Abstract—We propose a data reduction technique for scattered data based on statistical sampling. Our void-and-cluster sampling technique finds a representative subset that is optimally distributed in the spatial domain with respect to the blue noise property. In addition, it can adapt to a given density function, which we use to sample regions of high complexity in the multivariate value domain more densely. Moreover, our sampling technique implicitly defines an ordering on the samples that enables progressive data loading and a continuous level-of-detail representation. We extend our technique to sample time-dependent trajectories, for example pathlines in a time interval, using an efficient and iterative approach. Furthermore, we introduce a local and continuous error measure to quantify how well a set of samples represents the original dataset. We apply this error measure during sampling to guide the number of samples that are taken. Finally, we use this error measure and other quantities to evaluate the quality, performance, and scalability of our algorithm.

Index Terms—Data reduction, sampling, blue noise, entropy-based sampling, scattered data, pathlines

1 INTRODUCTION

In the field of scientific visualization, interactive exploration and analysis are considered essential to gain insight into large and complex datasets. Although data sizes are growing rapidly, for example due to advancements in high-performance computing or increasingly accurate measurement devices, storage bandwidth does not increase accordingly.

Data reduction is thus a necessary means to reduce storage requirements for both simulation, measurement devices, and for subsequent data analysis.

We specifically consider the reduction of large, spatio-temporal scattered data, i.e. unstructured points in space-time with an associated value domain. In particular, we investigate the use of statistical sampling to reduce large data sets to a representative subset. Sampling scales well to higher dimensional data and is well-suited for scattered data. Although simple random sampling gives decent results, recent work improves upon this using stratified [21,28] and information-guided sampling [5,27]. These results emphasize the significance of...
stratification in the spatial domain and adaptive sampling guided by the value domain.

We propose a sampling strategy for scattered data generalizing the void-and-cluster technique from Ulichney [25] that stratifies optimally in the spatial domain. Specifically, we find samples that are well distributed with respect to the blue noise property, which implies large mutual distances between samples without causing regularity artifacts. Additionally, we discuss how to adapt the sampling strategy to the value dimensions by better sampling regions of value distributions with high entropy. Moreover, the sampling technique implicitly defines an ordering on the samples that enables progressive data loading and continuous level-of-detail during visualization and analysis. Our proposed algorithm is fast, scalable, and well-suited for GPU acceleration. Therefore, it is applicable in-situ, i.e. while a simulation is running, but also as a traditional post-processing step.

Furthermore, we extend our sampling technique to time-dependent scattered data. Instead of considering each time step independently, we sample trajectories, i.e. sequences of scattered points defined over time. We find representative trajectories that evenly cover the spatio-temporal domain based on an efficient iterative extension of the void-and-cluster technique. An example for such trajectory datasets are particle-based simulations that trace particles over time. Additionally, representing fluid flows using Lagrangian trajectories, i.e. pathlines, instead of velocity fields has recently gained popularity [1]. Pathlines are thereby advected during simulation time using the high-resolution vector field data, which could not be stored otherwise. In both of these examples, the data consists of a large amount of trajectories that we reduce using our sampling technique.

Lastly, we introduce an error measure to quantify how well a set of samples represents the data with respect to both the spatial and the value domain. In particular, we derive a continuous error measure that quantifies the difference in the value distributions for every point in the dataset. This error measure integrates well into our sampling technique, where we use it to determine when a sufficient number of samples has been taken. We evaluate the quality of our proposed sampling technique on different synthetic and real-world datasets using this error measure and other derived quantities, such as the quality of scattered data interpolation. Finally, we investigate the performance and scalability of our proposed sampling technique and compare it to related approaches.

To summarize, our contribution is a sampling technique that:

- Takes optimally distributed samples in the spatial domain with respect to the blue noise property,
- Adapts to an arbitrary density, for example derived from a multivariate value domain,
- Implicitly defines an ordering of the samples that we use for continuous level-of-detail and progressive data loading,
- We extend to sample time-dependent data, for example pathlines in a fluid flow.

2 RELATED WORK

We first discuss the visualization of large data with a focus on data reduction and scattered data, before we introduce the concept of blue noise and discuss corresponding sampling strategies.

2.1 Visualizing Large Datasets

To visualize large datasets, we focus on approaches that create a compact derived representation, instead of orthogonal approaches such as data compression. Li et al. [13] survey data reduction techniques for simulation, visualization, and data analysis.

