An Efficient Dual-Hierarchy t-SNE Minimization

Mark van de Ruit, Markus Billeter, and Elmar Eisemann

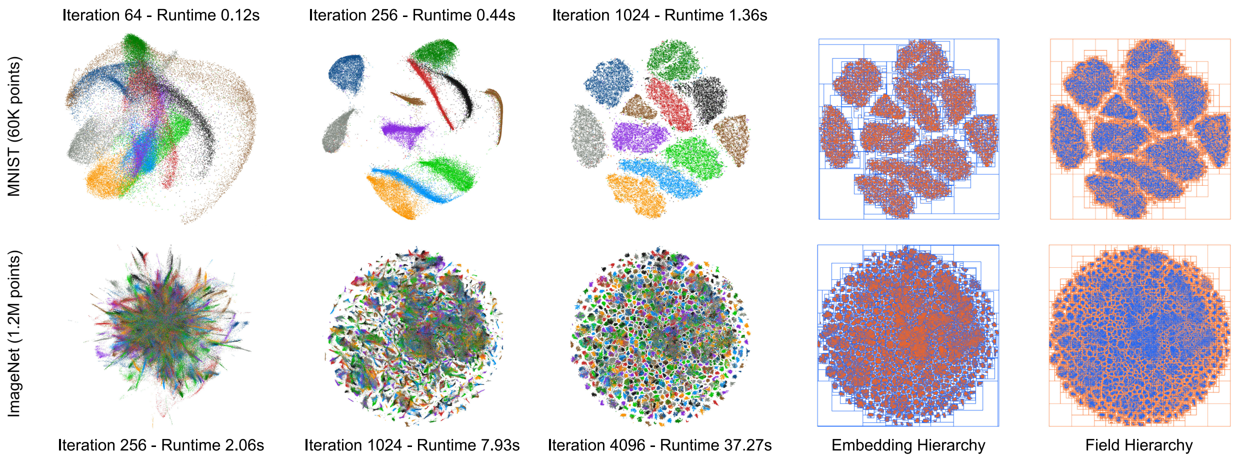


Fig. 1: Our method leverages a pair of spatial hierarchies over the embedding (center right) and a field (far right) over the embedding space to accelerate t-SNE minimization. Progression of minimizations (left) using these hierarchies is shown for a 60K point MNIST dataset (top) and a 1.2M point ImageNet dataset (bottom). The hierarchies are visualized for the last iteration of minimization.

Abstract— t-distributed Stochastic Neighbour Embedding (t-SNE) has become a standard for exploratory data analysis, as it is capable of revealing clusters even in complex data while requiring minimal user input. While its run-time complexity limited it to small datasets in the past, recent efforts improved upon the expensive similarity computations and the previously quadratic minimization. Nevertheless, t-SNE still has high runtime and memory costs when operating on millions of points. We present a novel method for executing the t-SNE minimization. While our method overall retains a linear runtime complexity, we obtain a significant performance increase in the most expensive part of the minimization. We achieve a significant improvement without a noticeable decrease in accuracy even when targeting a 3D embedding. Our method constructs a pair of spatial hierarchies over the embedding, which are simultaneously traversed to approximate many N-body interactions at once. We demonstrate an efficient GPGPU implementation and evaluate its performance against state-of-the-art methods on a variety of datasets.

Index Terms—High dimensional data, dimensionality reduction, parallel data structures, dual-hierarchy, GPGPU

1 Introduction

The exploration of high-dimensional data has received significant interest. Non-linear dimensionality reduction techniques have made it possible to visualize structures in large-scale high-dimensional datasets, leading to discoveries in many different domains, such as immunology [23] and forensic analysis [13]. The ability to successfully preserve local structures in the data is especially important. The *t-Distributed Stotachstic Neighbour Embedding* (t-SNE) algorithm [25] achieves this goal by matching pairwise similarity distributions, representing the original data in the high-dimensional space and a possible embedding in a low-dimensional space. The algorithm consists of two phases. First, a similarity distribution is constructed over the high-dimensional data. Second, a minimization is performed using the Kullback-Leibler (KL) divergence [9] between this distribution and a low-dimensional distribution, which is initially constructed over a random embedding.

Both phases of the t-SNE computation are costly operations, becoming impractical for very large datasets. While significant effort has been invested into lowering the computational cost of the similarity computations [12, 18, 19, 22, 24], the minimization remains costly. Here, efforts have focused on efficiently mapping the minimization to GPU hardware [2, 7] or on reducing the $\mathcal{O}(N^2)$ runtime complexity; the commonly used Barnes-Hut t-SNE (BH-SNE) [24] initially obtained a $\mathcal{O}(N \log N)$ runtime complexity, and $\mathcal{O}(N)$ complexities were achieved afterwards by both Linderman et al. [11] and Pezotti et al. [20]. While effective for smaller 2D embeddings, millions of points remain costly and there is a significant overhead in 3D.

Our work introduces a pair of sparsely constructed spatial hierarchies to accelerate the t-SNE minimization. The first hierarchy is constructed over the embedding, and the second over a discretization of the embedding's space. We approximate N-body computations, a costly part of the t-SNE minimization, by computing interactions between the two hierarchies using a dual-hierarchy traversal. During traversal, we eliminate the majority of these interactions using an improved formulation of the BH-SNE approximation [24]. While our minimization retains a $\mathcal{O}(N)$ runtime complexity, the number of considered interactions is significantly reduced. As N-body computations previously dominated the runtime of t-SNE for two- and especially three-dimensional embeddings, our method provides a strong improvement, significantly outperforming the state-of-the-art while generating high-quality embeddings. Further, our method is designed with GPGPU programming in mind, leveraging the compute capabilities of modern GPUs.

We first formally introduce t-SNE (Sect. 2) and related work (Sect. 3). We then cover our method (Sect. 4), its implementation details (Sect. 5), and evaluation (Sect. 6), before concluding (Sect. 7).

• *Mark van de Ruit is with the Delft University of Technology. E-mail: m.vanderuit-1@tudelft.nl.*
• *Elmar Eisemann is with the Delft University of Technology. E-mail: e.eisemann@tudelft.nl.*
• *Markus Billeter is with the University of Leeds. E-mail: m.billeter@leeds.ac.uk.*

2 t-SNE

t-SNE models a dataset of points $X = x_1, \ldots, x_N$ in a high-dimensional space through pairwise similarities, represented as a symmetric joint probability distribution P. Likewise, a randomly initialized *embedding* of low-dimensional points $Y = y_1, \ldots, y_N$ is represented in a similarity distribution Q. The goal of t-SNE is to minimize the difference between P and Q according to a cost function.

The distribution P, defined over the high-dimensional data points, represents the joint similarity p_{ij} between all pairs x_i and x_j. This similarity can be interpreted as the probability of these data points being near to each other in high-dimensional space. In a similar manner, the similarity between representative low-dimensional embedding points y_i and y_j is represented as q_{ij}. To minimize the difference between P and Q, the cost function C is used

$$C(P,Q) = KL(P \parallel Q) = \sum_{i=1}^{N} \sum_{j \neq i}^{N} p_{ij} \ln(\frac{p_{ij}}{q_{ij}}), \tag{1}$$

which is the KL-Divergence between P and Q. During minimization the positions of embedding points are updated to minimize this cost. The joint similarity p_{ij} is modeled through centering of a pair of Gaussian kernels on either high-dimensional data point as

