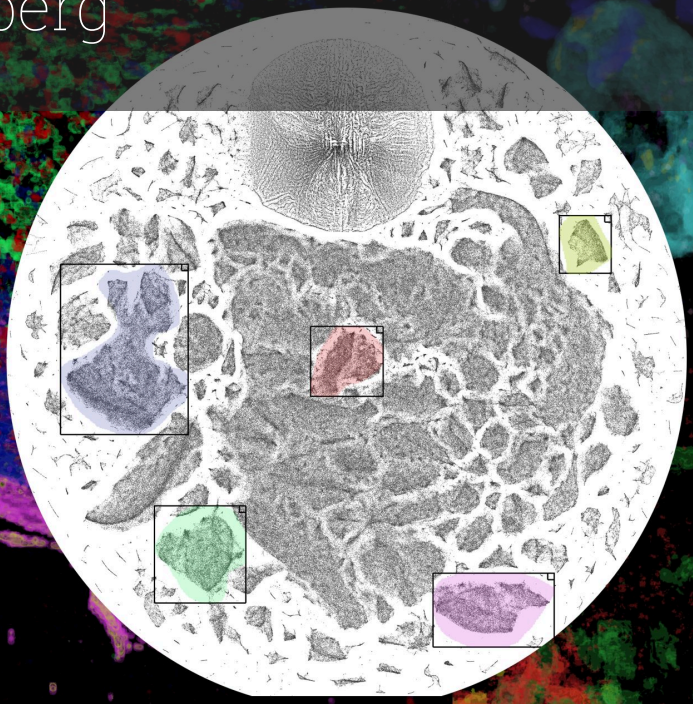
Neighborhood-Preserving Dimensionality Reduction for Multivariate Volume Rendering

Ravi Snellenberg



Delft University of Technology

TU Delft

Neighborhood-Preserving Dimensionality Reduction for Multivariate Volume Rendering

by

Ravi Snellenberg

In partial fulfilment of the requirements for the degree of **Master of Science**
in the Artificial Intelligence Technology track
at the Faculty of Electrical Engineering, Mathematics and Computer Science of
Delft University of Technology,
to be defended publicly on Friday September 12, 2025 at 9:30 AM.

Student number: 5315867
Project Duration: December 18, 2024 — September 12, 2025
Thesis committee: T. Höllt, TU Delft, thesis advisor, daily supervisor
C. Raman, TU Delft

Preface

At this point, I feel like writing this thesis has been quite a long journey, and thinking back to when it all started already feels like ages ago, though maybe I just have a very short-term memory. However, what I do remember is that I have gotten quite a bit of help along the way from various people.

The most significant of whom is my supervisor, Thomas Höllt, who has been meeting and discussing my thesis with me almost every single week over this entire journey. During those meetings, he has provided guidance and ideas, as well as referring me to useful resources such as datasets and other people, so a big thank you for that.

Speaking of those other people, thank you to the people in the ManiVault Discord who helped me on my way when I was struggling with it, as well as to Ali Samadzadeh, who provided me with a lot of helpful knowledge about ANN algorithms when I first started looking into using them for my methods.

I would also like to thank my friends and family who have had to suffer through listening to me talk about my thesis far too much. Especially my mother, who has been lamenting my nearly daily status reports, sorry for that. However, you all have been a great way to fan the flames of my enthusiasm while blowing away my frustrations.

And finally, I am grateful grammar checkers exist, since you would think that after all the writing I have been doing I would be able to spell properly, but unfortunately, it's the grammar checkers that have made my text, hopefully, somewhat readable.

Ravi Snellenberg
Delft, September 2025

Abstract

Multivariate volumetric datasets, which are volumetric datasets that contain more than one variable per voxel, are becoming increasingly prevalent, such as transcriptomic datasets, which can contain up to hundreds of dimensions. This high dimensionality has proven to be a difficult obstacle to overcome when trying to visualize such datasets, especially in creating user-friendly transfer functions, which are needed to guide the direct volume rendering pipeline into visualizing information relevant to the user. This work proposes the use of neighborhood-preserving dimensionality reduction, such as t-SNE and UMAP, to define a transfer function that simplifies highlighting highly similar points in data with an arbitrarily high number of dimensions. However, since direct interpolation between points in the resulting non-linear embedded space does not provide correct results, we propose and evaluate several interpolation methods to address this limitation. The most accurate of which is the use of approximate nearest neighbor algorithms to approximate the position an interpolated high-dimensional point would have in the embedded space. We also propose an adaptation of a two-step TF that can simulate some of the surface-based effects whose scalar-based implementations cannot be directly applied to multivariate data, such as gradient-based opacity and surface shading. Finally, we implement a plugin in the ManiVault framework, which is an existing framework made to analyze multivariate data, to test and analyze our method.

Contents

1

Introduction

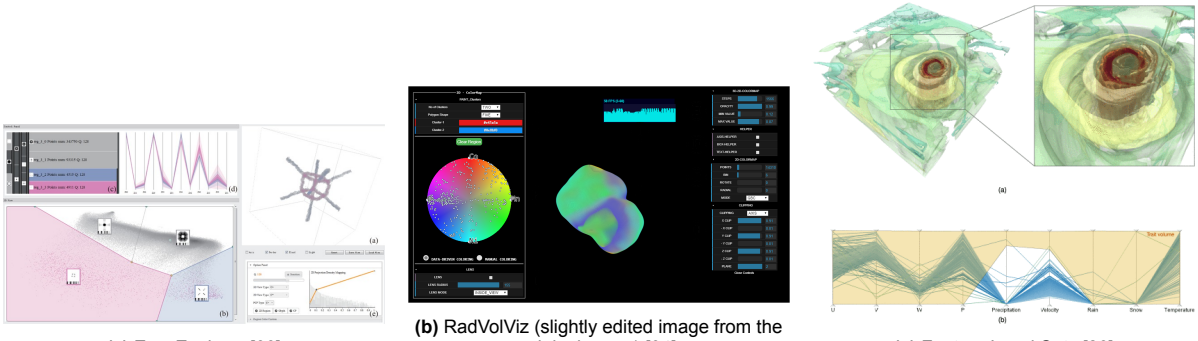
The number of multivariate volume datasets has increased over the past two decades. The primary reasons for this growth are the ever-expanding computational power of modern hardware, which enables complex physics simulations, and advances in the medical field, which have produced numerous transcriptomics datasets [23, 59]. As these datasets become more prevalent, visualizing them has become an increasingly important area of research, which has led to a wide range of visualization techniques. These techniques typically rely on direct volume rendering (DVR), the standard approach for visualizing volumetric data, but differ in the type of transfer function (TF) used in conjunction with DVR.

Unlike discrete color maps that associate individual values with distinct visual elements, TFs define a continuous mapping from the data domain to visual representations, typically encoded by attributes such as color and opacity. Most TFs used for multivariate data combine multiple techniques to extract meaningful insights. This is needed due to the complexity of said data, as the primary focus is not only on individual variables at specific spatial locations but also on the relationships and interactions between variables, which often reveal the most valuable insights.

Several types of TFs have been proposed to visualize multivariate volume datasets, including TFs based on parallel coordinate plots (PCPs) [22, 38, 66], correlation analysis graphs [6], guided transfer functions (where users highlight areas of interest) [69], iso-surface detection schemes [26, 55], linear dimensionality reduction methods [9, 30, 34], and many more novel approaches [23]. However, these methods all involve trade-offs. Some TFs are easy and quick to use, but limit what the user can visualize, such as iso-surface detection schemes, which can only visualize the computer-detected surfaces of user-selected attribute values (see Figure 1.1c). Others, such as PCPs, allow users to explore a broad range of variable relationships but are cumbersome and difficult to work with. Furthermore, the exceptionally high dimensionality of some datasets, often exceeding 100 dimensions, as seen in transcriptomics datasets, renders many of these visualization techniques impractical. Since their complexity scales with the number of dimensions, as can be seen in Figure 1.1, none of the existing software examples display more than 12 dimensions, since they are not built for datasets containing hundreds.