Several distribution-based data representation approaches have been proposed, which represent large datasets using distributions that are sampled during subsequent visualization and analysis. In particular, Thompson et al. [23] represent value distributions by storing a histogram per block of voxels. Sicat et al. [19] construct a multi-resolution volume from sparse probability density functions defined in the 4D domain comprised of the spatial and data range. Several promising approaches rely on Gaussian mixture models (GMMs), which represent arbitrary distributions as a weighted combination of Gaussians. Wang et al. [20] employ a spatial GMM in addition to a value distribution in each data block, whilst Dutta et al. [7] partition the data into local, homogeneous regions and fit a GMM in each partition. For in-situ processing, Dutta et al. [6] perform incremental GMM estimation instead of expectation maximization to compute the mixture models. Hazarika et al. [11] model distribution-based multivariate data using copula functions, which allow modeling the marginal distributions separately from the dependencies between dimensions.

Although distribution-based approaches achieve a significant reduction in data size, they are difficult to extend to higher-dimensional data due to the curse of dimensionality. Moreover, these approaches have been developed for uniformly structured data and the extension to scattered data is non-trivial. While scattered data can be visualized by first reconstructing a structured representation [8,10], this approach has its own drawbacks and is not an option for all analysis techniques and needs. For example, particle-based visualizations benefit from specific visualization and analysis techniques [10].

2.2 Sampling

Statistical sampling of data [5] is gaining popularity in the field of scientific visualization. Reinhardt et al. [17] use stochastic sampling to improve performance and reduce visual clutter for the visual debugging of smoothed particle hydrodynamics (SPH) simulations. Sauer et al. [18] propose a data representation that combines particle and volume data and supports sampling of particles by using the corresponding volume to find evenly distributed samples. Woodring et al. [28] describe a simulation-time stratified sampling strategy for a large-scale particle simulation. For stratification, the authors construct a kd-tree from the data that is also used as a level-of-detail representation. Su et al. [21] discuss server-side sampling using bitmap indices and stratify both in the spatial and value domain. Wei et al. [27] extend their approach with an information-guided sampling strategy and recovery technique. During sampling, they measure the information per stratum by computing the entropy of the value distribution and draw samples accordingly. Biswas et al. [3] similarly employ an information-guided strategy to sample adaptively during in-situ simulation, but use a global histogram for the entropy computation.

A desirable property of point distributions is the blue noise characteristic [24], which leads to large mutual distances between points without noticeable regularity artifacts. Balzer et al. [2] compute Capacity-constrained Voronoi diagrams (CCVD) to optimize the blue noise property of point distributions, which allows adapting the point distributions to a given density function. To find representative particles, Frey et al. [9] propose loose capacity-constrained Voronoi diagrams (LCCVD) that relax the capacity-constraints of the CCVD method and are computed on the GPU. All methods based on capacity-constrained Voronoi diagrams can be used to sample scattered data, but are computationally demanding. Bridson [4] presents a Poisson disk sampling technique to generate blue noise samples in arbitrary dimensions by enforcing a minimal and maximal distance between nearest neighbors. The technique is designed to produce entirely new sample sets, not to reduce an existing sample set.

Ulichney’s algorithm [25] is restricted to regular grids and produces samples with a uniform density. In this section, we generalize the
approach to scattered data with a non-uniform distribution. To preserve the spatial density, we compute a density estimate on the whole dataset. Our generalized void-and-cluster sampling works on scattered data and enforces the given density (see Sect. 3.1). Like the original algorithm, it orders all sample points to enable level of detail and progressive data loading (Sect. 3.2). The algorithm is efficient because each iteration only requires local updates with compact kernels (Sect. 3.3). Our parallel implementation in Sect. 3.4 further exploits this locality. Supporting arbitrary sampling densities lets us emphasize regions of high entropy in the value domain (Sect. 3.4). Finally, we extend our technique to the sampling of time-dependent trajectories (Sect. 3.5).