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}, \tag{2}$$

where

$$p_{j|i} = \frac{\exp(-(\|x_i - x_j\|^2)/(2\sigma_i^2))}{\sum_{k \neq i}^{N} \exp(-(\|x_i - x_k\|^2)/(2\sigma_i^2))} \tag{3}$$

and variance σ_i is defined according to the local density in the high-dimensional space around x_i. As $p_{j|i}$ acts on a local neighbourhood outside of which influence diminishes rapidly, the effective number of considered points is typically much lower than N. It is instead based on a user-controlled *perplexity* value μ, and σ_i is then chosen such that

$$\mu = 2^{-\sum_j^N p_{j|i} \log p_{j|i}} \tag{4}$$

holds for each i. For the low-dimensional similarity q_{ij}, a *Student's t-Distribution* with one degree of freedom is used instead of a Gaussian distribution. q_{ij} is defined as

$$q_{ij} = \Big((1 + \|y_i - y_j\|^2)Z\Big)^{-1}, \tag{5}$$

where

$$Z = \sum_{k=1}^{N} \sum_{l \neq k}^{N} \Big(1 + \|y_k - y_l\|^2\Big)^{-1}. \tag{6}$$

Intuitively, to ensure that distribution Q closely represents P, their local neighbourhoods should match each other. Hence, the algorithm iteratively moves randomly-initialized embedding points around to match this criterion. This movement stems from a gradient descent applied to C. In each iteration, the gradient is computed and subsequently used to update the positions of the embedding points relying on its analytical formulation over y_i:

$$\frac{\delta C}{\delta y_i} = 4(Z \sum_{j \neq i}^{N} p_{ij} q_{ij} (y_i - y_j) - \sum_{j \neq i}^{N} q_{ij}^2 Z (y_i - y_j)) \tag{7}$$

$$= 4(F_i^{attr} - F_i^{rep}). \tag{8}$$

As shown, the gradient is decomposed into F^{attr} and F^{rep}, which allows for a potential reformulation as an *N-body problem*, where each of the N embedding points exerts attractive and repulsive forces on surrounding points. As is typical for N-body problems, the computational complexity is $\mathscr{O}(N^2)$.

3 Related Work

After the introduction of t-SNE [25], *Barnes Hut SNE* (BH-SNE) [24], reduced the runtime complexity to $\mathscr{O}(N \log N)$, and memory complexity to $\mathscr{O}(N)$. It models the similarity computation in Equation 3 as a k-nearest-neighbour (KNN) graph problem, computed using *Vantage Point trees* [28]. In addition, a Barnes-Hut approximation [1], previously used in physics calculations, significantly reduces the number of force computations in the N-body problem.

More recent developments can be divided into two areas: improving similarity computations and improvements/replacements of the minimization algorithm. Early on, *Approximated tSNE* (A-SNE) [19], relied on principles of *Progressive Visual Analytics* [16, 21] to selectively refine parts of approximate embeddings during the optimization, while replacing a precise KNN-graph with an approximate graph relying on a forest of randomized KD-trees. A similar approach was demonstrated with *LargeViz* [22], which instead leverages randomized projection trees to obtain similarities. In addition, it links the minimization's objective function to a probabilistic graph-visualization model, which is optimized through an asynchronous stochastic gradient descent. A rather different approach is *Uniform Manifold Approximation and Projection* (UMAP) [12], which instead performs a minimization between topological representations of the high-dimensional and low-dimensional spaces. While it provides superior performance to all t-SNE variants described so far, it has been shown to suffer from many of the same downsides [8]. Despite the improvements, current available implementations are orders of magnitude slower than more recent GPGPU solutions.

A fast GPU-based approach is CUDA-SNE [2]. The method approximates KNN in a manner similar to A-SNE [19] with the GPU-based FAISS library [6], and maps the BH-SNE [24] optimization to a GPGPU programming environment. Although this approach achieves good performance on large datasets, it largely relates to engineering optimizations and remains bound by $\mathscr{O}(N \log N)$ runtime complexity.

More recently, linear runtime complexity was reached by *Fast Fourier transform accelerated interpolation-based t-SNE* (FIt-SNE) [11], demonstrated with a CPU-based implementation. It uses an alternative approximation for computing repulsive forces by redefining them in terms of a convolution over an equispaced grid, which is subsequently interpolated to recover repulsive forces.

A similar GPU-based approach was developed by Pezotti et al. [20], named *GPGPU linear complexity t-SNE* (L-SNE). The authors rewrite Equation 8 as a function of *scalar* and *vector fields* — continuous functions assigning scalar or vector values to positions in space — which are then approximated in a discrete format using a GPU texture in $\mathscr{O}(FN)$ time (where F is the size of the discrete texture). Afterwards, force components are recovered through texture interpolation, which is highly efficient on GPUs. The method's runtime is dominated by the computation of this *field texture*, which suffers from scaling in either F or N and is particularly inefficient for 3D embeddings. In the following, we briefly cover this field-based formulation before presenting our approach, which avoids these shortcomings.

Given are the scalar and vector fields $\mathscr{S} : \mathbb{R}^d \to \mathbb{R}$ and $\mathscr{V} : \mathbb{R}^d \to \mathbb{R}^d$, d being the dimensionality of the embedding, typically 2 or 3. At an arbitrary position p the fields are defined as

$$\mathscr{S}(p) = \sum_{i}^{N} \Big(1 + \|y_i - p\|^2\Big)^{-1}, \tag{9}$$

$$\mathscr{V}(p) = \sum_{i}^{N} \frac{y_i - p}{\Big(1 + \|y_i - p\|^2\Big)^2} \tag{10}$$

Based on the Student's t-distribution, $\mathscr{S}$ represents the effective density of the embedding space, while $\mathscr{V}$ represents the gradient of the repulsive forces applied. Assuming for now that these fields are available, attractive forces can be approximated in a restricted neighbourhood as

$$\hat{F}_i^{attr} = \hat{Z} \sum_{\ell \in kNN(i)} p_{i\ell} q_{i\ell} (y_i - y_\ell), \tag{11}$$

as seen in BH-SNE [24]. The normalization factor $\hat{Z}$ is now approximated in linear time by consulting the scalar field:

$$\hat{Z} = \sum_{\ell=1}^{N} (\mathscr{S}(y_\ell) - 1). \tag{12}$$

The repulsive force for a single point is approximated as

$$\hat{F}_i^{rep} = \mathscr{V}(y_i)/\hat{Z}. \tag{13}$$

Computing an approximate gradient now requires linear runtime, as the fields are queried in constant time, approximated in a discrete texture format, and separately computed through a summation of the contributions of all embedding positions. Formally, positions in the fields sum kernels S and V as follows:

$$\mathscr{S}(p) = \sum_i^N S(y_i - p), \quad S(t) = \left(1 + \|t\|^2\right)^{-1}, \tag{14}$$

$$\mathscr{V}(p) = \sum_i^N V(y_i - p), \quad V(t) = t\left(1 + \|t\|^2\right)^{-2}. \tag{15}$$

While this leads to a linear runtime, there are two observations. The kernels S and V are again based on a Student's t-distribution and have limited effects on far-away positions, but are applied to all positions with full accuracy. In addition, as the kernels have a fixed support in the embedding space, the field's discrete representation must grow with the embedding as the minimization progresses, gradually becoming larger. Pezotti et al. [20] propose that $F \ll N$ generally holds. However, while the texture grows slowly in two dimensions, the addition of a third dimension (which implies a cubic scaling of F) strongly reduces potential effectiveness. While theoretically of linear runtime, the solution is not optimal when F becomes large.

4 Dual-Hierarchy t-SNE Minimization

Here, we present our approach to an efficient t-SNE minimization using the field-based formulation [20]. Our approach reduces the field texture's construction time, which dominates the original runtime and renders the solution impractical for higher embedding dimensionalities. We observe that this discrete representation in form of a texture requires evaluating many small regions with varying local interactions, but similar global interactions. We propose to represent both the embedding and the discrete field as spatial hierarchies, henceforth referred to as the *embedding hierarchy* and the *field hierarchy* respectively. We perform a dual traversal over these hierarchies, during which we employ an improvement of the approximation criterion used in BH-SNE [24] to selectively compute interactions between hierarchy nodes, which represent large regions in the embedding and the field (Fig. 2). These interactions between the regions are not directly transferred to data points but are first stored in the hierarchy itself; specifically, for a region, the interaction is added to its corresponding node of the hierarchy. Hereby, we benefit from both hierarchies. After dual traversal, we accumulate these interactions that are stored throughout the hierarchy to form a complete, yet sparsely-computed approximated field. In this way, we improve upon the original $\mathscr{O}(FN)$ complexity of the field computation, as our cost approaches $\mathscr{O}(N)$. We provide a proof in the supplementary material, but suggest to first follow the algorithm in this section to ease understanding. Fig. 3 shows an overview of our method, divided into three steps: *hierarchy construction*, *dual traversal*, and *field accumulation*. We detail each step in the following.