This high dimensionality not only makes TF design difficult, but also presents challenges for common DVR functionalities based around surfaces, such as surface shading (of smooth surfaces) and transparency modulation.

Non-linear dimensionality reduction (DR) techniques that aim to preserve neighborhoods, such as t-SNE [43] and UMAP [48], are some of the most popular visualization methods for high-dimensional multivariate datasets without a spatial component [2, 14, 16, 32]. These neighborhood-preserving DR methods require minimal user input, regardless of data dimensionality, while producing intuitive visual representations by clustering similar points near one another. This is particularly beneficial for multivariate datasets, where meaningful information is often derived from a combination of multiple variables. In contrast, linear DR techniques usually produce suboptimal visualizations that fail to highlight interesting properties of the data.



(a) EasyExplorer [66]

(b) RadVolViz (slightly edited image from the original paper) [34]

(c) Feature Level Sets [26]

Figure 1.1: Examples of existing multivariate volume visualization transfer functions.

Despite the success of these neighborhood-preserving DR techniques for high-dimensional multivariate datasets without a spatial component, their application to TF design in DVR remains limited. Since linear interpolation, the usual method for smoothing the underlying data in the visualization, is no longer easily achievable when integrating these DR techniques into the TF design. This thesis presents two approaches to solving this issue. These can be categorized as approximate linear interpolation sampling, and nearest neighbor (NN) sampling with surface smoothing.

The issue with using linear interpolation with neighborhood-preserving DR methods is that inserting a new point (in this case, the interpolated point) into the lower dimensional space typically requires recomputing the entire transformation, a process that is computationally expensive and usually precomputed. In this work, we approximate the position of an interpolated point in the lower-dimensional space without recomputing the full DR, leading to faster, though still not real-time, rendering.

In contrast, NN sampling does support real-time rendering (since it only samples from the points already in the precomputed DR) and may even produce a more faithful visualization of the sampled data. Therefore, NN sampling may be preferred, despite generally producing blocky visualizations. To mitigate this blockiness, we propose a simple surface smoothing technique that can be used with NN sampling to improve visual quality.

Finally, we propose an adaptation of a two-step material-based TF, originally introduced for scalar datasets in the work of Igouchkine et al. [25], for multivariate volume rendering. This method allows differentiation between surfaces and homogeneous areas in multivariate datasets with minimal computational overhead, making surface shading and transparency modulation around surfaces possible again.

The key contribution of this work is a system that can render high-dimensional volumetric datasets using a TF based on neighborhood-preserving DR. [1] To accomplish this, we:

- Designed and tested several interpolation schemes that can be applied to the new non-linear domain, the TF is defined in.
- Adapted an existing surface-based TF to the new domain, such that certain surface-based effects can be reproduced in this new domain.

[1] It has been implemented as a plugin for the ManiVault framework [65]. The implementation and source code are publicly available at: https://github.com/Rsnelllenberg/VolumeProjectorPlugin

2 Background

In this chapter, the basic concepts necessary to understand the context of this thesis are covered. Section 2.1 provides a brief overview of direct volume rendering (DVR), establishing the foundational knowledge required to understand our proposed methods and explaining why interpolation in the color domain results in incorrect visualizations. And Section 2.2 introduces various dimensionality reduction techniques, highlighting the advantages of neighborhood-preserving methods for our application and discussing the challenges they pose for interpolation.

2.1. Direct Volume Rendering

Direct Volume Rendering (DVR) is a widely used technique for visualizing volumetric data without requiring explicit surface extraction. Unlike traditional surface-based rendering, which relies on geometric representations like meshes, DVR operates directly on volume data.

2.1.1. Data Representation and Sampling Strategies

Volumetric datasets aim to represent data that has a naturally three-dimensional, spatial structure, such as fluid flow fields, atmospheric measurements, or medical imaging data like MRI scans. These phenomena extend continuously throughout space, and their values can vary smoothly at every point within a volume.

The measured phenomenon is discretized into a 3D grid of scalar values, commonly known as voxels [29]. Each voxel stores a measured value (more than one in the case of multivariate datasets) within a small cubic region, representing the local intensity of a variable, such as air pressure in a simulation or signal strength in an imaging scan. This discretization is necessary due to the finite resolution of measuring instruments, as well as the impracticality of storing infinitely dense data.

Reconstructing the continuous volumetric signal from these voxel samples, such as the measured temperature in a volume (which changes gradually rather than jumping at discrete intervals), can be approached as a signal reconstruction problem [47]. Theoretically, if the sampling rate of the volume exceeds the Nyquist frequency of the underlying data, the original signal can be perfectly reconstructed using the ideal sinc filter [50]:

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}. \tag{2.1}$$

However, the sinc filter requires an infinite number of samples and calculations, making it impractical for real-time applications. Furthermore, in practice, the sampling rate of volume data often does not exceed the Nyquist frequency, meaning even a theoretically perfect reconstruction would still fail to return a perfect representation of the measured phenomenon.

Instead, the most common approach is linear interpolation, which assumes a linear transition between neighboring voxel values and thus requires at most 8 samples. Furthermore, volume data is stored as 3D textures on modern GPUs, enabling efficient hardware-accelerated trilinear interpolation while sampling [33].

Figure 2.1: Comparison of sampling and rendering strategies on a carp volume dataset. Left to right: isosurface with nearest neighbor sampling, isosurface with linear interpolation, composite rendering with linear interpolation.

When interpolation between samples is not applied and values are instead read directly from the voxels, the process is referred to as nearest neighbor (NN) sampling. Using it leads to the characteristic blocky appearance due to the lack of smoothing. Despite this, NN sampling is still sometimes preferred, since it preserves the original discrete values of the dataset, avoiding the introduction of artificial intermediate values that linear interpolation may produce, which might be wrong, especially near high-frequency areas of the real underlying data. For instance, in an CT dataset, at the boundary between lung tissue (which has a Hounsfield unit (HU) value of around -650) and soft tissue (a HU value of around 200), linear interpolation could yield a value corresponding to an entirely different tissue type, such as liver (which has around HU 60), misrepresenting the data. By contrast, NN sampling ensures that only actual, measured values are visualized, making it the preferred option in some cases where avoiding the misclassification of any samples is more important than visual smoothness or the theoretical best reconstruction of the data. See Figure 2.1 for a visual comparison between the two sampling strategies on a single variable volume dataset.

2.1.2. Transfer Functions

When sampling data from a volume, the visualization program requires a way to map those values to the visual properties used in rendering. The most common approach is to use a transfer function (TF), which maps each sampled value to a corresponding color and opacity, as shown in Equation 2.2. S represents the sampled value, and c and α represent the resulting color and opacity.

$$\mathsf{TF}(S) \rightarrow \big(c, \alpha\big) \tag{2.2}$$

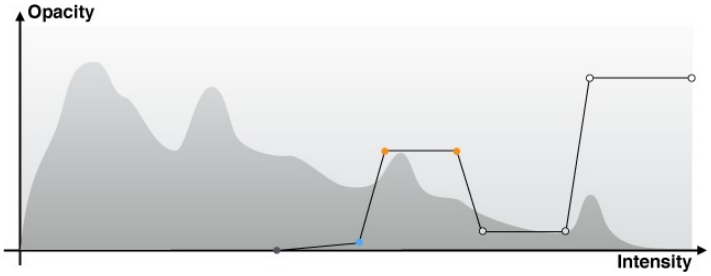
Some TFs can also require additional inputs to further guide the output, such as the gradient magnitude at the sampled position, which leads to the form shown in Equation 2.3. This extra information can, for example, be used to enhance surfaces by modifying opacity around samples with high gradient magnitudes, e.g., surfaces.

$$\mathsf{TF}(S, \|\nabla S\|) \rightarrow \big(c, \alpha\big) \tag{2.3}$$

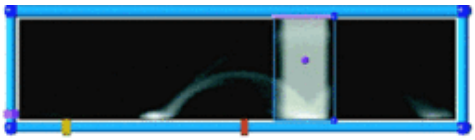
Transfer functions are typically user-created or guided and thus require a user interface, as illustrated in Figures 2.2a and 2.2b. Since no single TF can highlight all important features of a dataset without producing visual clutter. It is therefore often up to the users themselves to modify them based on the dataset and their specific goals. Some rendering techniques, however, require less or no user input, either because they visualize a simple principle (such defining a ray only by their maximum value an map it to greyscale values where white is defined as the maximum value in the data) or because the system automatically identifies certain important features.