3.1 Void-and-Cluster Sampling

Assume we have a dataset with points $P \subset \mathbb{R}^d$ in a $d$-dimensional spatial domain. Each sample is mapped to a value in the possibly multivariate value domain $V$ through $v: P \rightarrow V$. Among these points, we want to pick a representative subset $S \subset P$. Therefore, we optimize the placement of samples in the spatial domain by estimating the density of selected samples $\lambda_S : P \rightarrow \mathbb{R}^+$ for each point $p \in P$. A high sample density indicates a large number of nearby samples, whilst a low density indicates few. We want to place samples such that dense regions (clusters) and empty regions (voids) are avoided. Or, in other words, reduce the maximum of the sample density $\lambda_S$ and increase its minimum.

This does not work for spatially non-uniformly distributed data points since we have to account for the original distribution in the spatial domain. Even for uniformly distributed points the border region of the spatial domain is less densely populated. We account for the spatial distribution of the points by first computing a point density $p_P$ for each $p \in P$:

$$p_P(p) := \sum_{p \in S} k(||p - p||),$$

(1)

using a kernel function $k$. Given a subset of samples $S \subset P$, the sample density at $p \in P$ is then defined as:

$$\lambda_S(p) := \frac{\sum_{s \in S} k(||p - s||)}{p_S(p)}.$$

(2)

We will now describe a strategy to find the optimal set of samples in the spatial domain with respect to the sample density $\lambda_S$ by extending the void-and-cluster algorithm (25). This is an iterative and greedy algorithm that at each step finds a locally optimal distribution of samples. An overview of our modified void-and-cluster sampling technique is depicted in Fig. 2.

Initially, we take a fixed number of random samples. Although the point density $p_P$ stays constant, we have to update $\lambda_S$ when we change the set of samples. The sample density is computed incrementally when a sample $s$ is added (or removed), by adding to (or subtracting from) the density $\lambda_S(p)$ for all points.

We then optimize these initial samples by removing the tightest cluster

$$s_{\max} = \arg \max_{s \in S} \lambda_S(s) \in S,$$

(3)

i.e. the sample with largest $\lambda_S$. Then, we add the largest void $p_{\min}$ of $\lambda_S$ that is not a sample yet. Since we add and remove a sample, we have to update the sample densities accordingly. The optimization stops once the tightest cluster that we remove then becomes the largest void, that is $s_{\max} = p_{\min}$.

After construction of the optimal initial sampling, we iteratively find and add the largest void to the set of samples until we have reached the desired amount of samples. We provide detailed pseudocode of the entire algorithm in the supplementary document.

3.2 Ordering of Samples

A positive side-effect of the greedy approach is that the void-and-cluster strategy implicitly defines an ordering of the samples. With respect to this ordering, any prefix of the sample set $S$ still has good blue noise characteristics. Ulichney (25) denotes it as the rank $r: P \rightarrow N$, where $r(p) = \infty$ for all $p \in P \setminus S$. To compute this ordering, we assign and increment the rank when adding a sample during the initial random sampling or the void filling steps. For a sample $s_i$ that is added as the $i$-th sample, we set $r(s_i) = i$. During the void-and-cluster optimization, when we exchange the tightest cluster $s_{\max}$ with the largest void $p_{\min}$, we have to swap the rank accordingly, i.e. we set $r(p_{\min}) = r(s_{\max})\) and $r(s_{\max}) = \infty$.

We re-order (or index) the samples according to this mapping. We can use this ordering for continuous level-of-detail and for progressive data loading during the subsequent visualization and analysis.

3.3 Compact Kernels

If the kernel $k$ is compact, i.e. has a finite extent, only a local neighborhood has to be considered when updating the densities of samples and points. For compact kernels, the optimization is thus defined locally. In our experiments, we found that the choice of kernel does influence the distribution of samples and the quality of the blue noise. Nonetheless, we always achieved good results as long as the kernel size $h_P$ was in a reasonable order of magnitude with respect to the spatial domain. If we take a fraction of all samples $|S| < |P|$, we have to increase the kernel size $h$ used for sampling accordingly:

$$h := h_P \sqrt{\frac{|P|}{|S|}},$$

(5)

using the spatial dimension $d$. Although we did experiment with a Gaussian kernel, we use a cubic spline (15) in the remainder of this work since it yields similar results, but is compact. Lastly, we denote points in the support of kernel $k$ at point $p \in P$ as its neighborhood $N_P(p) \subset P$.