4.1 Hierarchy Construction

We construct hierarchies over the embedding and field (Fig. 3, first part). Meyer et al. [14] showed that a careful choice of the spatial hierarchy provides performance improvements to BH-SNE [24]. We choose our structures with efficient execution on the GPU in mind.

As embedding hierarchy, we select an implicit linear *bounding volume hierarchy* (BVH), constructed in linear time on the GPU (Sect. 5). In a BVH, each node stores an *axis-aligned bounding box* (AABB) encompassing the child-node bounding boxes, while leaf nodes directly contain one or more objects (i.e., embedding points). BVHs provide a close fit around contained data, and allow for refitting of AABBs without fully rebuilding the hierarchy. The latter is an important cost-saving measure, made possible because embedding positions move slowly during the minimization. As with BH-SNE [24], nodes in the

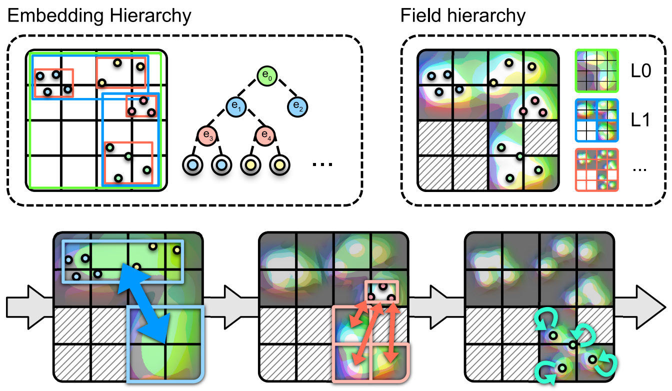
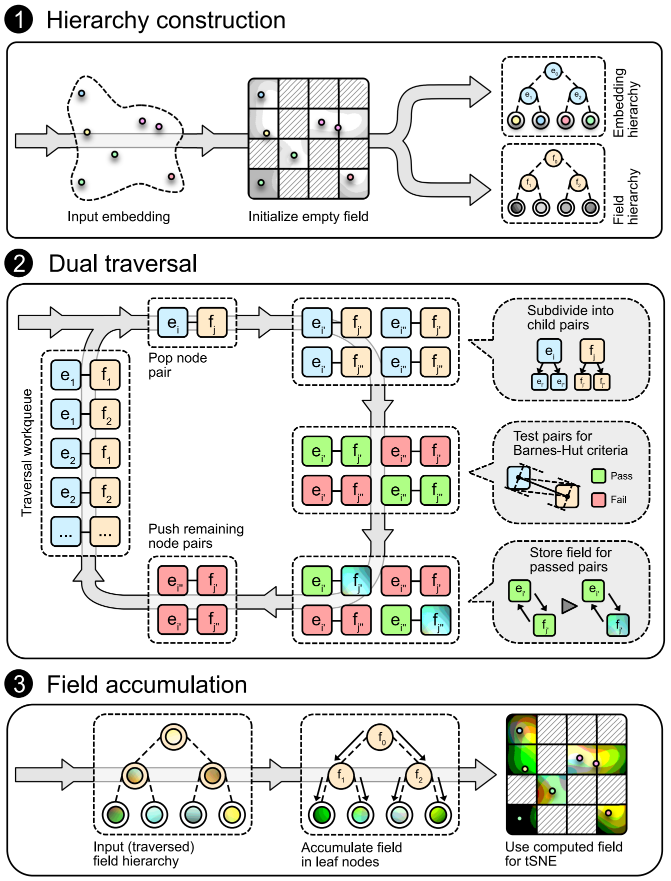


Fig. 2: We use embedding and field hierarchies (top), comparing their nodes to compute interactions between many points and large portions of the field in a single step (bottom left). Where refinement is necessary, we descend one or both hierarchies (bottom mid), continuing until points interact with a full-resolution field (bottom right).


1 Hierarchy construction
Input embedding
Initialize empty field
Embedding hierarchy
Field hierarchy
2 Dual traversal
Traversal workqueue
Pop node pair
Subdivide into child pairs
Test pairs for Barnes-Hut criteria
Pass
Fail
Push remaining node pairs
Store field for passed pairs
3 Field accumulation
Input (traversed) field hierarchy
Accumulate field in leaf nodes
Use computed field for tSNE
Fig. 3: We generate embedding and field hierarchies, and dual traverse these using a work queue. When a numeric approximation of the interactions suffices, we cull node pairs. Further, we evaluate and store interactions at different levels of detail in the hierarchies. A final traversal constructs the field used in the t-SNE minimization.

embedding hierarchy track their center of mass, defined as the average of the contained embedding points. The center of mass c_i of a node e_i with mass m_i is simply

$$c_i = \frac{1}{m_i} \sum_{j \in emb(e_i)} y_j, \tag{16}$$

where $emb(e_i)$ defines the indices of the points in the node.

Observing the discrete grid nature of the field texture, we select a sparse implicit quad-/octree for the field hierarchy. We mark cells of interest in the grid that we build our hierarchy upon. This is done in $\mathcal{O}(F \log N)$ time, but marking costs are negligible in practice ($< 1\%$ of total compute time). For each grid cell, we descend the previously generated embedding hierarchy to determine if the cell overlaps or borders embedding points. We then construct the sparse field hierarchy with the marked cells as leaf nodes. Computing the field for these locations suffices, as it will only be accessed here during the minimization. Each node f_j in the hierarchy has scalar and vector field entries $\hat{\mathscr{S}}_j$ and $\hat{\mathscr{V}}_j$, which are initialized as 0 at the start of every iteration of the minimization and used as intermediate storage during traversal. Contrary to the embedding hierarchy, nodes in the field hierarchy represent regions and their center of mass c_j is simply their region's geometric center.

4.2 Dual Traversal

With both hierarchies available, we perform a dual traversal (Fig. 3, second part), formulated as a top-down breadth-first traversal of a single, larger tree. This tree consists of nodes representing node pairs (e_i, f_j), where e_i and f_j are respectively nodes in the embedding and field hierarchies. Each node pair represents a potential interaction between the embedding points and field regions described by the two contained nodes. We model traversal using a work queue, in which we store node pairs in the dual hierarchy that still have to be traversed. At the start of traversal, a root node pair, i.e. (e_0, f_0), is pushed on the queue. During traversal, a node pair is popped from the queue, and is subsequently subdivided. We descend one level in both hierarchies under each node if possible. If both nodes are leaves, we compute the underlying interactions directly. Otherwise, the different possible pairs of child nodes from both hierarchies are tested via an approximation criterion (Sect. 4.3), to determine if they represent interactions with a sufficient accuracy. If this criterion fails for a child node pair, it is pushed on the work queue for further subdivision. If it holds, we will not further descend into the dual hierarchy underneath this child node pair but process them directly.

To process a node pair, we compute the interactions by using an approximation of the kernels in Equation 14 and Equation 15:

$$\hat{\mathscr{S}}(e_i, f_j) = m_i\, S(c_i - c_j), \tag{17}$$

$$\hat{\mathscr{V}}(e_i, f_j) = m_i\, V(c_i - c_j). \tag{18}$$

Both values are computed once and will be used for all m_i points in the embedding node and all regions under the field node instead of evaluating m_i values for potentially many field cells. These values are atomically added to $\hat{\mathscr{S}}_j$ and $\hat{\mathscr{V}}_j$ in the field node f_j.

Traversal is finished once the work queue is empty. We purposefully subdivide node pairs before testing an approximation criterion — as opposed to the inverse — for implementation reasons (Sect. 5).