Two-Step TF

One TF variation that is of particular importance to us is the two-step TF proposed by Igouchkine et al. [25], which was originally developed for assigning realistic materials to scalar datasets. It requires two samples as input: specifically, the value of the previous step (or, equivalently, a sample representing the start of the ray step), in addition to the current sample. In this method, the first step of the two-step



(a) A 1D transfer function mapping intensity (x-axis) to color and opacity using user-defined control points [39].



(b) A 2D transfer function interface using intensity (x-axis) and gradient magnitude (y-axis); users assign color and opacity by selecting regions [31].

Figure 2.2: Examples of transfer function user interfaces

TF, denoted T_{mat}, converts a data sample $\vec{\rho}$ into a discrete material ID m:

$$T_{mat} : \vec{\rho} \rightarrow m, \tag{2.4}$$

while the second step, T_{opt}, maps the pair of material IDs from the start and end of the ray step, (m_s, m_e), to an optical property (i.e., the color and opacity) $\vec{L}$:

$$T_{opt} : (m_s, m_e) \rightarrow \vec{L}. \tag{2.5}$$

This second step has discrete values as inputs, which means it also has a discrete output (this was important for assigning realistic materials). But it also allows for easy distinction of homogeneous and surface areas since two of the same material IDs as input means it is a homogeneous area, and vice versa.

2.1.3. Raycasting Methods

Direct Volume Rendering (DVR) is usually accomplished by sampling data from a volume using a technique called ray casting [35], where rays are traced from each pixel in the virtual camera through the volume at angles determined by the camera's perspective. The volume is sampled at fixed intervals along the ray (see Figure 2.3), and each sample is then processed using a specified technique. The volume rendering method most relevant to this thesis is compositing.

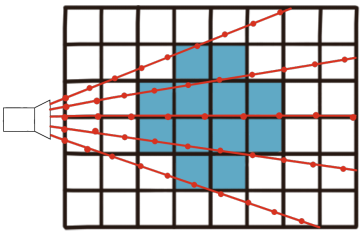


Figure 2.3: DVR raycasting step: the red lines indicate rays shot from each pixel, and the red dots are samples that are taken

Compositing

Compositing treats each sample as an entity that both emits and absorbs light. The amount of emission and absorption is determined by the sample's opacity, with higher opacity values resulting in stronger effects.

At each step of the ray[1], the output of the transfer function is used in Equation 2.6, where $C_{\text{acc}}(i)$ and $\alpha_{\text{acc}}(i)$ are the accumulated color and opacity after i steps:

$$\begin{aligned} C_{\text{acc}}(i) &= C_{\text{acc}}(i-1) + (1 - \alpha_{\text{acc}}(i-1))\, \alpha_i c_i \\ \alpha_{\text{acc}}(i) &= \alpha_{\text{acc}}(i-1) + (1 - \alpha_{\text{acc}}(i-1))\, \alpha_i \end{aligned} \tag{2.6}$$

Compared to other volume rendering techniques (such as Maximum Intensity Projection and Isosurface Rendering), this approach provides more flexibility in highlighting and coloring different samples. It also incorporates all samples along the ray, rather than selecting only the most significant one based on a predefined rule. As a result, it can render not only surfaces, but also cloudlike structures that lack a well-defined boundary, an effect illustrated in Figure 2.1. However, this expressiveness comes at the cost of increased complexity, as designing effective TFs for it often requires significant user input.

[1] We assume a stepsize of length 1, otherwise we need to replace α_i with $\alpha'_i = 1 - (1 - \alpha_i)^{\Delta stepsize}$

2.1.4. Pre-Classification vs Post-Classification

Suppose you want to pre-compute the conversion of all the samples in the volume to colors using the TF, instead of recalculating it for every sample you take. In that case, you would obtain a volume that stores color values, and the program would only need to interpolate the resulting colors (this is called pre-classification), rather than interpolating the measured data and applying the TF to the interpolated results every time the volume is sampled (post-classification). However, using post-classification instead of pre-classification is crucial for obtaining clear results without artifacts around the edges [20, 57], as shown in Figure 2.4.

A simple example can illustrate the cause of this issue: consider a TF where values 0 and 1 are mapped to white, while all intermediate values are mapped to a completely transparent color. If two adjacent voxels contain values of 0 and 1, interpolating in the color domain would incorrectly yield a solid white color, whereas interpolating the original scalar values would correctly return a transparent result. This idea is illustrated using a more realistic and complex example in Figure 2.5, which shows that pre-classification loses the high-frequency changes in the TF.

Thus, interpolation should always be performed on the raw scalar values before applying the transfer function to ensure an accurate representation of the underlying structures.

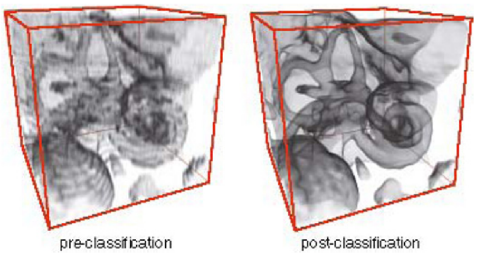


Figure 2.4: This figure shows the blurriness and artifacts around the surface that the use of pre-classification has on the visualization [57].

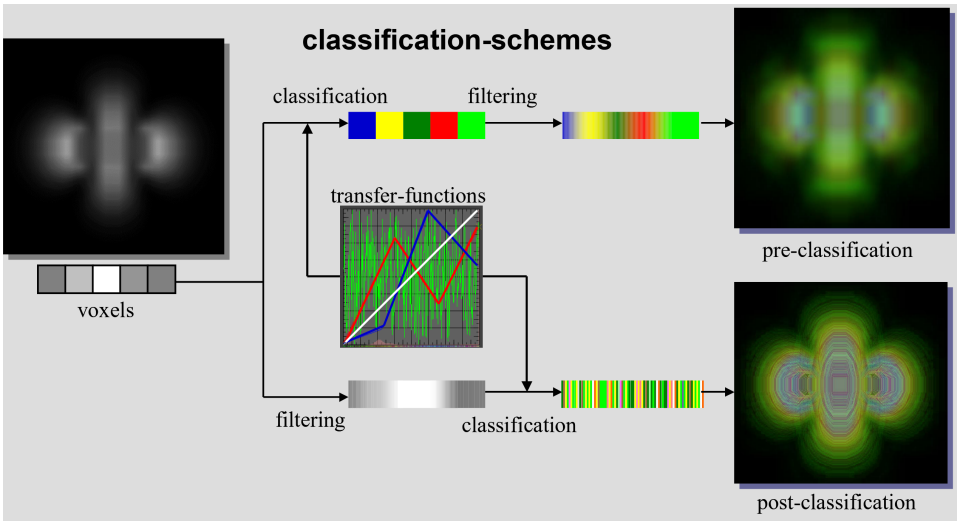


Figure 2.5: Compares the use of pre-classification against post-classification with a TF with a high frequency green channel. In the resulting images, it can be seen that the high-frequency color changes of the TF are only preserved when using post-classification [20].

2.2. Dimensionality Reduction Techniques

Dimensionality reduction (DR) techniques aim to map a high-dimensional space into a lower-dimensional space while optimizing for a specific metric. Each DR technique seeks to preserve certain properties of the original data, and an algorithm's approach depends on the metric it prioritizes. DR techniques can be broadly categorized into linear and non-linear methods.

Linear DR methods, such as Principal Component Analysis (PCA) [28] and Multi-Dimensional Scaling (MDS) [7], find the linear transformation that best optimizes the point distribution for their metric while casting the data to a specified lower-dimensional space. It then applies this linear transformation to each data point in the dataset. And since all points use the same linear transformation, linearly interpolating between points in the original space and then applying the linear transformation is equivalent to first projecting the points and then interpolating. In mathematical terms, for any two points x and y, if P denotes the projection, then the following equation holds:

$$[P\big[(1-\lambda)x + \lambda y\big] = (1-\lambda)(Px) + \lambda(Py),] \tag{2.7}$$

making the interpolation consistent across both approaches. This makes it highly usable in applications like DVR, where interpolation between values plays a large role, and it has already been adopted in proposed visualization tools [30].