3.4 Adaptive Sampling

So far we have taken all samples with equal probability and proportional to the spatial density. Now, we discuss the use of non-uniform
Thus, any normalized importance measure, for example a computed sample subset trajectory can have different starting and ending points in time, i.e. it of scattered data points in discrete time steps. In addition to sampling a single time step, we extend our sampling technique to time-dependent data. Specifically, we consider trajectories multiple value dimensions. Hence, we compute the entropy individ-
ualy in each dimension and use the maximal entropy at each point. For each point \( p \in P \) we exchange the tightest cluster with the largest void up to \( \epsilon \). To stop a trajectory \( T \) in time step \( t \), we take the prefix \( T':=(p_1,\ldots,p_j) \) instead of \( T \). The parameter \( \epsilon \) depends on the dataset and the specific application. In general, it should be inversely proportional to the amount of trajectories ending. In datasets where all trajectories exist in all time steps, \( \epsilon \) should be high. To select which trajectories to stop and which to start in time step \( t \), we perform the void-and-cluster optimization. That is, we exchange the tightest cluster with the largest void up to \( \epsilon \) times or until the sample distribution is optimal, i.e. the tightest cluster is equal to the largest void, and start or stop the corresponding trajectories. Note that longer trajectories may again be obtained by interpolation from shorter ones.

4 Parallel Implementation

In this section, we discuss the parallel implementation of the sampling technique, specifically the computation of the point and sample densities, and the parallelization of the void filling step.

4.1 Computing the Densities

One of the most computationally demanding parts of the algorithm is creating the density \( p_{\theta} \) and updating \( \lambda_{S} \). Each point \( p \) has to scatter its density, weighted by the distance and kernel function, to all neighboring points \( N_{p} \). This is an embarrassingly parallel task and is especially well-suited for GPU acceleration. If the kernel function is compact, data structures, such as a kd-tree or a regular grid, should be employed to efficiently retrieve the neighborhood of a point or sample.

In our implementation, we use a uniform grid of cell size \( h \). To find the neighborhood \( N_{p} \) of \( p \), we thus have to query \( 3d \) cells around \( p \). To speed-up the neighborhood search, we layout all cells in memory using a space-filling Z-curve to optimize memory access to neighboring cells.

4.2 Parallel Void Filling

The void filling step seems to enforce a sequential bottleneck: In each step, we find the sample \( p \in P \setminus S \) with the smallest sample density \( \lambda_{S}(p) \). Then we add \( p \) to the sample set \( S \) and increase sample densities in the neighborhood before we search the smallest sample density again. To overcome this sequential dependency, we store each added sample \( p \) alongside the sample density \( \lambda_{S}(p) \) that it has when it is added. Since we only ever add to the densities and pick the minimum in each step, these densities grow monotonically. If we can guarantee that they are
computed correctly for each added sample, sorting by the densities guarantees that we rank all selected samples correctly.

To provide this guarantee, we must never add a sample too early. All samples in a neighborhood \( p_n \in N_p \) with a rank \( r(p_n) < r(p) \) must have contributed to the sample density \( \lambda_S(p) \) before we add \( p \in P \setminus S \) and store \( \lambda_S(p) \). We can be certain that this is the case if \( p \) has the smallest sample density in its neighborhood, i.e.

\[
\lambda_S(p) \leq \min_{p_n \in N_p} \lambda_S(p_n).
\]

By selecting the sample \( p \in P \setminus S \) that minimizes \( \lambda_S(p) \) globally, we will never select another sample in \( N_p \) before \( p \). Hence, we know that the rank \( r(p) \) is also minimal within the neighborhood \( N_p \).

This principle enables our parallel implementation. We first sort all \( p \in P \setminus S \) in ascending order by their density \( \lambda_S(p) \) and take the first \( n \) points \( p_0, p_1, \ldots, p_{n-1} \) in each iteration. Then we compute an adjacency matrix in parallel using the kernel size \( h \):

\[
A := \begin{pmatrix}
0 & 0 & \ldots & 0 \\
\|p_1 - p_0\| - h & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\|p_{n-1} - p_0\| - h & \ldots & \|p_{n-1} - p_{n-2}\| - h & 0
\end{pmatrix}.
\]

A negative entry in the \( i \)-th row and \( j \)-th column with \( i > j \) indicates that \( p_i \) is in the neighborhood of \( p_j \). Since \( \lambda_S(p_i) \geq \lambda_S(p_j) \) due to the sorting, this means that \( p_i \) might not satisfy Equation (10) and it is flagged accordingly. Once the process is complete, all points that have not been flagged satisfy Equation (10). They are added to the sample set and their densities are stored. Note that the matrix \( A \) is never stored. We only need the flags.