4.3 Barnes-Hut Approximation

We modify the dual-hierarchy approximation criterion of BH-SNE [24] to determine whether a node pair can be processed. Originally, given lengths r_i, r_j of the diagonals of two nodes' AABBs, and node centers c_i, c_j, the following term is evaluated:

$$\frac{max(r_i, r_j)}{||c_i - c_j||} < \theta. \tag{19}$$

The parameter θ defines a maximum allowed ratio, interpreted as the tangent of an angle in a triangle whose opposite and adjacent

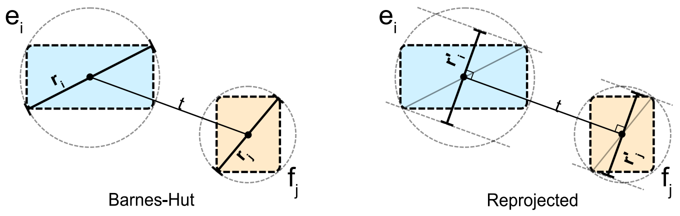


Fig. 4: Simple modification to Barnes-Hut approximation [1]. Instead of constant radii based on bounding box diagonals, we reproject diagonals, handling irregular bounding boxes regardless of their respective positions to each other.

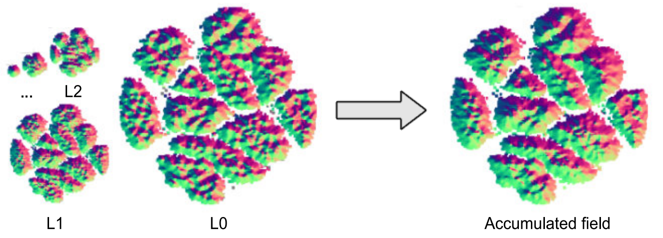


Fig. 5: The field hierarchy is ascended from leaf nodes (L0) and intermediate values in consecutive smaller levels (L1, L2, ...) are added, resulting in an approximate field. A sparse vector field is visualized, red and green colors marking x- and y-directions of the vectors.

edges have lengths $max(r_i, r_j)$ and $||c_i - c_j||$ respectively. A larger θ means larger bounding boxes closer to each other pass the test, leading to earlier processing in the hierarchy (faster traversal) but a coarser approximation. Similarly, if $\theta = 0$, the hierarchies are traversed fully, leading to an inefficient but accurate computation. For single-hierarchy traversals, θ typically lies between 0.1 and 0.5 [24]. The condition is simple as it is evaluated many times, assuming that all bounding boxes in both hierarchies have regular sides (as is the case for quad-/octrees).

Hierarchies such as a linear BVH tend to produce irregular bounding boxes that closely fit the contained data. Here, the Barnes-Hut criterion is suboptimal, as it considers a bounding box based on its diagonal, which is not a good representative of all sides when having a highly irregular bounding box. Hence, we modify Equation 19 to project the diagonals d_i, d_j of the nodes' bounding boxes so the approximation criterion accurately matches this irregularity, leading to projected diagonal lengths r_i', r_j'. We visualize our approach in Fig. 4.

To obtain projected diagonal lengths, we first compute a unit vector $\hat{t}$ along the difference $c_i - c_j$, but reflected across axes so it is near-orthogonal to the diagonals. In two dimensions, this is simply:

$$\hat{t} = \left| \frac{c_i - c_j}{||c_i - c_j||} \right| \begin{bmatrix} -1 \\ 1 \end{bmatrix}. \tag{20}$$

A suitable length is then obtained through vector rejection as:

$$r_i' = \|d_i - \hat{t}\,(d_i \cdot \hat{t})\|. \tag{21}$$

We compute r_j' in the same manner and use $\max(r_i', r_j')$ for the comparison in Equation 19. We evaluate this criterion in Sect. 6.

4.4 Field accumulation

After dual traversal, we collect the approximate interactions stored in the hierarchies (Fig. 3, third part). In particular, field hierarchy nodes now store intermediate parts of the actual fields in $\hat{\mathscr{S}}_j$ and $\hat{\mathscr{V}}_j$. As in [20], we want to interpolate the discrete field to obtain approximate field values at embedding positions, which is difficult in a hierarchy. Hence, we flatten it to recover a coarsely approximated texture, i.e., we ascend the field hierarchy upwards once for each non-empty leaf node, accumulating encountered field scalar and vector values and storing their sum in the respective texture position of said leaf node.

This requires $\mathcal{O}(F \log F)$ time when gathering upwards from a leaf to the root. Performing the operation in reverse leads to $\mathcal{O}(F)$ time, but becomes less practical on GPU hardware. Afterwards, the field can be queried for interpolated scalar and vector field values per point. Fig. 5 displays an accumulation of different levels of the field hierarchy.

5 Implementation

Our technique is implemented using a GPGPU approach. We develop our implementation in a combination of the OpenGL 4.6 API and CUDA 11, although the concepts we described can be applied on other APIs. Our implementation is available online.[1]

Mirroring the algorithmic description, our t-SNE implementation consists of two parts. We first generate the joint similarity distribution P in the same manner as Chan et al. [2], using approximate KNN information with $k = 3\mu$ obtained through the GPU-based FAISS library [6]. The formulation of the distributions remains the same as BH-SNE [24]. Second, we mirror the matrix-based minimization used by Pezotti et al. [20]. During the minimization, we invest time at the start of each iteration to rebuild or refit our spatial hierarchies, and then perform a dual-hierarchy traversal, replacing the expensive field computation.

5.1 Hierarchy Construction

As mentioned, we implement the embedding hierarchy as an implicit linear BVH, constructed on the GPU in $\mathcal{O}(N)$ time. We outline the general method, but refer the reader to Lauterbach et al. [10] for a full description. In short, the linear BVH method reduces BVH construction to a single sorting operation (Fig. 6). Each of the N embedding points is assigned a Morton code based on their discretized position in 2D/3D space. Based on these codes, the points are bucketed in leaf nodes, which are subsequently arranged along a space-filling z-order or Morton curve in a $\mathcal{O}(N)$ parallel radix sort, using the Morton codes as keys. After sorting, levels of the hierarchy are constructed iteratively by grouping nodes which share the same high order bits in their respective Morton codes. Our implementation adopts the work-queue based approach of Garanzha et al. [5]. Faster and more recent construction algorithms can be used at the cost of increased code complexity. For a parallel radix sort, we leverage the implementation available in the CUDA-based CUB library [17], which can access specific buffer objects in OpenGL through the included interoperability library.

We implement the field hierarchy as a sparse implicit quad-/octree due to the discrete nature of the field texture. As nodes in this hierarchy are regular, we do not store bounding box information, instead deriving these from node indices when necessary. The only information we store in a node is its type and the mentioned intermediate scalar and vector values used during traversal.

Although the embedding changes rapidly during early iterations, changes are less pronounced later on. Early iterations of t-SNE, typically the first 250, use *early exaggeration*, multiplying p_{ij} by some scalar to aggressively separate clusters. We use this to our advantage to reduce hierarchy-construction costs significantly. While we rebuild hierarchies on every iteration during early exaggeration, we only do so at regular intervals after. We can often simply refit bounding boxes around the newly updated positions, avoiding costly sorting. As the number of leaves in the field hierarchy can change at each iteration — leading to a substantially different spatial hierarchy — we include the cells bordering embedded points as leaves, which enables a reuse.

5.2 Dual Traversal

As described in Sect. 4, dual-hierarchy traversal is formulated as a breadth-first traversal, in which node pairs are read, subdivided, and tested for further traversal. We leverage a pair of work-queues to track traversal. At the start of traversal, an initial set of node pairs (matching to root nodes) is written to the first, or primary work-queue. During a single step of traversal, all node pairs on the primary work-queue are subdivided and tested for Equation 19. Node pairs which fail the approximation criterion are pushed on the secondary work-queue, which is subsequently swapped with the primary work-queue for the next traversal step. We repeat this process until the primary work-queue is empty or the leaf levels are reached, at which point traversal has completed.