In contrast, non-linear DR methods or more specifically the ones preserving neighborhoods, such as t-Distributed Stochastic Neighbor Embedding (t-SNE) [43], Uniform Manifold Approximation and Projection (UMAP) [48], and Locally Affine Multidimensional Projection (LAMP) [27], rely on iterative optimization processes where each data point is adjusted relative to its close environment. The resulting lower-dimensional embedding tries to preserve local neighborhood structure, placing points that are close in the high-dimensional space, also close together in the low-dimensional space.

Thus, unlike linear methods, no single closed-form transformation applies uniformly to all points, and a newly added point unequally influences the positions of others within the reduced space. As a result, updating the dataset typically requires recomputing the entire DR process. Despite this limitation, neighborhood-preserving DR methods are widely used because they can better preserve complex structures in the data, making them a good option for visualizing high-dimensional relationships [2, 14, 16, 32]. An example of this can be seen in Figure 2.6 that displays the DR of a multivariate mouse brain transcriptomics dataset [67] using both a popular linear (PCA) and neighborhood-preserving (t-SNE) DR algorithm.

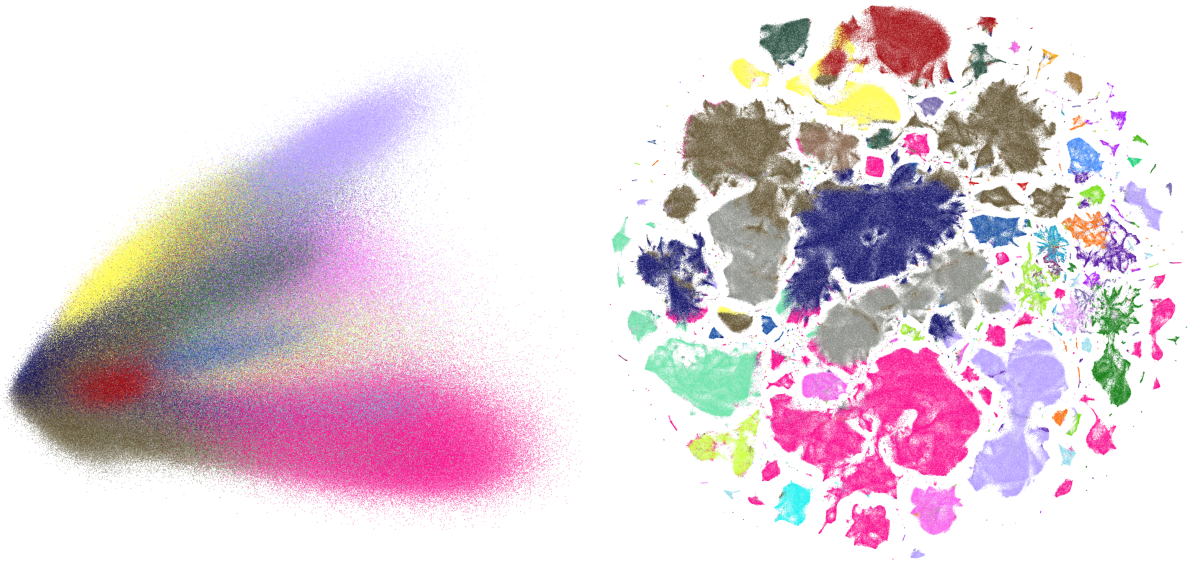


Figure 2.6: Dimensionality reduction of a mouse brain transcriptomics dataset. Left: result of a linear DR (PCA); Right: result of a neighborhood-preserving DR (t-SNE). The colors indicate cell class labels corresponding to known cellular functionalities.

3

Related Work

In this chapter, prior work relevant to the contributions of this thesis is discussed. Section 3.1 reviews existing transfer function (TF) approaches for visualizing multivariate volume datasets, highlighting their limitations and the need for further innovation. Section 3.2 examines the challenges of applying surface rendering techniques to multivariate volume visualization. Section 3.3 discusses existing methods for surface extraction, as several techniques from this field can be adapted to improve the visual quality of nearest-neighbor (NN) sampling in DVR shaders. Finally, Section 3.4 explores research related to estimating the placement of new points in an existing neighborhood-preserving dimensionality reduction (DR) mapping, a key challenge when integrating DR techniques into TF design.

3.1. Transfer Functions for Multivariate Volume Visualization

Several approaches have been proposed to visualize multivariate volume data, most of which have already been thoroughly reviewed in the literature survey by He et al. [23]. In this section, we focus on the most relevant techniques found in research concerning TFs and their application to multivariate volume visualization, discussing their relation to this thesis.

Generally, techniques for handling the complexity of multivariate data can be categorized into the four main strategies discussed below. While these distinctions are not always clear-cut, as many solutions incorporate aspects of multiple categories, this classification helps to understand the strengths and limitations of different approaches.

3.1.1. Multi-dimensional Transfer Functions

These methods attempt to create new UI widgets specifically designed to handle multivariate data. Some extend 1D TFs to multiple dimensions, such as parallel coordinate plots [22, 38], while others introduce entirely new designs like the radial-based TF in RadVolViz [34]. Another approach is to use multiple widgets in conjunction to simplify interactions, as seen in EasyExplorer [66] and the work by Kewei Lu et al. [41]. These methods can also incorporate iterative user-guided refinement, where user feedback indicates a region of interest [69].

However, as the number of variables increases, interaction complexity and usability become major challenges. Many of these methods scale poorly with high-dimensional data, making them impractical for datasets with many variables. However, they provide users with a high degree of control over what is visualized. Figures 1.1a and 1.1b and the PCP of in Figure 1.1c fall mainly under this category.

3.1.2. Fusion Visualization Methods

This approach treats multivariate data as a collection of separate scalar datasets, generating individual visualizations for each variable and then combining them. Examples include techniques that blend multiple DVR renderings [17] as seen in Figure 3.1 or those that assign unique visualizations to each variable [61]. A specialized variant of this approach is the use of glyphs [8], which encode multiple variables within a single visual icon.

These methods struggle with scalability, as combining multiple visualizations can quickly clutter the screen. Additionally, they do not effectively capture interactions between variables, as each variable is treated independently.

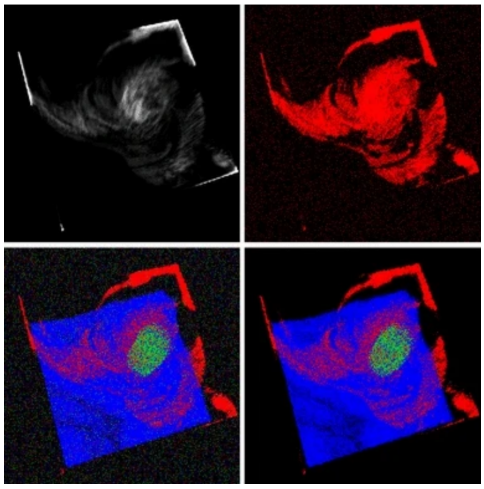


Figure 3.1: Each variable is rendered separately and then combined into a single visualization using a form of dithering [17].

3.1.3. Dimensionality Reduction Methods

These methods simplify visualization by reducing the number of variables. Some approaches create custom methods for this, such as generating new scalar fields that condense multivariate information [19] (see Figure 3.3). Others apply existing linear dimensionality reduction techniques, such as a GBC plot [11] in RadVolViz [34] or PCA [28] in the work by Kim et al. [30]. However, as discussed in Chapter 2, linear DR usually fails to represent very high-dimensional data in a visually usable manner.

