Permutation Learning with Only N Parameters: From SoftSort to Self-Organizing Gaussians

Kai Uwe Barthel
Visual Computing Group
HTW Berlin
Berlin, Germany
0000-0001-6309-572X

Florian Tim Barthel
Vision and Imaging Technologies
Fraunhofer HHI
Berlin, Germany
0009-0004-7264-1672

Peter Eisert
Vision and Imaging Technologies
Fraunhofer HHI
Berlin, Germany
0000-0001-8378-4805

***Abstract*—Permutation learning is essential for organizing high-dimensional data in optimization and machine learning. Current methods like Gumbel-Sinkhorn require N^2 parameters for N objects, operating on the full permutation matrix. While low-rank approximations offer some reduction to 2NM (with $M \ll N$), they still create a computational bottleneck for very large datasets. SoftSort, a continuous relaxation of the argsort operator, enables differentiable 1D sorting but struggles with multidimensional data and complex permutations. We introduce a novel method for learning permutations using only N parameters, dramatically reducing storage costs. Our method extends SoftSort by iteratively shuffling the N indices of the elements to be sorted and applying a few SoftSort optimization steps per iteration. This significantly improves sorting quality, especially for multidimensional data and complex criteria, outperforming pure SoftSort. Our method offers superior memory efficiency and scalability while maintaining high-quality permutation learning. Its drastically reduced memory requirements make it ideal for large-scale optimization tasks like "Self-Organizing Gaussians", where efficient and scalable permutation learning is critical.**

***Index Terms*—Permutation Learning, Visual Image Exploration, Sorted Grid Layouts**

I. Introduction & Related Work

Learning permutations is a fundamental challenge in machine learning, computer vision, and optimization. Many real-world problems, such as ranking, assignment, and sorting, require finding an optimal arrangement of elements [13], [15], [16]. The challenge in efficiently learning permutations lies in their discrete, binary matrix representation (one '1' per row/column), which prevents direct gradient-based optimization.

While our proposed permutation learning method has broad applicability across various fields, this paper focuses on its use in distance-preserving grid layout algorithms for color or visual image sorting (Figures 1 / 5) and 3D scene reconstruction in 3D Gaussian Splatting [7] (Figure 6). Sorting images based on similarity enhances visual perception, enabling users to efficiently explore hundreds of images simultaneously. In 3D scene reconstruction, organizing each scene attribute into a sorted 2D grid increases spatial correlation, and the resulting high-dimensional maps can be compressed using standard codecs, achieving substantial reductions without sacrificing rendering quality.

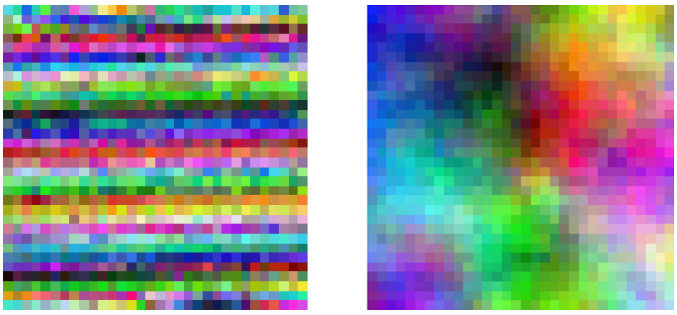

Fig. 1. Example of grid-based sorting for 1024 random RGB colors sorted by *SoftSort* (left) and the new proposed approach using the newly proposed *ShuffleSoftSort* (right). The loss function minimizes the average color distance of neighboring grid cells.

A. Permutation Learning

Permutation learning aims to estimate a permutation matrix P_{hard} that sorts input vectors x into a desired order x_{sort} according to a predefined criterion. As depicted in Figure 2, the fundamental challenge in this process is that matrix multiplication with the discrete P_{hard} is not differentiable, which prevents direct optimization. To circumvent this, a continuous relaxation, P_{soft}, is optimized via a carefully designed loss function. This function's purpose is twofold: to ensure that the final binarized P_{soft} (yielding P_{hard}) is a valid permutation and to guarantee it meets the specified optimization criteria.