[1] `https://www.github.com/markvanderuit/dual_hierarchy_tsne`

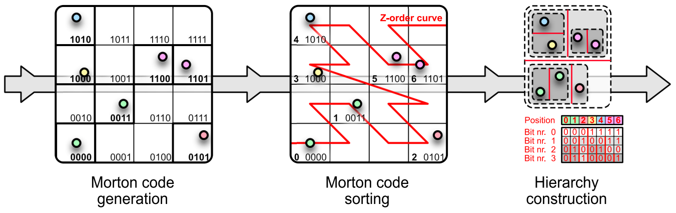


Fig. 6: Linear BVH [10] construction: points are discretized and assigned Morton codes, after which they are sorted along a spatial curve. Sorted points are then subdivided into a hierarchy based on their codes.

As root nodes encompass the entire embedding, they will always be subdivided. As an optimization, we start traversal at a lower level in both hierarchies by pushing all pairs corresponding to the selected levels on the work-queue (we use levels 3/2 for a 2D/3D embedding, leading to 4096 node pairs for hierarchies with fan-outs 4/8).

To optimize subdivision, we leverage local cross-communication capabilities of modern GPUs (subgroups in OpenGL/GLSL, warps in CUDA) to test multiple combinations of node pairs per GPU thread (invocation) while minimizing memory operations. To subdivide a single node pair on both sides, we use two threads (four for quadtrees, eight for octrees), having each thread load a single child node from both sides of the hierarchies. The total number of node pairs that must be tested (four for binary trees, 16 for quadtrees, 64 for octrees) can be obtained by rotating the child nodes on one side of the hierarchy along the 2/4/8 threads, using the subgroup capabilities.

5.3 Single-Hierarchy Fallback

As described in Sect. 3, the discrete field grows in size as the minimization progresses. At its start, the small discrete field implies few regions of interest require computation, leading to a sparse hierarchy. In this scenario, a dual traversal is inefficient as the field hierarchy's levels have too few nodes to fully occupy the GPU. We establish a maximum positive difference in depths d_e, d_f between the embedding and field hierarchies (i.e., $d_e - d_f \leq d_{max}$) to determine when dual hierarchy traversal is used. We empirically established $d_{max} = 4$ as a suitable threshold. Whenever we forego a dual-hierarchy traversal, we only construct the embedding hierarchy, and depth-first traverse it for the entire field in $\mathcal{O}(F \log N)$ time.

6 Evaluation

We first evaluate specific choices of our method, and afterwards compare against state-of-the-art solutions in terms of computational cost and embedding quality. All experiments run on a particular dataset use the same configuration and parameters. Further, all experiments are conducted on a single machine with an Intel Core i7-9900 (16 logical threads @3.1 GHz), 16 GB of DDR4 RAM and a GeForce RTX 2080 Ti GPU with access to 11 GB VRAM. For each experiment, we record the minimization runtime and resulting KL-divergence as a direct measure of how far a specific minimization has progressed. As KL-divergence is directly coupled to minimization, we additionally consider an unrelated metric, selecting Nearest-Neighbourhood-Preservation (NNP) as described by Venna et al. [26]. It measures how well local neighbourhoods in the low-dimensional embedding preserve characteristics of their high-dimensional counterparts. In order to obtain correct results for NNP, the gradient descent must be (mostly) converged. Hence, we use a larger number of iterations for larger datasets.

6.1 Datasets

We select four datasets that are frequently applied in the evaluation of dimensionality reduction algorithms such as t-SNE. As different datasets typically differ in size, dimensionality, and structure, we select

Table 1: Time and space complexities for the different stages of our method compared to other methods. N is input size, K is restricted neighbourhood size, and F is field size used by our method and L-SNE [20]. N_{int} and p are parameters of FIt-SNE [11]: N_{int} represents a discrete grid size, and p represents the number of equispaced points over parts of said grid. Note that F, N_{int} and p are independent of N.

Ours			L-SNE [20]			FIt-SNE [11]			BH-SNE [24]		
Component	Time comp.	Space comp.	Component	Time comp.	Space comp.	Component	Time comp.	Space comp.	Component	Time comp.	Space comp.
Hier. construction	$\mathcal{O}(F\log N+F+N)$	$\mathcal{O}(F+N)$									
Dual traversal	$\mathcal{O}(max(F,N))$	$\mathcal{O}(F+N)$				Point-grid interaction	$\mathcal{O}(pN)$	$\mathcal{O}(pN_{int}+N)$			
Field accumulation **	$\mathcal{O}(F\log F)$	$\mathcal{O}(F)$	Field computation	$\mathcal{O}(FN)$	$\mathcal{O}(F+N)$	Grid-grid interaction	$\mathcal{O}(pN_{int}\log pN_{int})$	$\mathcal{O}(pN_{int})$	Hier. construction	$\mathcal{O}(N)$	$\mathcal{O}(N)$
F^{rep} lookup *	$\mathcal{O}(N)$	$\mathcal{O}(F+N)$	F^{rep} lookup *	$\mathcal{O}(N)$	$\mathcal{O}(F+N)$	F^{rep} computation	$\mathcal{O}(pN)$	$\mathcal{O}(pN_{int}+N)$	F^{rep} computation	$\mathcal{O}(N\log N)$	$\mathcal{O}(N)$
F^{attr} computation	$\mathcal{O}(KN)$	$\mathcal{O}(KN)$	F^{attr} computation	$\mathcal{O}(KN)$	$\mathcal{O}(KN)$	F^{attr} computation	$\mathcal{O}(KN)$	$\mathcal{O}(KN)$	F^{attr} computation	$\mathcal{O}(KN)$	$\mathcal{O}(KN)$
Apply forces *	$\mathcal{O}(N)$	$\mathcal{O}(N)$	Apply forces *	$\mathcal{O}(N)$	$\mathcal{O}(N)$	Apply forces *	$\mathcal{O}(N)$	$\mathcal{O}(N)$	Apply forces *	$\mathcal{O}(N)$	$\mathcal{O}(N)$

* Component has negligible computational runtime (Fig. 11). ** Can be performed in $\mathcal{O}(F)$ time (Sect. 4.4).

Table 2: The numbers of points and dimensions, number of minimization iterations, and selected perplexity μ for each dataset.

Dataset	Points	Dims.	Iters.	μ
MNIST	60000	784	1000	50
Fashion	60000	784	1000	50
ImageNet	1250000	128	4000	10
Word2Vec	3000000	300	4000	5

Table 3: Compared runtimes of different methods for a 2D minimization using the full datasets listed in Table 2.

Dataset	CUDA-SNE	L-SNE	Ours
MNIST	5.86s	1.70s	1.36s
Fashion	5.74s	1.83s	1.40s
ImageNet	94.12s	346.94s	51.10s
Word2Vec	94.90s	212.11s	86.90s

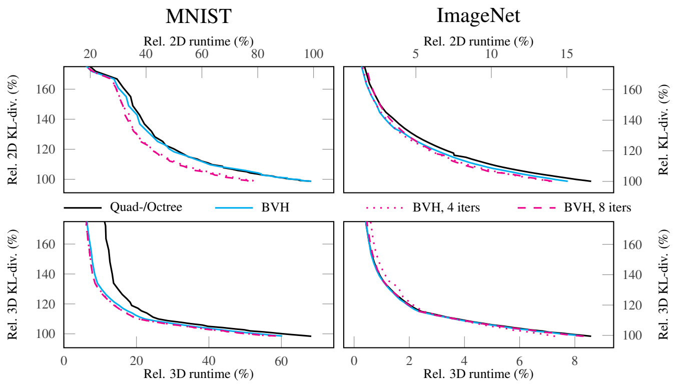


Fig. 7: Comparison of a BVH with/without refitting and a quad-/octree as a spatial hierarchy over 2D/3D embeddings of the MNIST and ImageNet datasets. We minimize for increasing numbers of iterations and plot the resulting relative runtime and KL-divergence. Baseline results are established with GPGPU linear complexity t-SNE [20].