There does exist quite a bit of work that incorporates non-linear DR techniques, such as LAMP [27] in EasyXplorer [66], UMAP [48] in VolumePuzzle [1], MDS [7] in the work of H. Guo et al. [22], or Isomap and Locally Linear Embedding in the work by Kim et al. [30]. However, apart from the work by Kim et al., these methods typically use DR as a visual guide rather than as a direct component of the TF, or they apply pre-classification (see Section 2.1.4, which discusses its limitations). And the approach by Kim et al. features a built-in DR that is not well-suited for guiding multivariate visualization, a slow approximation pipeline that uses exact nearest-neighbor (NN) algorithms with no provided information on how well this approximation works, the dataset they use natively only has 3 dimensions per voxel (they simply add extra gradient information to it, as seen in Figure 3.2) and a plethora of other points of improvement. Thus, while it does enable multivariate visualization using neighborhood-preserving DR, several open questions and limitations remain.

Dimensionality reduction helps manage complexity by reducing the number of variables the TF has to process. It can also highlight relationships between variables, as shown in Figure 3.3. However, compressing data into lower dimensions may distort or obscure important relationships within the original dataset, resulting in misleading or difficult-to-interpret visualizations. This is evident in the PCA embedding in Figure 2.6.

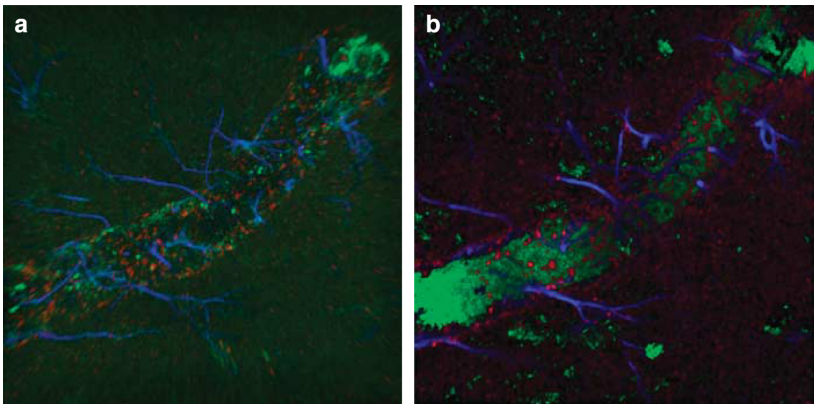


Figure 3.2: Visualization of a 3D dataset using MIP: (a) with scalar values mapped to RGB channels; (b) with added gradient magnitudes, making a 6D dataset and neighborhood-preserving DR which defines a 3D TF [30].

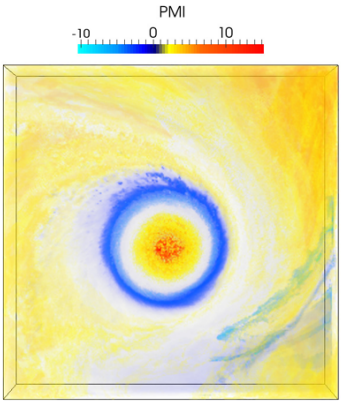


Figure 3.3: Visualization of the Hurricane Isabelle dataset, showing the relationship between two variables as a single PointWise Mutual Information (PMI) variable [19].

3.1.4. Automated Visualization Methods

These methods automate visualization generation, reducing user effort by allowing the system to determine how data should be visualized. Some approaches construct graphs of suggested visualizations based on predefined metrics, such as clustering in MultiClusterTree [63] or isosurface analysis in FeatureNet [55]. Others, like GuideMe [69], allow users to select regions of interest, which are then processed to extract meaningful surfaces. Domain-specific automated visualization tools also exist, such as SpaceWalker [37], which is tailored for transcriptomics datasets by prioritizing genome gradients.

While these methods reduce user workload, they often sacrifice fine-grained control, making customization more difficult. Some tools are also highly specialized, limiting their applicability beyond their intended dataset types.

3.2. Rendering Surface-Based Effects

In scalar volume data, surface-based effects, such as highlighting or shading, are typically derived from the local gradient. Techniques can use gradient magnitude as an axis in TF interfaces (as seen in Figure 2.2b) [31, 58] or as an opacity modifier [36]. Other methods rely on the gradient direction for shading approaches like Phong shading, which uses local gradients to apply lighting across the volume, or for silhouette enhancement [58], which can highlight fine structures in the data while keeping less interesting regions transparent to reveal features behind them, as shown in Figure 3.4.

In multivariate datasets, however, such methods are not directly applicable as they lack usable gradient data, and no proposed alternatives have achieved comparable results. Instead, shading in multivariate visualization has primarily been incorporated in two ways. The first involves generating a tag volume derived from the multivariate data and applying directional occlusion shading [60], as demonstrated in GuideMe [69] or surface shading on NN sampled fully opaque surfaces in VolumePuzzle [1]. The second approach is fusion-based techniques where shading is computed independently for each scalar variable [55], allowing the use of standard scalar-based shading methods.

While these approaches support surface-based effects to some extent, they do not offer integrated surface rendering that supports both surfaces and homogeneous regions.

Figure 3.4: An orange rendered in a DVR with and without boundary/silhouette enhancement using gradients [58].

3.3. Surface Smoothing for Voxel Volumes

Most direct volume rendering applications prefer using linear interpolation to smooth the visualization and avoid blocky artifacts. However, as discussed in Section 2.1.1, NN sampling is sometimes preferred to avoid incorrect classification of sampled data. In this section, we discuss several algorithms that could help reduce the blockiness caused by NN sampling, addressing its primary disadvantage relative to linear interpolation.

Surface extraction is a well-established research area, and many potentially useful algorithms have been proposed over the years. While these algorithms are typically designed to extract a surface mesh from a volume, rather than detect surfaces along a ray, many are inherently parallelizable and could possibly be adapted to run on localized regions. This opens up the possibility of adapting them to operate along ray paths in DVR. Alternatively, a surface could be extracted in a pre-rendering step, with an auxiliary data structure constructed to enable surface-aware rendering during ray traversal. These possibilities make surface extraction research relevant to our goals.

3.3.1. Marching Cubes

The most well-known surface extraction algorithm is Marching Cubes (MC) [40]. Originally designed to extract isosurfaces in scalar datasets, numerous MC variants have since been proposed to improve mesh quality or performance [12, 15, 49]. Of particular interest to our use case are the variants that support more than two discrete material classes [24, 56].

The generalized MC algorithm extends MC to handle multiple materials by dividing the cube into subregions and estimating the material for each. This process can be significantly sped up by building generalized lookup tables using the results from this generalized approach (which is an essential step for any MC variant), but its performance suffers when more than three neighboring material classes are present due to memory constraints on GPU-resident tables. The multi-material MC variant [56] addresses multiple materials differently: it treats transitions between different materials as the original cubic shape of the voxel grid and relies on a subsequent smoothing algorithm to remove artifacts from this approximation.

3.3.2. Dual Contouring Methods

Despite many optimizations, MC has a few fundamental limitations. To address these limitations, dual contouring methods were introduced. These methods define a single vertex per cell, located anywhere inside the voxel, avoiding redundancy and enabling smoother results.

The most relevant dual contouring variant, called SurfaceNets [10], does not require gradient data and determines vertex positions based on the positions of neighboring connected surface cells. Over time, several SurfaceNets variants have been developed specifically for multi-material datasets [21, 62].

3.4. Placing New Datapoints into a Known Embedding

As discussed in Section 2.2, interpolation would require inserting a new point into an embedding of a neighborhood-based DR method, which would theoretically require recalculating the entire reduction, something that is computationally infeasible for real-time applications. In this section, we review the few methods that have been proposed to address this challenge.

A parametric version of t-SNE (or any other neighborhood-based DR method) that uses a feed-forward neural network to approximate the t-SNE distribution would allow new points to be inserted after training without further modification [13]. However, these approaches have seen limited adoption, as the resulting visualizations are often considered inferior to those produced by iterative neighborhood implementations.