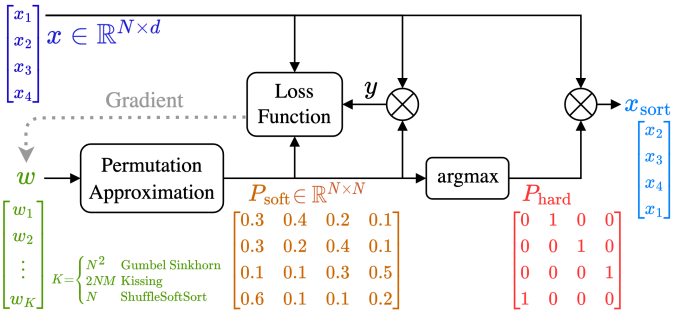

Fig. 2. Permutation learning optimizes the differentiable permutation matrix P_{soft} by adjusting the weights w based on a given loss function. The number of weights, K, depends on the chosen permutation approximation method. The input vectors x are then reordered into x_{sort} using P_{hard}, which is obtained by applying a row-wise argmax operator to P_{soft}.

This following discusses various differentiable permutation learning approaches, from existing methods to our proposed *ShuffleSoftSort*.

One of the most well-established approaches is the *Gumbel-Sinkhorn* method [11]. This technique relaxes discrete permutations into doubly stochastic matrices and applies Sinkhorn normalization, allowing for end-to-end differentiable learning of approximate permutations. While Gumbel-Sinkhorn can generate high-quality permutations, it requires storing and computing an N^2 matrix, where N is the number of elements to be sorted. This quadratic memory consumption makes it impractical for large-scale problems.

To mitigate these high memory requirements, *low-rank factorization techniques* have been proposed, such as the "Kissing to Find a Match" method by Droge et al. [4]. This approach approximates permutation matrices using a low-rank decomposition, typically involving two row-normalized matrices V and W of size $N \times M$, where $M \ll N$. The permutation matrix is then approximated by $P \approx VW^T$, followed by scaling and row-wise softmax normalization. These methods require only $2NM$ parameters, significantly reducing memory usage compared to Gumbel-Sinkhorn, though they may still be too memory-intensive for very large problems.

An alternative is *SoftSort* [14], a continuous relaxation of the *argsort* operator, which enables differentiable sorting. Unlike Gumbel-Sinkhorn or low-rank methods, SoftSort requires only N parameters, making it highly memory-efficient. However, its inherent limitation to 1D sorting restricts its use in complex permutation tasks like assignment problems or distance-preserving grid layouts.

In this paper, we extend SoftSort to learn permutations with only N parameters, specifically addressing its quality limitations and enabling multi-dimensional sorting. Our proposed *ShuffleSoftSort* technique overcomes these challenges.

B. Heuristic Distance-Preserving Grid Layout Algorithms

Distance-Preserving Grid Layouts optimize the assignment of objects or images to grid positions by aligning spatial proximity with the similarity relationships of their feature vectors. Various algorithms have been proposed for arranging vectors on a 2D grid, most of which rely on heuristic methods that offer efficient performance. However, recent research [2] has demonstrated that gradient-based learning techniques can produce sorted grid layouts with superior sorting quality compared to traditional approaches.

In the following sections, we present a brief overview of the most commonly used non-learning methods.

A *Self-Organizing Map* (SOM) [8], [9] uses a grid of map vectors with the same dimensionality as the input vectors. It assigns these vectors to the most similar grid position and iteratively updates the map vectors based on their neighborhood.

A *Self-Sorting Map* (SSM) [17], [18] initializes grid cells with input vectors and employs a hierarchical swapping process. This method compares vector similarity with the average of their grid neighborhood, considering all possible swaps within a set of four cells.