Although we can parallelize this computation, the workload is unevenly distributed. The \( i \)-th row of \( A \) has \( i \) non-zero entries. Therefore, we index the non-zero entries of \( A \) with a single linear index \( k \in \{0, \ldots, \frac{2n(n-1)}{2}\} \). In the supplementary material we show that row and column indices can be computed from this flat index through

\[
i \equiv \left[ \frac{1}{2} + \sqrt{\frac{1}{4} - 2k} \right], \quad j \equiv k - \left( \frac{i-1}{2} \right).
\]

For large \( k \), the square root has to be evaluated in double-precision to avoid rounding errors.

Thanks to the sorting by density, the procedure described above guarantees a correct relative rank of selected samples. However, samples may be missing if we just terminate after a particular iteration. For a complete result, we perform additional iterations. If the largest density of a sample added in the last proper iteration was \( \lambda_{\text{max}} \), we continue iterating until the smallest sample density in \( P \setminus S \) is greater than \( \lambda_{\text{max}} \).

At this point, we can be certain that we have not missed a sample that should have been added up until the last proper iteration. This way, we guarantee that we take the same samples as the sequential algorithm.

5 Local Error Measure

In this section, we discuss an error measure to quantify how well a set of samples represents a dataset. We propose a measure that takes not only the spatial domain into account, but also how well the value domain is represented in each region of the dataset. To this end, we first discuss how such a local error can be defined, before we discuss how to compare value distributions. Lastly, we discuss an error guided sampling strategy that relies on an efficient iterative error estimation to sample just below a given error threshold, instead of drawing a fixed amount of samples.

5.1 Locality and Continuity

We derive a local error measure that compares the value distribution \( V_S \subset V \) of the sampled dataset with the value distribution \( V \) of the original dataset. Specifically, we propose to compare value distributions in the local neighborhood \( N_p \) for each corresponding \( p \in P \). This method implicitly accounts for non-uniformly distributed data points. Additionally, we weight the contribution of each \( p_i \in N_p \) to the value distribution by its distance \( k(||p - p_i||) \) so that the error varies smoothly over the spatial domain.

5.2 Wasserstein Distance

To measure the difference between the original value distribution given by values \( V = \{X_0, \ldots, X_n\} \) and a sampled subset \( V_S \subset V \), we use the corresponding cumulative distribution functions (CDFs) \( F_V \) and \( F_S \). The CDF at a point \( p \in P \) is estimated as

\[
F_V(p, t) = \frac{1}{n-1} \sum_{i=0}^{n-1} k(||p - p_i||) \quad \text{if } X_i \leq t,
\]

otherwise.

In practice, this implies that we need to sort the \( X_i \) before accumulating them. Since the samples are a subset \( V_S \subset V \), it is sufficient to sort the values \( V \) to estimate both CDFs. Alternatively, we estimate the CDFs based on a histogram of \( V \) and \( V_S \), which introduces a discretization, but is more efficient to evaluate.

To measure the distance, we found the Wasserstein distance, or earth movers distance, to be a good choice. In the one-dimensional case, it is defined as the L1-norm between the two CDFs:

\[
W(p, F_V, F_S) := \int_{-\infty}^{\infty} |F_V(p, x) - F_S(p, x)| \, dx.
\]

In contrast, we found the Kolmogorov-Smirnov distance, defined as the infinity norm between the CDFs, to be unsuited since it is not robust to small shifts in the value dimension. Note that this definition of the Wasserstein distance is only valid for one-dimensional value distributions. Thus, we compute a separate error for each value dimension. We can further deduce the error across dimensions, e.g. by taking the mean or maximum. In the following we will use the maximum; however, this is an application and data specific decision.

5.3 Error Guided Sampling

During void-and-cluster sampling, we efficiently keep track of the error distribution, for example to stop sampling if the average error falls below a given threshold. In detail, we compute the error for all samples after the initial void-and-cluster optimization. When adding a sample \( p_{\text{min}} \), we compute the error of \( p_{\text{min}} \) and additionally update the error for all neighbors \( N_{p_{\text{min}}} \) since these have changed as well.