Previous literature has also proposed an alternative method for inserting points into an existing distribution [32, 53]. This approach uses the 10 nearest neighbors (already present in the embedding) of the new point and takes the median or weighted mean of their positions as an initial estimate. However, the work by Kobak et al. [32] only mentioned this as a possible solution and did not conduct any in-depth evaluation. In contrast, the work by Poličar et al. [53] found that using nearest neighbors alone was insufficient and therefore proposed a follow-up step involving 100 iterations of optimization. That said, their focus was on merging two separate datasets with differing internal distributions into a single t-SNE visualization, and the challenges in that context may not transfer directly to others.

Neither paper referenced the earlier work by Kim et al. [30], which already used the same method of using NN in its TF. However, that paper only briefly mentioned the technique, offered little analysis of its effectiveness, and, as discussed in Section 3.1.3, left several areas open for improvement.

4

Method

In this chapter, we discuss the design and implementation of the proposed visualization software. In Section 4.1, we will discuss how incorporating neighborhood-preserving dimensionality reduction (DR) simplifies the transfer function (TF) user interface, as well as highlight some of the limitations current DR algorithms have, and how these can impact our visualization. In Section 4.2, we will also introduce an alternative TF, which is a modification of the 2-step TF that was explained in Section 2.1.2. This TF is capable of simulating some of the surface-based effects that were no longer possible for multivariate data.

After this section, we will introduce several ways of sampling the volume, each with a different trade-off, since the non-linear nature of the space the TF is now defined in means we cannot simply use linear interpolation anymore. Thus, Section 4.3 introduces the most straightforward sampling methods, which we will use as a baseline to compare against. Section 4.4 attempts to approximate what position a interpolated high-dimensional should have in the embedded space (which is the 2D space). And Section 4.5 uses nearest neighbor sampling, which sidesteps the issue as it is not affected by the non-linear nature of the TF domain, and instead tries to resolve some of the blockiness that you get when using NN sampling.

Finally, to close things off, in Section 4.6 we will go over some relevant implementation details.

4.1. Impact of Dimensionality Reduction on Transfer Function

In Section 2.1, we have already gone over the basics of how direct volume rendering (DVR) works. Namely, we shoot rays from the camera through the volume, and we take samples from the volume along these rays. These samples are passed to a TF, which converts this data into something the user can interpret, such as color. These TFs usually have a user interface to give the user control over which parts of the volume they actually want to highlight. However, as was discussed in Section 3.1, making an understandable user interface for high-dimensional data is a difficult problem. However, we aim to solve this by first reducing the high-dimensional data down into a 2D space using neighbourhood-preserving DR.

In this section, we will discuss our TF design that makes use of this 2D data, and we will also highlight the issue that making use of a DR algorithm limits the size of the dataset we can work with without affecting the TF quality.

4.1.1. Transfer Function Widget

The 2D data the TF now receives as input is relatively simple, as neighbourhood-preserving DR already informs the user of potentially interesting areas, such as gene expression clusters, which are represented as clusters of points that are close together in the embedded space and thus are easily selected. Furthermore, neither axis has any inherent meaning, and they are equally important.

This means that our new TF UI design only needs to implement some basic functionality, as seen in

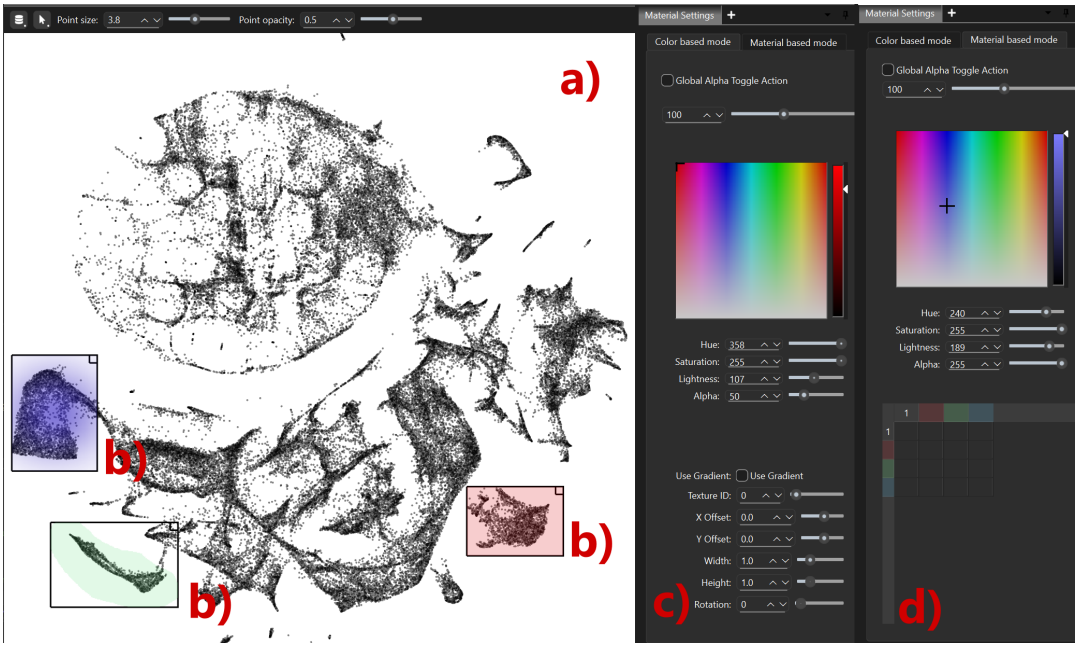


Figure 4.2: The transfer function widget interface. a) t-SNE embedding, b) TF widgets, c) color picker and alpha gradient editor, and d) two-step TF panel.

Figure 4.2. First, it needs to be able to display the data distribution to the user in a way that makes it clear which areas they may want to select (Figure 4.2a). Second, the user can select the desired areas through either a drag-and-drop square selection for simple, quick selections, or a lasso tool for less isolated clusters that require more careful tracing. To add an element of interactivity to these selections, they should be draggable, deletable, and resizable on the fly (Figure 4.2b). Once these selections are made, the user should be able to allocate which colors the points in a selection will be converted to. This is accomplished by selecting an area selection and using a color picker tool, like the one in Figure 4.2c, to assign a color to that selection. With that, we have all the functionality needed for the user to easily create an informed TF for the data.

We also made an extra gradient opacity modification tool in our application, as it is a common feature in existing 2D TFs. Its use, however, is relatively limited, as the inter-cluster distribution of points in the 2D space often does not have any significant gradients in the data that make sense to use for opacity modulation. Finally, since we also make use of the two-step TF discussed in Section 2.1.2, as will be discussed again later in the thesis, we also built a material combination matrix UI element for our application, based on the UI proposed by Igouchkine et al. in the original two-step TF paper [25], as can be seen in Figure 4.2d.

To pass all this information to other widgets, fixed-size textures can be used (or something equivalent, depending on the implementation).

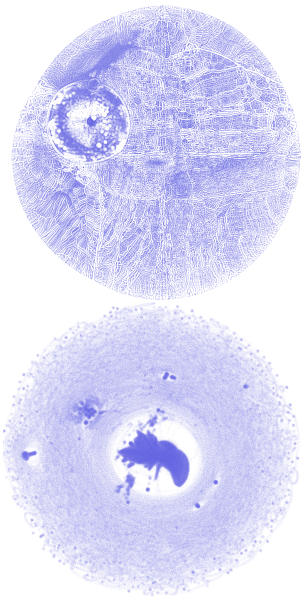
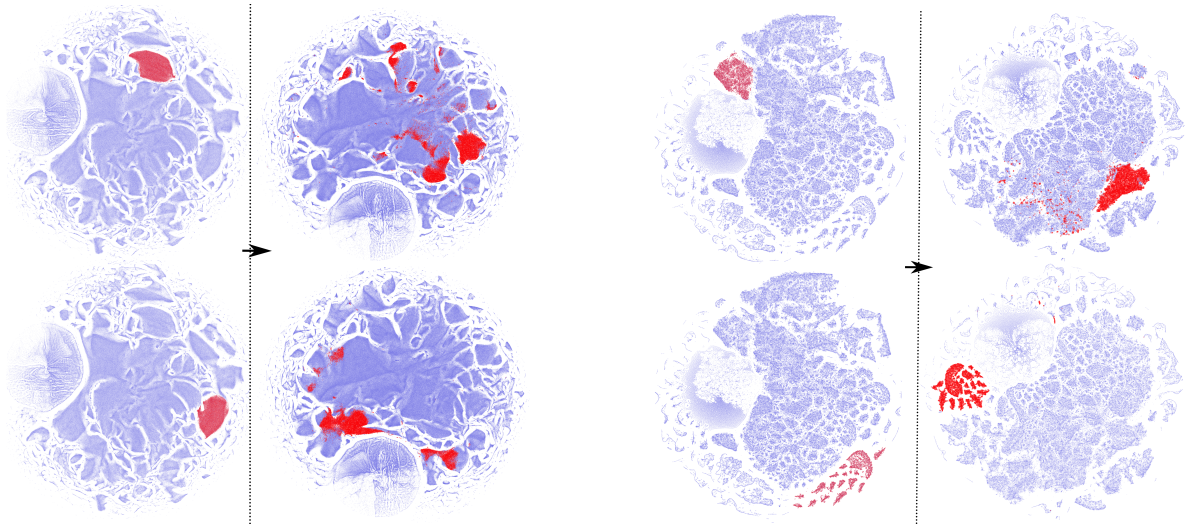


