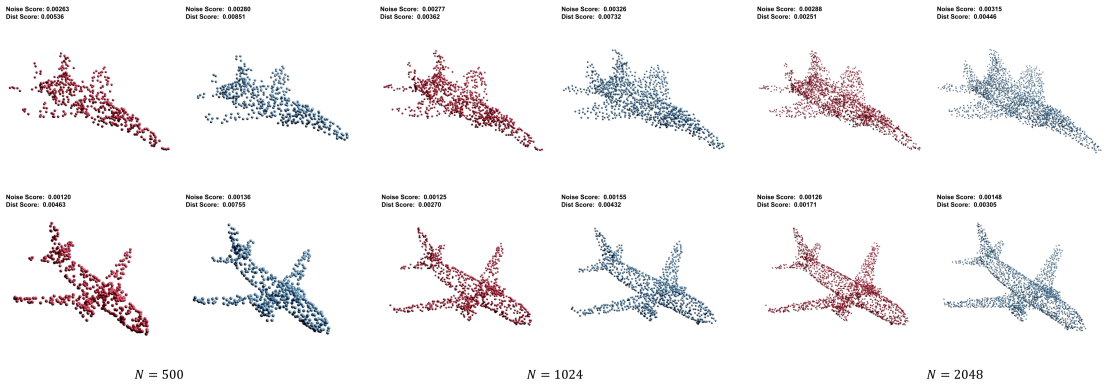


Figure 19: Examples of generation results with regard to the increasing number of sample points N. Red: Raw generation results. Blue: LJL-embedded generation results.

C.2.4 More LJL-embedded Generation Results

More LJL-embedded generation results are shown in Fig. 20, Fig. 21, and Fig. 22.