To describe the distribution of errors during sampling, we found the average error to be a robust statistic that is efficient to compute. In contrast, the maximal error does not decrease smoothly with respect to the number of samples and is not robust against outliers, e.g. stemming from small, but complex value regions.

6 Results and Discussion

In this section, we evaluate our sampling technique using four real-world datasets and the synthetic sinc signal.

6.1 Synthetic Data: Sinc

We have created the sinc dataset by randomly placing 500,000 points in the domain \([-5, 5]^2\) and by evaluating for each point \( p \in [-5, 5]^2 \) the function

\[
sinc(||p||) = \frac{\sin(\pi ||p||)}{\pi ||p||}.
\]

![Image](Fig. 6) (left) compares different sampling strategies and shows the mean Wasserstein distance. We employ simple random sampling and random sampling with non-uniform probabilities based on the entropy. Moreover, we compare to stratified sampling utilizing a kd-tree based on a median split, similar to Woodring et al. [28]. Lastly, we employ Poisson disk sampling [4] and loose capacity constrained Voronoi diagrams (LCCVD, [9]). For an increasing number of samples, the error from most strategies converges to zero, which implies that these strategies sample a representative subset. However, Bridson’s Poisson disk sampling [4] lacks explicit control over the sample count, which
is instead steered by the enforced minimal and maximal distance. The technique is unable to surpass a certain sample count for this dataset. Our proposed void-and-cluster sampling strategies perform best for all sample counts. The entropy-based strategies perform similar to their uniform counter parts.

In Fig. 6 (right), the mean error has been computed iteratively during sampling until the error was less than \( \varepsilon = 0.0065 \), which led to a sampling percentage of 34.1%. The error first falls rapidly then converges asymptotically to zero. Initially, we sample 5,000 and iteratively add the remaining samples. In each void filling step, we take only 32 samples in parallel to still keep the error up to date. In comparison, we take up to 12,288 voids in parallel if we do not perform error guided sampling.

We use scattered data interpolation to interpolate the sampled data values to a grid of size \( 1024^2 \), see Fig. 5. Our void-and-cluster strategies show a major improvement compared to the other sampling strategies. The signal-to-noise (SNR) ratios shown in the logarithmic decibel scale support this assessment. Note that the LCCVD and Poisson disk sampling strategies also achieve good results, but are still worse than our proposed methods. For reconstruction, the entropy-based sampling strategies perform slightly better compared to the uniform approaches. For all sampling strategies, the quality of the reconstruction agrees with the error measure.

Lastly, Fig. 7 shows the spectrum of the sinc dataset reduced to a subset of 1% with our uniform void-and-cluster strategy, LCCVD, and Poisson disk sampling. The Fourier transform shows the blue noise property for these methods. Low frequencies are substantially weaker and the spectrum is isotropic. However, LCCVD has more energy in low frequencies than our void-and-cluster strategy. Poisson disk sampling has less energy in low frequencies, but contains a noticeable spike near zero. Note that the random and stratified sampling strategies do not have this property, which suggests that the blue noise property is desirable for scattered data interpolation. Indeed, the error of kernel estimation has been shown to depend on the disorder of particles [14].

6.2 Von Kármán Vortex Street

The von Kármán vortex street is a time-dependent SPH dataset that contains about 5 million particles in each time step. Since the particles enter the domain on the left side and exit on the right, the amount of particles per step changes. A circular boundary in the mid of the domain causes a repeating pattern of swirling vortices, the vortex street.

We compare the error measured from the different techniques for sampling the first time step in Fig. 9 (left). Since the original dataset already contains well distributed samples, as a result of the SPH simulation, the Poisson disk sampling can correctly sample the dataset even for large sample counts. Still, the void-and-cluster techniques consistently lead to the lowest error. The decreased error of stratified kd-tree sampling and entropy random sampling compared to naïve random sampling implies that both stratification and entropy-based sampling are beneficial for this dataset.

We sample 10% of the trajectories in the discrete time interval \([0, 20]\). Since particles frequently enter and exit the domain, we do not explicitly stop trajectories to sample new ones. In Fig. 8 we plot the error over time for the different sampling strategies. The void-and-cluster strategy is not only consistently better, but also stays nearly constant over time. In contrast, we observe an increase of the error over time for the random sampling techniques. Lastly, the entropy-based
Fig. 10: The von Kármán vortex street after ten time steps using random (a), uniform (b), and entropy (c) void-and-cluster trajectory sampling. The corresponding \( u \)-velocity fields are shown, which have been created using scattered data interpolation. Our error measure is shown in (d). The entropy void-and-cluster sample distribution in the first time step is illustrated in (e).