Linear Assignment Sorting (LAS) [3] merges the concepts of SOM (using a continuously filtered map) and SSM (performing cell swaps), extending the idea to optimally swap all vectors simultaneously. *Fast Linear Assignment Sorting* (FLAS) improves runtime efficiency by iterative swapping subsets, achieving sorting quality close to LAS.

Dimensionality reduction methods like t-SNE [19] and UMAP [10] can project high-dimensional vectors onto a 2D plane before grid arrangement. Linear assignment solvers, such as the Jonker–Volgenant algorithm [6], then map these positions to optimal grid locations. Fast placement strategies are discussed in [5].

C. Contributions

In [2], we introduced the first gradient-based grid sorting method using Gumbel-Sinkhorn. It ensures a valid permutation matrix while optimizing the grid arrangement for vector similarity, achieving superior sorting quality. However, its $\mathcal{O}(N^2)$ memory demand limits scalability for large datasets. To address this, we introduce *ShuffleSoftSort*, a novel SoftSort extension for high-performance, memory-efficient permutation learning. Our key contributions are:

- **Memory Efficiency:** ShuffleSoftSort uses only N parameters. It enhances SoftSort via iterated shuffling of indices, enabling large-scale swaps and improving sorting quality.
- **Superior to Low-Rank Methods:** ShuffleSoftSort outperforms low-rank approximations like *Kissing to Find a Match* with significantly lower memory consumption.
- **Scalable for Large-Scale Applications:** Drastically reducing memory needs compared to Gumbel-Sinkhorn while maintaining strong performance, ShuffleSoftSort enables scalable permutation learning for large optimization tasks, e.g., *Self-organizing Gaussians* and data visualization.

II. Proposed Method

Our new approach is based on *SoftSort*, which was originally developed as a continuous relaxation for the *argsort* operator, defined as

$$SoftSort^{\mathrm{D}}_{\tau}(w) = softmax\left(\frac{-\mathrm{D}(w_{\mathrm{sort}}, w)}{\tau}\right), \qquad (1)$$

with $\mathrm{D}(w_{\mathrm{sort}}, w)$ being the L1 distance matrix between the sorted and the unsorted elements of the vector w. Row-wise softmax increases matrix elements at positions where elements coincide, while suppressing the others. With decreasing τ the matrix converges to the permutation matrix. SoftSort produces valid permutations without iterated normalization, but is poorly suited for high-dimensional data due to its inherent one-dimensional sorting.

A limitation of SoftSort is illustrated by the 1D color toy example in Figure 3. Achieving better color ordering (by swapping yellow and magenta) necessitates traversing intermediate positions that initially degrade quality, causing gradient-based optimization to fail. Figure 1 (left) further demonstrates this in

Fig. 3. 1D color arrangement highlighting SoftSort's challenges (see text).

a 2D grid-based RGB color sorting scenario. Because SoftSort uses a one-dimensional weight vector to represent element order, any learned position change is restricted to this single dimension, proving inadequate for complex reordering across several rows.

Our newly proposed approach, *ShuffleSoftSort*, retains the advantages of SoftSort, such as low memory usage and the ability to easily achieve valid permutations, while improving the quality of the permutations compared to SoftSort. Since SoftSort can only perform one-dimensional sorting, it struggles with problems like those shown in Figures 1 or 3. To solve this problem, we circumvent the one-dimensional constraint without altering the fundamental principle.

Our method iteratively reorganizes the elements' indices and then applies SoftSort to these updated one-dimensional indices. By dynamically reordering the 1D input, we enable element repositioning not achievable with a static input order.

Figure 4 highlights ShuffleSoftSort's core concept, demonstrating how its repeated shuffled index reorganization dramatically improves sorting flexibility. Notably, the loss function is computed on reverse-shuffled elements. Training involves gradually decreasing τ, which minimizes the influence of more distant elements. Consequently, sorting quality improves with an increased number of iterations (R). Algorithm 1 presents the ShuffleSoftSort procedure, which sorts the N elements of a vector x, where each element has dimension d.