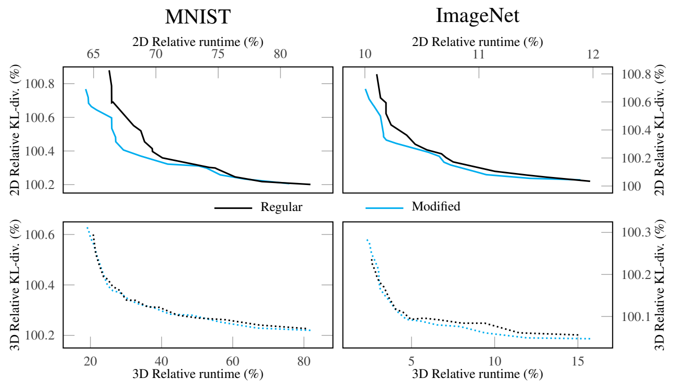


Fig. 8: Comparison of our modified Barnes-Hut approximation and the regular form for generation of 2D/3D embeddings over the MNIST and Imagenet datasets. We minimize for differing $\theta \in [0.2, 0.6]$ and plot the resulting relative runtime and KL-divergence. Baseline results are established with GPGPU linear complexity t-SNE [20].

separate iterations and perplexity μ for each dataset. Specific sizes and parameters selected for each dataset are displayed in Table 2.

The commonly-used MNIST dataset consists of labeled 28×28px grayscale images of handwritten digits, each represented as a vector storing an image's pixel values. MNIST is often used for this kind of evaluation as it contains 10 clearly-defined classes corresponding to 10 different digits. The similar Fashion-MNIST [27] contains images of 10 different types of clothing, instead of digits, which are sometimes closely related but harder to separate into clusters with an algorithm such as t-SNE. For this reason, we included it in our evaluation.

The ImageNet dataset [3] stores approximately 1000 categories of random images of objects at varying resolutions. We use a reduced and formatted version previously published by Fu et al. [4], processed such that each vector in the dataset has a dimensionality of 128.

The GoogleNews dataset stores a collection of three million words, each represented as a vector generated by Word2Vec [15]. This tool consumes a text corpus — in this case originating from Google News — and assigns words in the corpus a representative vector in such a way that words are closely related if they share a similar context.

6.2 Hierarchy Evaluation

We first evaluate our choice of spatial hierarchy. Although our method works with different hierarchies, we focus on the implicit linear BVH [10]. Use of alternatives such as a quad-/octree is possible. However, BVHs have several benefits: they fit the contained data closely, and their bounding volumes can be refitted when data changes. Refitting instead of rebuilding provides a significant reduction in runtime over consecutive iterations. We compare minimizations of MNIST and ImageNet in four cases: using quad-/octrees, using a BVH rebuilt every iteration, and using BVHs that are rebuilt after four or eight iterations of refitting. No refitting is performed in the first 250 iterations as early exaggeration takes place. Results are displayed in Fig. 7. The quad-/octree has to be rebuilt every iteration. It only matches the BVH performance when the latter is always rebuilt. The benefit of the BVH becomes apparent when refitting is used, e.g., during four iterations. However, this degrades BVH quality, and refitting for too many iterations results in unpredictable runtimes. This is seen in the ImageNet minimization for 8 iterations of refitting, where the embedding still undergoes significant changes after the early exaggeration phase. Here, refitting degrades the hierarchy quality. We show iteration runtimes in Fig. 12, where these effects are visible. In practice, we employ four consecutive iterations in all other examples.

6.3 Barnes-Hut Evaluation

Next, we evaluate our modified Barnes-Hut approximation criterion in conjunction with a BVH. This criterion handles irregular bounding volumes, which occur in a BVH, better than the original. We compare minimizations of MNIST and ImageNet with our criterion and the

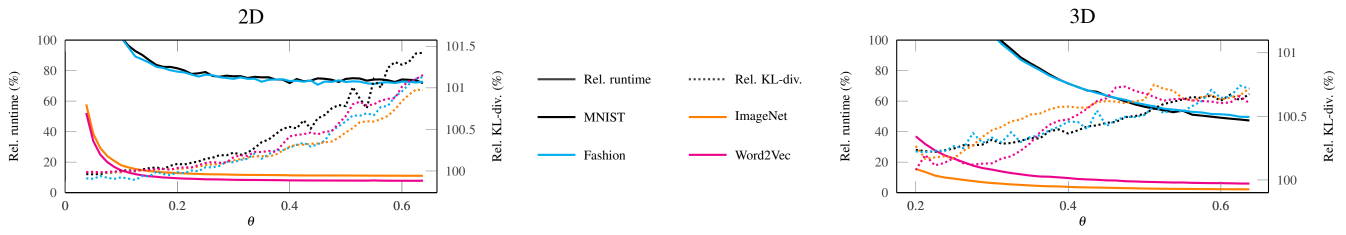


Fig. 9: Evaluation of θ for generation of 2D/3D embeddings. We show the resulting relative runtime and KL-divergence of three datasets. Baseline results ($\theta = 0$) are 100% and are established with GPGPU linear complexity t-SNE [20]. In larger datasets and small $\theta < 0.2$, 3D minimizations may exceed the memory capacity of our GPU and are not shown.