(a) Random sampling and reconstruction in \( t_{10} \)  
(b) Void-and-cluster uniform sampling and reconstruction in \( t_{10} \)  
(c) Void-and-cluster entropy sampling and reconstruction in \( t_{10} \)  
(d) Entropy sampling error in \( t_{10} \)  
(e) Entropy sampling distribution in \( t_1 \)

Fig. 11: The surface-mounted cylinder, after sampling 466,103 particles, is shown in (a). We use the continuous level-of-detail, in addition to a transfer function, to further reduce the amount of particles. Slices of the dataset using the uniform (b) and entropy (c) sampling illustrate the difference between the sampling strategies. The entropy strategy samples the less interesting region above the empty cylinder less densely.

(a) We visualize the particles using sphere and arrow glyphs with our level-of-detail  
(b) Uniform sampling  
(c) Entropy sampling

Fig. 12: We create a histogram (a) and a scatter plot (c) of the surface-mounted cylinder, after sampling 466,103 particles using the void-and-cluster entropy strategy. With our level-of-detail, we select a subset of 10,000 particles and create a histogram (b) and a scatter plot (d).

(a) Histogram from 466,103 samples  
(b) Histogram from 10,000 samples  
(c) Scatter plot from 466,103 samples  
(d) Scatter plot from 10,000 samples
We have computed the FTLE after sampling 10% of the trajectories by random sampling (b) and using the uniform void-and-cluster (c) technique. The reference, computed on all trajectories, is shown in (a).

A comparison between random sampling, uniform, and entropy void-and-cluster sampling after 10 time steps is shown in Fig. 10. Although all sampling strategies deteriorate slightly over time, the samples are still well distributed for the uniform void-and-cluster sampling approach. We reconstruct the $u$-velocity field using scattered data interpolation. The results are considerably better for the void-and-cluster approaches. Moreover, the entropy sampling strategy leads to a better reconstruction compared to the uniform void-and-cluster technique. The Arnold-Beltrami-Childress (ABC) flow is a three-dimensional, steady velocity field:

$$\begin{align*}
\dot{x} &= A\sin z + C\cos y \\
\dot{y} &= B\sin x + A\cos z \\
\dot{z} &= C\sin y + B\cos x.
\end{align*}$$

We set $A = \sqrt{3}, B = \sqrt{3}, C = 1$. We represent the flow in the Lagrangian basis with 134,217,728 trajectories that start in the spatial domain $[0,2\pi]^3$ and are integrated using a 4th-order Runge-Kutta scheme over the time interval $[0,10]$. We then sample 10% of the trajectories, without stopping and starting new trajectories. In Fig. 11 (g) and (h), 50,000 and 500 trajectories are shown as illuminated pathlines using the continuous level-of-detail that is implicitly given by the rank of our sampling strategy. Since we reorder the samples by their rank, we only have to load the first samples for visualization and can load additional samples progressively.

After random sampling and uniform void-and-cluster sampling, we have computed the forward finite-time Lyapunov exponent (FTLE). This quantity measures how neighboring trajectories separate over time and is used to visualize a time-dependent flow behavior. A slice of the FTLE is shown in Fig. 13. Computing the FTLE after sampling with the uniform void-and-cluster strategy yields better results compared to random sampling, even though the same number of samples have been taken.

6.5 Dark Sky

The Dark Sky simulations are a series of cosmological N-body simulations of the evolution of the large-scale universe [20]. We study a subset that consists of 111 million particles with a position, velocity, and unique identifier. Fig. 14 shows a visualization of the dataset reduced to 5% using the uniform void-and-cluster technique (a). A slice of the dataset is shown in (b), with the corresponding slice from the original dataset in (c).

Fig. 13: Slices of the finite-time Lyapunov exponent (FTLE) from the ABC flow. The reference, computed on all trajectories, is shown in (a). We have computed the FTLE after sampling 10% of the trajectories by random sampling (b) and using the uniform void-and-cluster (c) technique.