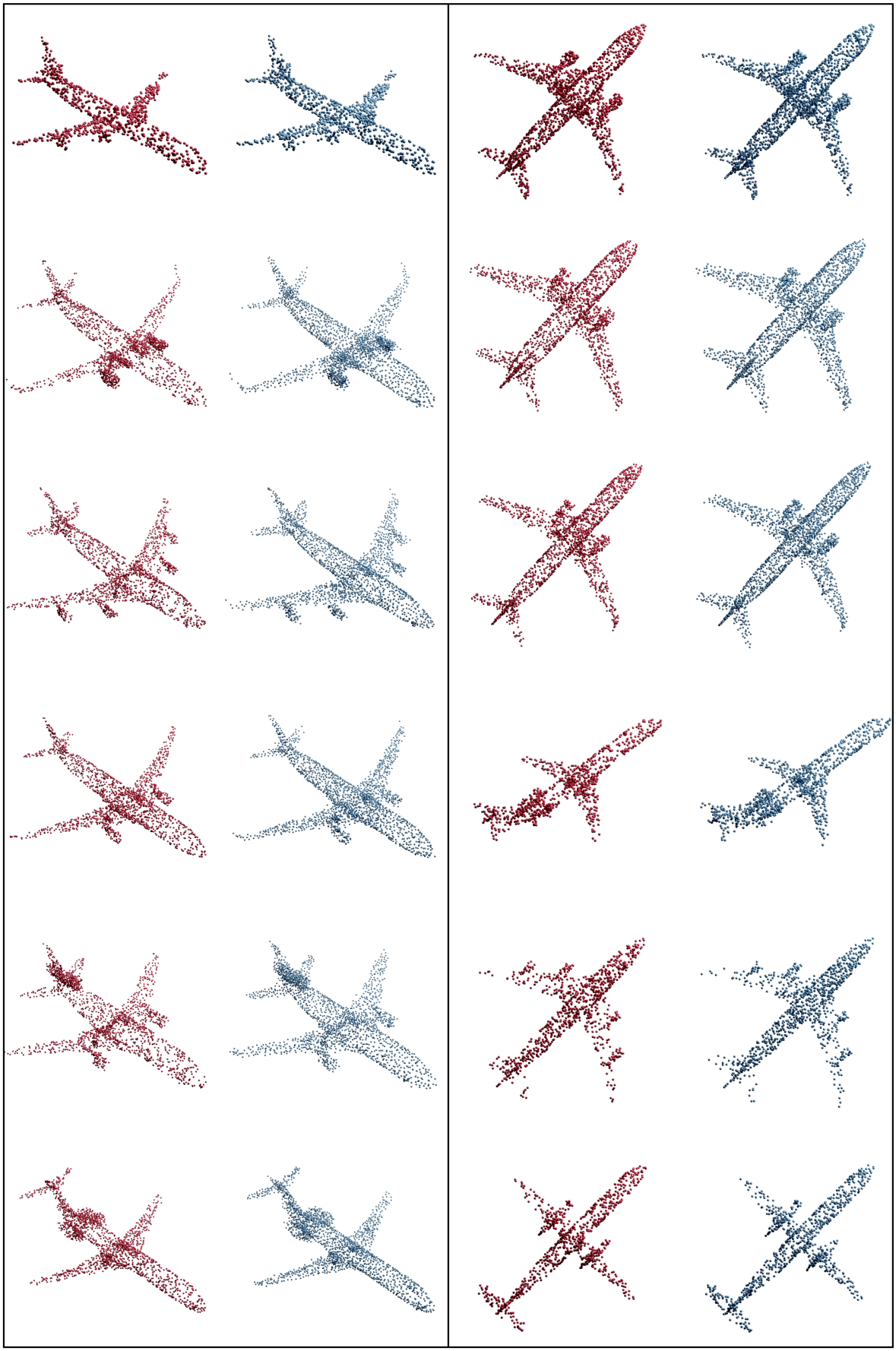


Figure 20: Comparison of raw generation (red) and LJL-embedded generation (blue).

Figure 21: Comparison of raw generation (red) and LJL-embedded generation (blue).

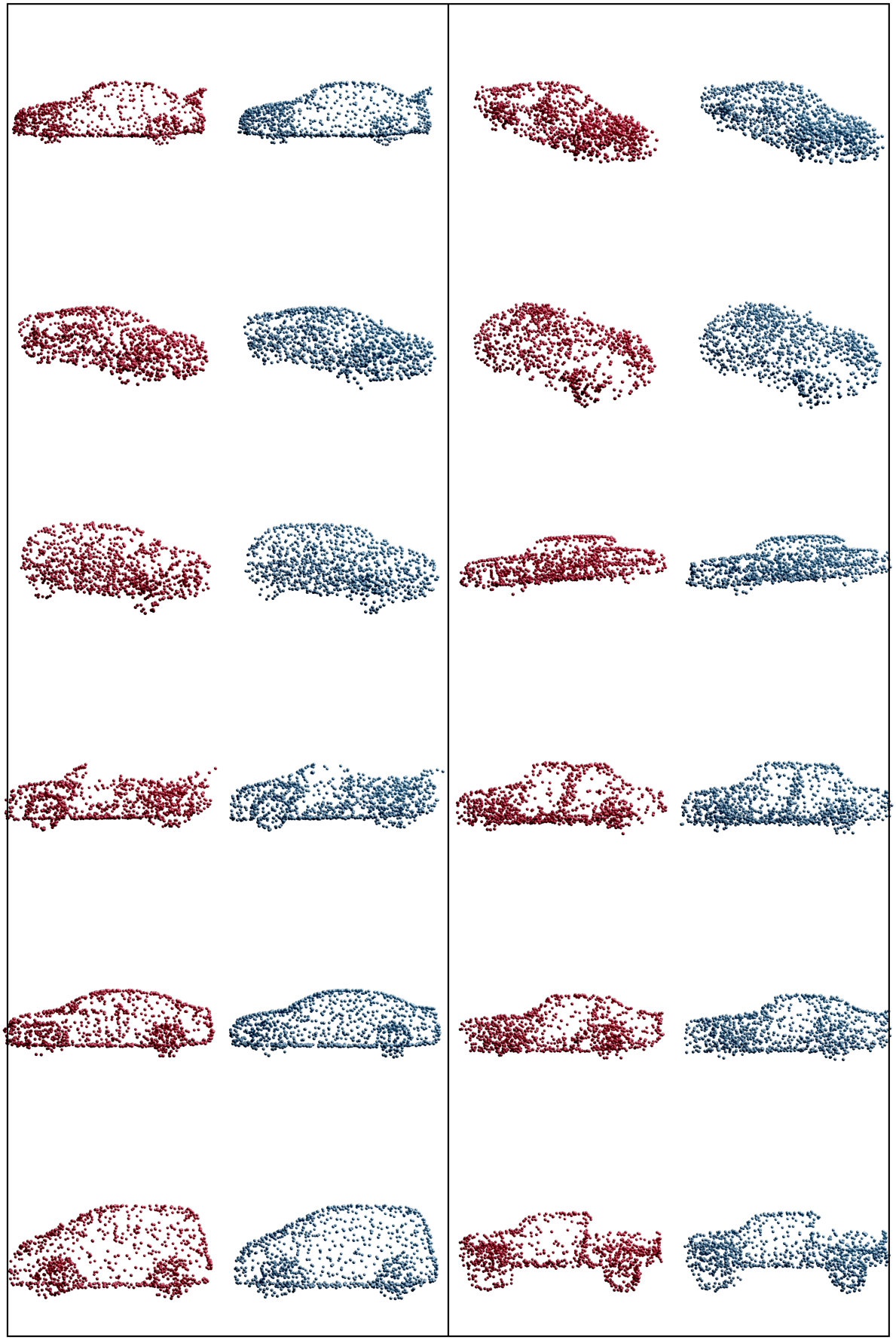


Figure 22: Comparison of raw generation (red) and LJL-embedded generation (blue).