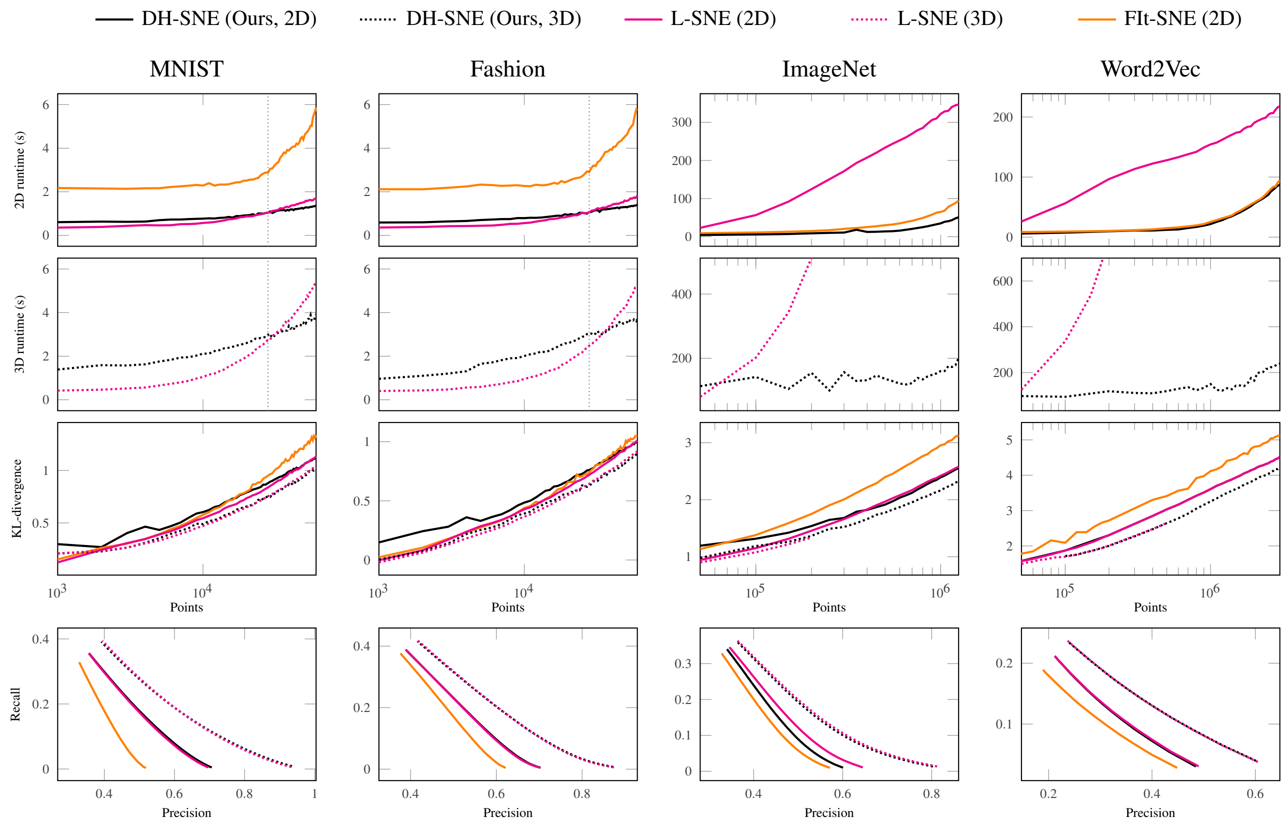


Fig. 10: Comparison of linear complexity tSNE [20], a CUDA-SNE [2] implementation of FIt-SNE [11], and our method (DH-SNE). The first row shows minimization runtimes for 2D embeddings over increasingly large subsets of data. The second row repeats this experiment for 3D. The third row shows evolution of KL-divergence over the same subsets. Horizontal axes for the first three rows are logarithmic. The fourth row shows NNP in the form of precision/recall curves. Our method outperforms the state-of-the-art on large datasets in terms of runtime for 2D and 3D. In addition, it retains a similar quality to linear tSNE [20] in terms of KL-divergence and NNP. 3D embeddings are consistently of higher quality.

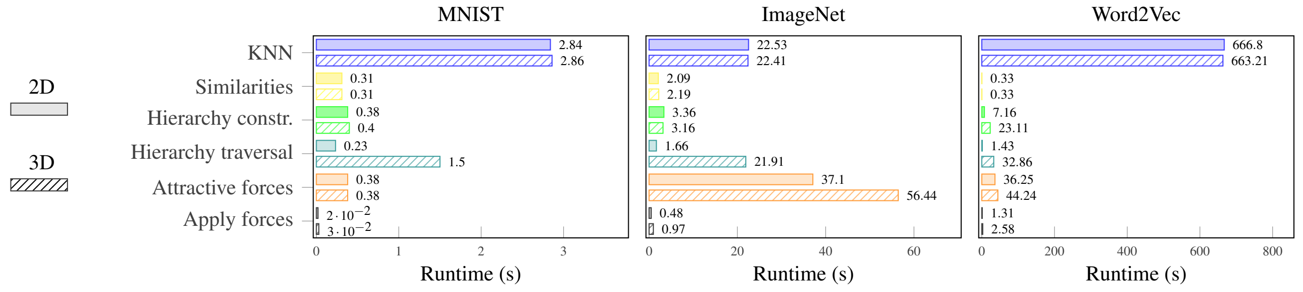


Fig. 11: Comparison of the runtimes of the most expensive components of our method on three datasets of varying sizes, for 2D/3D embeddings. The different perplexity values listed in Table 2 imply different local neighbourhood sizes, leading to varying attractive force computation costs (Equation 11). As demonstrated, runtime is dominated by KNN and attractive force computations on larger datasets.

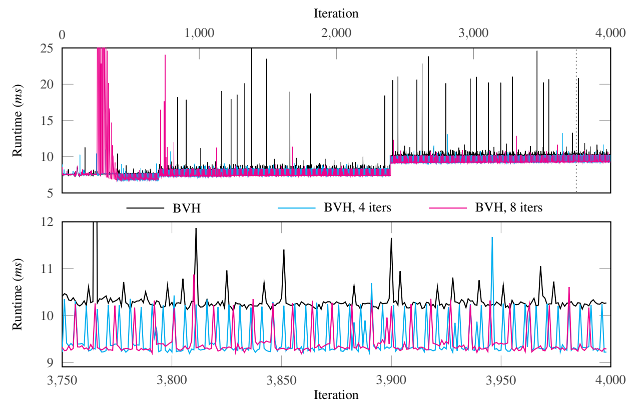


Fig. 12: Influence of refitting a BVH during minimization of the ImageNet dataset. We show runtime per iteration for all 4000 iterations (top) and the last 250 iterations (bottom). Note the spike in runtime after early exaggeration for 8 iterations of refitting.

regular criterion [1]. We test for differing $\theta \in [0.2, 0.6]$ and record resulting (relative) runtime and KL-divergence. Results are displayed in Fig. 8. While the improvements are stronger for larger values of θ, the new criterion outperforms the regular criterion in all cases.

Larger θ leads to a coarser approximation and faster traversal as nodes are culled earlier. While this parameter was evaluated in BH-SNE [24] in the context of single-hierarchy traversal, the established $\theta \leq 0.5$ does not hold for our method. In addition, the parameter's impact on traversal may vary across 2D/3D embeddings. We investigate its effect in both scenarios in Fig. 9. We consider $\theta = 0.25$ a good tradeoff for 2D, and $\theta = 0.4$ for 3D. There is a noticeable increase in KL-divergence for larger θ across all datasets, which becomes visible as grid-like patterns. The supplemental material shows generated 2D/3D embeddings for different values of θ.

6.4 Comparative Evaluation

Finally, we compare with state-of-the-art techniques with linear runtime complexity. First, we select the field-based L-SNE developed by Pezotti et al [20], with which we generate both 2D/3D comparisons. This technique shows excellent performance on smaller datasets and provides high quality embeddings in comparison with earlier techniques. We use field scalings of 2.0 (2D) and 1.2 (3D) for measurement with both our method and L-SNE. We also select a current version of CUDA-SNE [2] which, instead of a Barnes-Hut approximation, recently adapted the $\mathcal{O}(N)$ FIt-SNE [11] to the GPU. Although their approach incurs an overhead for smaller datasets, it outperforms the original implementation due to a linear runtime. This implementation only generates 2D embeddings, so we only compare it in this regard. Older BH-SNE [24] or baseline $\mathcal{O}(N^2)$ t-SNE algorithms [25] have been omitted, as their practical performance is typically orders of magnitude slower.

As our technique uses a field similar to Pezotti et al. [20], we expect to produce similar embeddings at improved runtime performance. With regards to FIt-SNE [11], we expect to reach similar or improved performance on large datasets, while producing substantially different embeddings, as our minimization differs from theirs by definition. To correctly compare the different minimization methods, we ensure they use identical KNN information and an identical joint similarity distribution P. We further ensure all methods use an identical initial embedding, and use identical parameters for their gradient descent. Differences between methods then correspond solely to the differences in their respective complexities. We provide an overview of the different time/space complexities of each method in Table 1.

The first two rows of Fig. 10 show minimization runtimes for 2D/3D embeddings separately. We run on increasingly large subsets of the datasets to show how minimization progresses. A logarithmic scale is used on horizontal axes to account for large dataset sizes. We list exact runtimes of minimizations on the full dataset in Table 3.

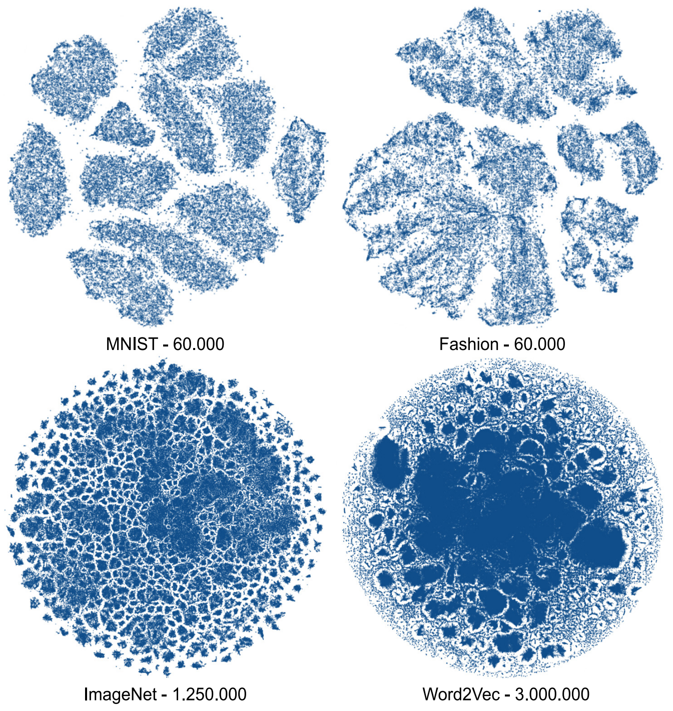


Fig. 13: Embeddings of datasets (Table 2), generated by our method.