Figure 4.1: Results of running t-SNE (top) and UMAP (bottom) on an 8-million-point dataset over the course of a day and thousands of iterations

4.1.2. Dataset Size Limits

The inclusion of neighborhood-preserving DR in our TF also places limits on the size of the dataset the application can handle. Neighborhood-preserving DR algorithms are very expensive and do not scale well to datasets larger than several million points, and as shown in Figure 4.1, they may even stop converging once the dataset reaches such a size.

Even before a dataset reaches this limit, the neighborhood-preserving DR can start to have problems clustering all the points [32]. An example of this can be seen in Figure 4.3, which shows two DRs of the tissue dataset.



(a) DR of a 6 million point slice of the tissue dataset **(b) DR of a 625 thousand point slice of the tissue dataset**

Figure 4.3: The left and right sides of the arrow represent two different instances of running the DR on the same dataset. The red-highlighted points in each DR show how the same cluster is distributed

Figure 4.3a contains approximately 6 million data points, and Figure 4.3b approximately 625 thousand data points. They show how the points that form a single cluster in one instance of the DR are distributed when running the exact same algorithm again, but with different initialization points.

As shown, especially in the large dataset, points that are considered a single cluster are often split into multiple clusters or spread throughout the 2D space. This causes the misclassification of these essentially random points, which in turn results in visual clutter. For the smaller dataset, the number of misplaced points is drastically reduced, meaning that visualizing smaller volumes also leads to a better representation of the data with less clutter.

Another issue that arises with large datasets is that clusters become less defined, which makes it harder to identify and mark them in the TF. This further increases visual clutter and can cause misinterpretation, where points in distinct clusters are perceived as the same.

Since volumetric datasets often contain many voxels, these limitations can make it impossible to work with the entire dataset at once. In such cases, if the user wants an overview of the entire data, they must lower the data resolution or create a new dataset from a subset of the volume if they want to inspect a smaller region at higher resolution, as shown in Figure 4.4.

Some of these issues might be resolvable with the use of hierarchical DR algorithms like the HSNE [51] DR algorithm. However, resolving these issues lies outside the scope of our project, as they stem from the DR algorithm and not our proposed visualization framework.

4.2. Two-step Transfer Function

The basic TF type, which directly converts one sample input to an output color, cannot produce surface-based effects in a compositing pipeline for multivariate data due to the lack of usable gradient information, as discussed in Section 3.2. To address this issue, we introduce an adaptation of the two-step TF [25] from Section 2.1.2, which we can use to simulate some of these effects within our visualization framework.

In this section, we explain the adjustments required to use this two-step TF in a multivariate setting, as well as how it can be applied to create surface-based effects such as highlighting surfaces or adding shading.

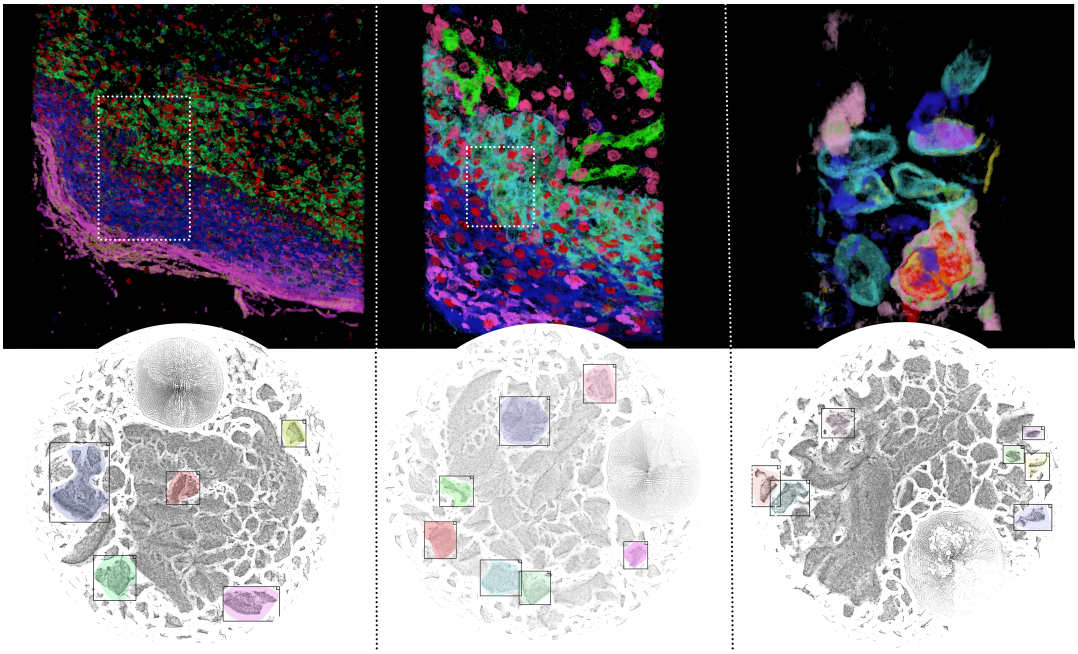


Figure 4.4: Visualizations of a slice of the tissue dataset, each containing 1.6 million points, but at different resolutions. The white dotted lines roughly display the area that the image right of it visualizes

4.2.1. Adaptation to a Multivariate Setting

The specifics of how we use the samples collected using the DVR pipeline to determine color differ slightly from what was proposed in the original two-step TF paper [25]. Since implementing the original algorithm directly results in a large amount of visual clutter. Additionally, the transition from a TF defined in a 1D space to a 2D space requires adjustments to how the algorithm identifies surface positions following material changes.

Artifact and Clutter Reduction

There are two main problems causing visual clutter. First, the visual clutter caused by points that are misplaced in the TF due to the problems discussed in Section 4.1.2, which can be seen in in Figure 4.5a. Second, assuming a linear distribution of values between voxels may return material values that are not actually present at that location in the dataset (hereafter referred to as artifacts). An example of this in scalar data was discussed at the end of Section 2.1.1. These artifacts are usually quite rare, however as can be seen in in Figure 4.5b that is not the case in our method. They are noticeable in our case since the DR algorithms misplace points when handling large datasets, creating more surfaces where these artifacts can occur. And when interpolating in the 2D space directly, we interpolate in a non-linear space, which makes these artifacts more likely to happen (as will be discussed in more detail in Section 4.3).

The two-step TF emphasizes these issues, as clutter and artifacts each create additional surfaces, and since surfaces are typically rendered with high opacity to highlight structure when using the two-step TF, this quickly leads to visual occlusion. In contrast, the basic TF weighs all samples equally, so clutter has less visual impact.

Misplacement of points is inherent to the DR algorithm, and fixing it is outside the scope of this thesis. Instead, we address its consequence: excessive surfaces that obscure important features. We thus need a way to classify problematic surfaces and remove them. We define isolated voxels as clutter, meaning those that do not share their material with surrounding samples, since these introduce both an entering and an exiting surface along the ray while adding little useful information.