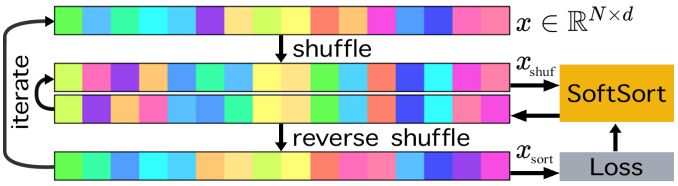

Fig. 4. ShuffleSoftSort enhances sorting by iteratively applying SoftSort to randomly shuffled elements. The loss is computed on the reverse-shuffled output, refining the permutation and addressing SoftSort's limitations.

Algorithm 1 ShuffleSoftSort

for $r = 0$ to R **do** ▷ Perform global iterations
  $\tau = \tau_{\text{start}}\left(\frac{\tau_{\text{end}}}{\tau_{\text{start}}}\right)^{\frac{r}{R}}$ ▷ Decrease τ from τ_{start} to τ_{end}
  w = arange(0, N) ▷ Initialize SoftSort weights w linearly
  shuf_idx = randperm(N) ▷ Generate random shuffle indices
  x_shuf = x[shuf_idx] ▷ Apply shuffle to input vector x
  for $i = 1$ to I **do** ▷ Perform SoftSort inner optimization steps
    perm_soft = soft_sort(w, τ) ▷ Compute soft permutation matrix
    x_sort_soft = perm_soft @ x_shuf ▷ Apply soft permutation
    x_sort_soft[shuf_idx] = x_sort_soft ▷ Reverse shuffle for loss
    get_loss(x_sort_soft, perm_soft).backward() ▷ Loss backward pass
  sort_idx = $argmax$(perm_soft, -1) ▷ Extract hard permutation indices
  x_sort = x[shuf_idx] = x_shuf[sort_idx] ▷ Apply hard permutation

For grid-based sorting with ShuffleSoftSort, we use a loss function that can be efficiently computed in a separable manner. It builds on [2], incorporating the *neighborhood loss* L_{nbr} (the normalized average distance of neighboring grid vectors in horizontal and vertical directions). However, it avoids the computationally and memory-intensive distance matrix loss. We define our loss function as follows (using P for P_{soft}):

$$L(P) = \underbrace{L_{nbr}(P)}_{\text{smoothness term}} + \underbrace{\lambda_s L_s(P) + \lambda_\sigma L_\sigma(P)}_{\text{regularization terms}} . \quad (2)$$

We obtained the best results by combining two loss components to generate valid permutation matrices. The first regularization term, the *stochastic constraint loss* $L_s(P)$, ensures that P_{soft} converges to a doubly stochastic matrix by penalizing deviations from 1 in the column sums of the matrix:

$$L_s(P) = \frac{1}{N}\sum_j \left(\left(\sum_i P_{ij}\right) - 1\right)^2 \quad (3)$$

The $L_\sigma(P)$ term is the *standard deviation loss*. It encourages P_{soft} to preserve original feature statistics by minimizing the sum of absolute differences between the column-wise standard deviations of x and $y = P_{\text{soft}} \cdot x$, normalized by the sum of $\boldsymbol{\sigma}_x$. $\boldsymbol{\sigma}_x$ and $\boldsymbol{\sigma}_y$ are the vectors of column-wise standard deviations for x and y, respectively.

$$L_\sigma(P) = \frac{\sum |\boldsymbol{\sigma}_x - \boldsymbol{\sigma}_y|}{\sum \boldsymbol{\sigma}_x} \quad (4)$$

The following table summarizes the properties of the discussed permutation approximation methods, along with the characteristics of our proposed *ShuffleSoftSort* scheme. Stability here means how certain it is to obtain a valid permutation matrix without duplicates.