Fig. 14: The Dark Sky dataset reduced to 5% using the uniform void-and-cluster technique (a). A slice of the dataset is shown in (b), with the corresponding slice from the original dataset in (c).
of cosmological mass. This is not possible using Poisson disk sampling or entropy-based adaptive sampling. In contrast, our uniform void-and-cluster technique optimizes the blue noise property with respect to the spatial density of particles in the dataset. The spatial distribution is thus preserved, whilst the samples are optimally stratified.

### 6.6 Performance and Scalability

To assess the performance of our algorithm, we compare the run time of different sampling strategies. For the measurements, we use an Intel Core i7-6700 and an Nvidia Quadro RTX 8000. We enable GPU acceleration where possible.

Measurements for all of our datasets are shown in Fig. 15. We were not able to measure our implementation of LCCVD for larger datasets since it is computationally demanding and we did not parallelize it. Although Poisson disk sampling is a linear time algorithm, it is inherently sequential and leads to long run times for large data sizes. Random and stratified sampling are fast even though no GPU acceleration is used. In comparison, the void-and-cluster techniques are slower, but considering the data sizes we argue that the performance is acceptable. For example, we take 1,342,177 samples out of 134 million from the ABC flow dataset in 68 seconds. Due to the non-uniform input data of the dark sky simulation, a large kernel support is required that leads to a significantly increased run time. The use of adaptive kernel sizes or better suited data structures could potentially improve the efficiency of the neighborhood search for non-uniformly distributed datasets.

The uniform and entropy-based sampling strategies perform similar. However, for small sampling percentages the entropy computation is noticeably slower since the computation depends on the kernel size $h$, which increases for smaller sampling percentages, and scales with the input data size. This is especially visible in the ABC flow where the run-time of the entropy computation increases dramatically for smaller sampling percentages due to the neighborhood lookup limiting the GPU efficiency. In general, the run-time increases when a larger number of samples is taken, which indicates that the void filling step is the bottleneck.

Lastly, to measure the time complexity and scalability of the algorithm, we measure GPU and single-threaded CPU performance with differently sized sinc datasets. The measurements are shown in Fig. 16. Although our GPU implementation is competitive with random and stratified sampling, the single-threaded CPU implementation is considerably slower. This highlights the benefits of parallelization and GPU acceleration for our algorithm. Compared to LCCVD our single-threaded implementation achieves an enormous speed-up. Although not shown in the plot, LCCVD took more than 2,500 seconds to sample a dataset of size $10^6$.

### 7 Future Work: Multi-Node Parallelism

An important use case that has not been addressed so far is the parallel implementation for distributed memory systems. To scale the sampling technique to multiple nodes on a compute cluster, the spatial domain could be subdivided into uniform tiles. Each compute node then performs void-and-cluster sampling of one tile. Although this will produce tiles that are well distributed according to the blue noise property, the samples in the border region between two or more tiles are not necessarily well separated. If this is not acceptable, then the cluster-and-void optimization step can be applied again to the whole dataset. However, this would have to be performed on a single node which might not be possible, for example due to memory constraints. The extension of the proposed algorithm for multi-node parallelism is thus still an open problem.

### 8 Conclusion

We present a novel approach for using statistical sampling to reduce large scattered datasets for visualization and analysis. In particular, our void-and-cluster technique optimizes sample distributions with respect to the blue noise property and thus produces samples that evenly cover the spatio-temporal domain. Our technique significantly improves the accuracy of operations such as scattered-data interpolation or the computation of the FTIE. In combination with the level-of-detail given by our technique, we are able to generate interactive and clutter-free visualizations of large datasets. Lastly, we introduce an error measure to quantify the error of a subsampled dataset, which takes both spatial and value domain into account. Our results show a clear correlation between the error measure and the quality of derived quantities such as scattered data interpolation.

### Acknowledgments

The von Kármán vortex street and the surface-mounted cylinder datasets are courtesy of Thilo Dauch and Rainer Koch from the Institute of Thermal Turbomachinery at the Karlsruhe Institute of Technology.

### References

[1] A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs. Improved post hoc flow analysis via lagrangian representations. In *IEEE Symposium on Large Data Analysis and Visualization*, pp. 67–75, 2014. doi: 10.1109/LDAV.2014.7013206

[2] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: A variant of lloyd’s method. *ACM Transactions on Graphics*, 28(3):86:1–86:8, 2009. doi: 10.1145/1531326.1531392