Our technique performs exceedingly well for sufficiently large datasets; starting at approximately 27K points (indicated by a dotted vertical line) it outperforms compared methods in both the relatively small MNIST and Fashion datasets. On the full datasets, a 1000 iteration minimization requires $1.36s$, compared to $1.70s$ for L-SNE [20]. This gap widens significantly in the 1.2M point ImageNet dataset, where our method completes a 4000 iteration minimization in $51.10s$, down from $346.94s$. On the 3M point Word2Vec dataset, FIt-SNE [11] performs a 4000 iteration minimization in $94.90s$, while our method requires $86.90s$. Convergence between the methods on the Word2Vec dataset is explained by attractive-force computations (Equation 11), which become exceedingly expensive for denser local neighbourhoods. For comparison: both existing methods perform relatively poorly on the smaller ImageNet dataset, where a higher perplexity value leads to larger neighbourhoods. To confirm this, we display runtimes of separate components in our method in Fig. 11. Evidently, attractive-force computation becomes a dominating factor in the minimization.

Observed scaling for 3D embeddings remains linear in all experiments, though there is a runtime overhead compared to 2D embeddings. This is expected, given the computational overhead involved in a third dimension. Linear complexity tSNE [20] is impractical for large datasets, as runtime spikes around 100K points, while our technique is orders of magnitude faster and completes a full 4000 iteration minimization on the 3M point Word2Vec dataset in $238.67s$.

While our technique improves runtime, embedding quality is another important metric. In the last two rows of Fig. 10, we examine KL-divergence of generated embeddings for increasingly large subsets of the datasets, in addition to computed NNP. The NNP metric is displayed in the form of precision/recall plots. For this, we repeat an experiment performed by Pezotti et al. [20]. For each point in a dataset, we observe points in an increasingly large neighbourhood of a size k based on perplexity. For every value from $k = 1$ to $k = 3\mu$, we compute T, defined as the accurate number of points belonging to both points' neighbourhoods. Precision is computed as T/k and recall is computed as $T/(3\mu)$. By averaging generated curves for each point, a representative curve is obtained for the entire dataset.

As demonstrated, our approximation has a minor impact on embedding quality compared to linear complexity tSNE [20]. The CUDA-SNE [2] implementation of FIt-SNE [11] interestingly delivers a lower quality of embeddings for the specified metrics. As explored by Linderman et al. [11], FIt-SNE reaches comparable levels of quality to BH-SNE [24], so these results are expected. The field-based approximation used by Pezotti et al. [20] is established to be more accurate, which is a quality our method mostly retains. We display embeddings generated with our method in Fig. 13. Further, we show example minimizations in a supplemental video.

7 Conclusion

We have presented a novel and improved minimization algorithm for t-SNE, providing significant performance improvements above the state-of-the-art, especially for large datasets and higher dimensional embeddings. The latter point is a crucial step forward, as it can improve embedding quality and could be of high relevance in many applications relying on a 3D visualization. For this reason, we have made an implementation of our method available (Sect. 5).

Our method illustrates that a field-based formulation of t-SNE, previously shown to have linear runtime, can still be significantly accelerated via a dual-hierarchy traversal. This allows us to compute N-body interactions efficiently, as is demonstrated in a GPGPU-based environment on modern graphics hardware. Our experiments reveal significant run-time improvements with regards to linear complexity t-SNE [20] and FIt-SNE [2] for two- and three-dimensional embeddings, while achieving comparable quality.

With these improvements, the t-SNE algorithm is, at this stage, dominated by the required KNN and attractive force computations, which are interesting challenges for future work.

References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986. doi: 10.1038/324446a0

[2] D. M. Chan, R. Rao, F. Huang, and J. F. Canny. T-SNE-CUDA: GPU-Accelerated T-SNE and its applications to modern data. In *30th International Symposium Computer Architecture and High Performance Computing*, pp. 330–338, 2018. doi: 10.1109/CAHPC.2018.8645912

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. doi: 10.1109/CVPR.2009.5206848

[4] C. Fu, Y. Zhang, D. Cai, and X. Ren. AtSNE: Efficient and robust visualization on GPU through hierarchical optimization. In *25th International Conference on Knowledge Discovery & Data Mining*, p. 176–186. Association for Computing Machinery, 2019. doi: 10.1145/3292500.3330834

[5] K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster HLBVH with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, p. 59–64. Association for Computing Machinery, 2011. doi: 10.1145/2018323.2018333

[6] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019. arXiv:1702.08734. doi: 10.1109/TBDATA.2019.2921572

[7] M. Kim, M. Choi, S. Lee, J. Tang, H. Park, and J. Choo. PixelSNE: Pixel-aligned stochastic neighbor embedding for efficient 2D visualization with screen-resolution precision. *Computer Graphics Forum*, 37:267–276, 2018. doi: 10.1111/cgf.13418

[8] D. Kobak and G. C. Linderman. UMAP does not preserve global structure any better than t-SNE when using the same initialization. *bioRxiv*, 2019. doi: 10.1101/2019.12.19.877522

[9] S. Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.

[10] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum*, 28:375 – 384, 2009. doi: 10.1111/j.1467-8659.2009.01377.x

[11] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger. Efficient algorithms for t-distributed stochastic neighborhood embedding. *CoRR*, abs/1712.09005, 2017.

[12] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. 2018. arXiv:1802.03426.

[13] B. Melit Devassy and S. George. Dimensionality reduction and visualisation of hyperspectral ink data using t-SNE. *Forensic Science International*, 311, 2020. doi: 10.1016/j.forsciint.2020.110194

[14] B. H. Meyer, A. T. R. Pozo, and W. M. N. Zola. Improving Barnes-Hut t-SNE scalability in GPU with efficient memory access strategies. In *2020 International Joint Conference on Neural Networks*, 2020. doi: 10.1109/IJCNN48605.2020.9206962

[15] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, vol. 2, p. 3111–3119, 2013. doi: 10.5555/2999792.2999959

[16] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1643–1652, 2014. doi: 10.1109/TVCG.2014.2346578

[17] NVIDIA. CUB: Cooperative primitives for CUDA C++, 2010. Accessed: 2021-06-24. [Online]. Available: https://github.com/NVIDIA/cub.

[18] N. Pezzotti, T. Höllt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. *Computer Graphics Forum*, 35(3):21–30, 2016. doi: 10.1111/cgf.12878

[19] N. Pezzotti, B. P. F. Lelieveldt, L. v. d. Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tSNE for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, 2017. doi: 10.1109/TVCG.2016.2570755

[20] N. Pezzotti, A. Mordvintsev, T. Höllt, B. P. F. Lelieveldt, E. Eisemann, and A. Vilanova. Linear tSNE optimization for the web. *CoRR*, abs/1805.10817, 2018.

[21] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014. doi: 10.1109/TVCG.2014.2346574

[22] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualization of large-scale and high-dimensional data. *CoRR*, abs/1602.00370, 2016.

[23] V. v. Unen, T. Höllt, N. Pezzotti, N. Li, M. Reinders, E. Eisemann, A. Vilanova, F. Koning, and B. P. Lelieveldt. Visual analysis of mass cytometry data by hierarchical stochastic neighbour embedding reveals rare cell types. *Nature Communications*, 8(1740), 2017. doi: 10.1038/s41467-017-01689-9

[24] L. van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

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

[26] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *Journal of Machine Learning Research*, 11:451–490, 2010.

[27] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[28] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 93. Society for Industrial and Applied Mathematics, 1993.