	Gumbel-Sinkhorn	Kissing	SoftSort	**Ours**
Number of Parameters K	N^2	$2NM$	N	N
Non-iterative normalization	no	yes	yes	yes
Quality	++	+	–	++
Stability	+	o	++	++

TABLE I
COMPARISON OF PERMUTATION APPROXIMATION METHODS

The proposed new sorting strategy extends to other permutation learning tasks, with the key idea of continuously modifying the index order to allow for greater flexibility in sorting. For a memory-efficient implementation, it is crucial to compute the permutation matrix and the loss components in a row-wise manner, as the complete storage would require N^2 elements.

III. Experimental Evaluation

In this section, we compare Gumbel-Sinkhorn, Kissing to Find a Match, and SoftSort with our newly introduced ShuffleSoftSort. The evaluation is conducted using 1024 randomly generated RGB colors. For consistency, the loss function from [2] is employed for the first three methods, while ShuffleSoftSort uses its adapted version from Eq. 2.

Training parameters were set as follows: regularization parameters $\lambda_s = 1$ and $\lambda_\sigma = 2$; a learning rate of 0.3; $\tau_{\text{start}} = 1$ and $\tau_{\text{end}} = 0.1$. The global iterations R were set to 2500, and the inner SoftSort iterations I to 3. While ShuffleSoftSort's inner SoftSort loop typically yields a valid permutation matrix, SoftSort iterations are extended in very rare cases where permutation matrix columns contain duplicates, ensuring a valid permutation is ultimately achieved.

In grid-based sorting scenarios, ShuffleSoftSort's performance can be further enhanced. The shuffling process can be adapted to progressively reduce the range of possible positional changes during training. This ensures that towards the end of training, only closely located positions are swapped. Furthermore, it's beneficial to alternate the indexing of 2D positions between row-wise and column-wise. This strategy enables fine-grained positional adjustments in both horizontal and vertical directions.

The table below compares these methods based on memory requirements for learnable parameters, runtime on an Apple M1 Max, and sorting quality, measured by Distance Preservation Quality (DPQ_{16}). As demonstrated in [3], DPQ_{16} is a perceptually driven metric that closely aligns with human visual judgment and strongly correlates with the mean similarity to neighboring elements. Figure 1 shows the images obtained with SoftSort and ShuffleSoftSort.

Method	**Memory** ↓	**Runtime [s]** ↓	**Quality** ↑
Gumbel-Sinkhorn [11]	1048576	226.8	0.913
Kissing [4]	26624	114.4	−*
SoftSort [14]	1024	110.7	0.698
ShuffleSoftSort (ours)	1024	91.5	0.909

*) invalid permutation

TABLE II
EVALUATING PERMUTATION LEARNING METHODS ON COLOR SORTING

Further experiments are needed to optimize the parameter settings. However, this table already demonstrates the overall feasibility and effectiveness of the proposed approach, achieving high quality with lowest memory and computational demands. The runtimes are not optimal and only have a relative meaning, as the implementations were not created as optimized GPU applications. ShuffleSoftSort benefits from the fact that its loss function is considerably simpler than that of the other schemes. Notably, the kissing approach exhibits poor convergence due to its simple softmax normalization, often failing to produce valid permutation matrices.

IV. APPLICATIONS

A. Grid-based Image Sorting

Viewing large volumes of images is a cognitive challenge, as human perception becomes overwhelmed when too many images are displayed at once. To address this, most applications limit the visible images to around 20. However, sorting images based on the similarity of their visual feature vectors enhances navigation, allowing users to view hundreds of images simultaneously. This method is particularly useful for stock agencies and e-commerce platforms, where efficient browsing is key.

Visual feature vectors, generated through image analysis or deep learning, help organize images by content. Low-level feature vectors, with tens of dimensions, are often more effective for sorting larger image sets, as they enable users to group similar images easily. In that study [3] and the sorting shown in Figure 5, we used 50-dimensional low-level feature vectors to capture the key visual properties for sorting. Applying grid-based sorting to these vectors results in organized, visually meaningful layouts, as shown below.

Fig. 5. Sorting example of a dataset of e-commerce images, simplifying navigation, browsing, and retrieval of large image databases.

B. Self-Organizing Gaussians

Another application for large scale permutation learning can be found in the field of 3D scene reconstruction, specifically in *3D Gaussian Splatting* (3DGS) [7]. 3DGS has found a lot of attention in recent years, as it solves several challenges in 3D scene reconstruction, such as rendering performance, photorealism, explicitness, or explainability compared to prior methods. On the downside, however, 3DGS suffers from very large storage sizes. This is because 3DGS stores a 3D scene as an unstructured point cloud with millions of data points, each consisting of several parameters like position, scale, orientation, opacity, base color and optionally spherical harmonics. As a result, larger scenes can take up several gigabytes of storage, making them less portable. This has motivated significant research [1] for efficient compression of these datasets. In contrast to other multimedia data, 3DGS data exhibits an interesting property, namely the ambiguity for point ordering. Any reshuffling of Gaussian Splats will lead to exactly the same results, which has been exploited

by *Self-Organizing Gaussians* (SOG) [12]. Instead of storing the data points in vectors, SOG creates a sorted 2D grid for each of the scenes attributes, as shown in Figure 6, increasing spatial correlation of neighboring points. By incorporating a 2D smoothness loss also in the optimization process of the ambiguous splat configurations, further gains can be achieved. The high dimensional maps are then compressed using standard image compression codecs, allowing for up to 40x storage reduction without compromising in rendering quality. The original SOG uses a heuristic non-differentiable sorting, given the large number of data points N. However, the proposed method offers gradient-based permutation learning also for such large datasets, requiring only the storage of N parameters instead of N^2. This allows for optimally sorting millions of data points without exceeding the memory capacity and enabling end-to-end learning for scene reconstruction.

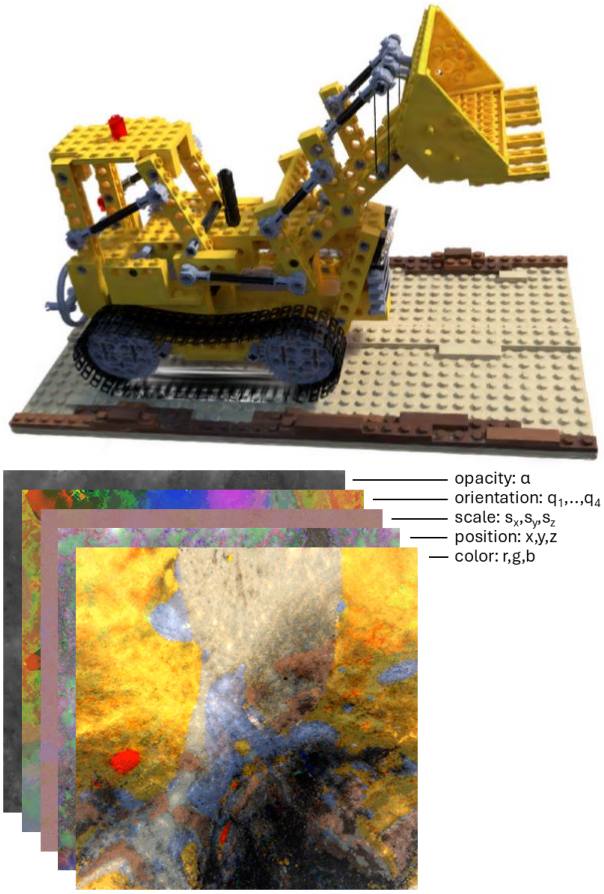

Fig. 6. Top: photorealistic rendering of a Gaussian Splatting scene; bottom: same Gaussians with all attributes sorted into a 2D grid using [12], which enforces high spatial correlation of points for efficient compression.

V. Conclusion

This work presented ShuffleSoftSort, a novel approach to permutation learning designed for high sorting quality with drastically reduced memory requirements. Our method achieves this by employing an iterative shuffling mechanism, enabling efficient N-parameter permutation learning, a significant advance over Gumbel-Sinkhorn and low-rank approximations. The demonstrated strong performance and superior memory efficiency make ShuffleSoftSort highly valuable for diverse large-scale optimization tasks, including Self-Organizing Gaussians, data visualization, and other complex permutation problems. Its versatility is further exemplified by applications like solving Sudokus, which can be explored in our GitHub repository. Future research will focus on further optimizing and extending these differentiable sorting techniques for multidimensional applications. Our code and experiments can be found at:

https://github.com/Visual-Computing/ShuffleSoftSort

References

[1] M. Bagdasarian, P. Knoll, Y. Li, F. Barthel, A. Hilsmann, P. Eisert, and W. Morgenstern. 3dgs.zip: A survey on 3d gaussian splatting compression methods. In *Proc. Eurographics*. London, UK, May 2025.

[2] K. U. Barthel, F. T. Barthel, P. Eisert, N. Hezel, and K. Schall. Creating sorted grid layouts with gradient-based optimization. In *Proc. International Conference on Multimedia Retrieval (ICMR)*, ICMR '24, p. 1199–1206, 2024. doi: 10.1145/3652583.3657585

[3] K. U. Barthel, N. Hezel, K. Jung, and K. Schall. Improved evaluation and generation of grid layouts using distance preservation quality and linear assignment sorting. *Computer Graphics Forum*, 42(1):261–276, 2023. doi: 10.1111/cgf.14718

[4] H. Droge, Z. Lahner, Y. Bahat, O. Martorell, F. Heide, and M. Möller. Kissing to find a match: Efficient low-rank permutation representation. In *Proc. Conf. Neural Information Processing Systems (NeurIPS)*. New Orleans, USA, Dec. 2023.

[5] G. M. Hilasaca, W. E. Marcílio-Jr, D. M. Eler, R. M. Martins, and F. V. Paulovich. A grid-based method for removing overlaps of dimensionality reduction scatterplot layouts. *IEEE Trans. on Visualization and Computer Graphics*, Aug. 2024.

[6] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[7] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (Proc. Siggraph)*, 42(4), Jul. 2023.

[8] T. Kohonen. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

[9] T. Kohonen. Essentials of the self-organizing map. *Neural Networks*, 37:52–65, 2013.

[10] L. McInnes and J. Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *CoRR*, 2018.

[11] G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning permutations with gradient descent and the sinkhorn operator. In *Proc. Int. Conf. on Learning Representations (ICLR)*. Vancouver, Canada, 2018.

[12] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 18–34. Milan, Italy, Oct. 2025. doi: 10.1007/978-3-031-73013-9_2

[13] F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen. Monotonic differentiable sorting networks. In *Proc. Int. Conference on Learning Representations (ICLR)*, Apr. 2022.

[14] S. Prillo and J. M. Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *Proc.37th International Conference on Machine Learning, (ICML)*, vol. 119, pp. 7793–7802, 2020.

[15] R. Santa Cruz, B. Fernando, A. Cherian, and S. Gould. Visual permutation learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3100–3114, 2018.

[16] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

[17] G. Strong and M. Gong. Data organization and visualization using self-sorting map. In *Proc. Graphics Interface*, pp. 199–206, 2011.

[18] G. Strong and M. Gong. Self-sorting map: An efficient algorithm for presenting multimedia data in structured layouts. *IEEE Trans. Multim.*, 16(4):1045–1058, 2014.

[19] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.