To identify artifacts, we rely on a series of observations. First, artifacts tend to occur between voxels with significantly different values and typically return a material that does not match nearby samples. Second, in most datasets, similar values naturally form spatial clusters rather than appearing in isolation.

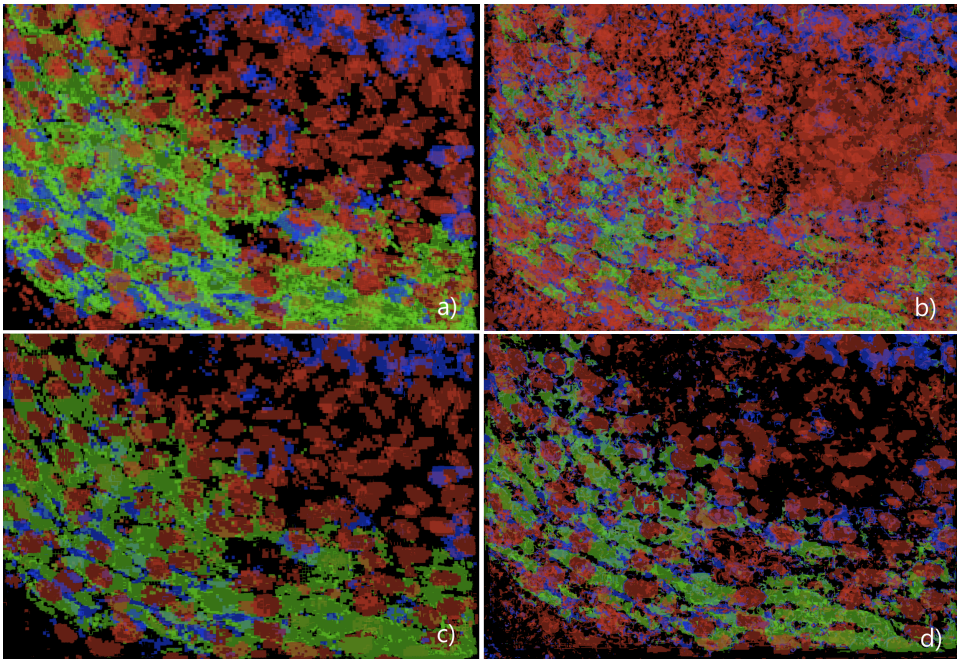


Figure 4.5: Visualization of a dataset with many points, leading to imperfect clustering and isolated values (clutter). Image a) shows NN sampling and c) adds clutter reduction to it. Image b) uses linear sampling in the 2D TF domain without artifact correction, while d) includes artifact correction.

This behavior is also preserved in the TF domain, as neighborhood-based DR algorithms are optimized to map similar points close together. As a result, if the ray step size is around 1, the likelihood of consecutively sampling two similar values is high, and artifacts are unlikely to form between them. Based on this, and on the fact that the region between two voxels where a sample may be misclassified is often quite narrow, we make the weak assumption that a material sampled twice in sequence likely reflects a real feature, not an artifact.

This assumption is not always reliable, such as when sampling both ends of an isolated voxel in mirrored locations, or when the ray is parallel to a surface, but the goal is to reduce artifacts overall, not eliminate every case.

Detecting clutter or artifacts requires more context than the two samples normally used by the TF. We therefore extend the ray state to two arrays of size five, tracking both sampled materials and positions. New samples shift values backwards, and corrected materials replace their original entries before being shifted. This forms a sliding window of context for classification.

Under this scheme, and by following our detection rules, the middle sample m_y is labeled as clutter in the following cases:

$$\mathbf{M} = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix} \vee \begin{bmatrix} m_x & m_y & m_x \end{bmatrix}$$

and as an artifact in these cases:

$$\mathbf{M} = \begin{bmatrix} m_x & m_x & m_y & m_z & m_z \end{bmatrix} \vee \begin{bmatrix} m_x & m_x & m_y & m_x & m_x \end{bmatrix}$$

If m_y is marked, we replace it with m_x in the case of clutter, thereby removing the isolated voxel. However, this is a relatively aggressive fix. To avoid replacing false positives in the case of artifacts, we instead apply NN sampling at m_y's location. This has a high chance of returning either m_x or m_z if the sample truly lies on a boundary between them, while preserving the chance of m_y being valid in the case of small isolated features. In either case, the value comes directly from the measured data and avoids introducing highly inaccurate material placements. This NN fallback is used when we aim to suppress artifacts while retaining most of the original data, even if some clutter remains.

Once m_y is evaluated (and possibly modified), its value becomes m_e in Equation 2.5, with the preceding sample designated m_s, and both passed into the TF as before. Since the samples after it serve only as context, we perform two extra samples at the end of the ray (by repeating the last valid position before going out of bounds), ensuring that border samples still reach the middle position and are composited (see Equation 2.6). Indices 0, 1, and 2 are initialized at the ray start.

The effect of this clutter reduction scheme can be seen in Figure 4.5c and Figure 4.5d.

Boundary Position Refinement

We also changed how this method estimates the actual surface position between the start and end of the ray step. The original two-step TF paper [25] proposes using the secant method, with the iso value separating the two materials as the target.

However, since the domain of our TF is 2D, the secant method is no longer sufficient, and the use of Broyden's method would be required instead. Furthermore, there is no longer a single well-defined iso value, as the intersection lies on the edge of an arbitrary shape rather than a point, making it unsuitable as a target. As a result, we opt to use a bisection method instead, where the bounds are adjusted at each step based on whether a surface is detected between the adjusted start and end points.

4.2.2. Surface-Based Effects

Scalar-based DVR typically incorporates surface-based effects such as surface shading or boundary enhancement. These effects are commonly used in scalar volumes to emphasize structural boundaries and improve visual clarity.

However, applying such effects in a multivariate context poses several challenges. Traditional techniques that rely on gradient magnitude or direction are not directly applicable, as multivariate datasets yield high-dimensional gradients that are difficult to interpret and visualize, making direction-based effects like Phong shading infeasible.

Moreover, visualizing all transitions between variables using gradient magnitude would introduce significant clutter and ambiguity, especially since transfer functions often suppress certain dimensions while emphasizing others.

To address this, we use the two-step transfer function, which provides a practical alternative to gradient-based methods. It allows the system to distinguish between surfaces and homogeneous regions without relying on explicit gradients, enabling surface-based effects through the methods described in the following subsections.

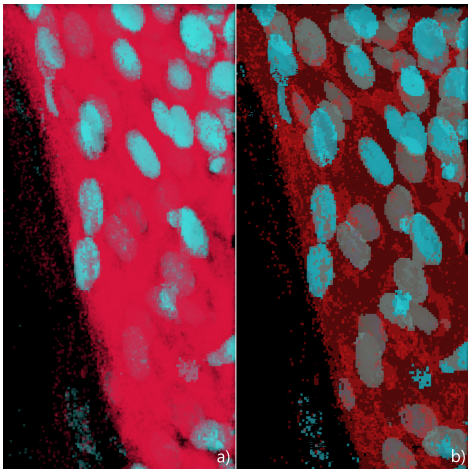


Figure 4.6: Comparison between a) NN sampling with the basic TF, and b) NN sampling with the two-step TF where homogeneous areas are rendered transparently.

Transparency Modulation at Surfaces

Since the two-step TF can treat surfaces and homogeneous areas differently, it is trivial for the user to increase the opacity of surfaces relative to homogeneous regions using the material combination matrix UI element we made, which can be seen in Figure 4.2d. The colors on the diagonal axis represent homogeneous areas, while the other cells define transitions between specific materials. This opens the possibility of visualizations where homogeneous regions do not occlude structures behind them, as shown in Figure 4.6, by assigning homogeneous areas a transparent color.

Surface Shading

Solving the issue of missing surface gradients is the only required step to enable local illumination models, such as Phong shading, within the current framework. Pre-computation of the gradients is not viable since it would not be able to accurately store the continuous surface positions in a 3D texture. Instead, gradients are computed dynamically during ray traversal.

Since we are only interested in the surface gradients, not those in homogeneous regions, the default traversal method is used unless a surface is detected. A surface is considered detected when the

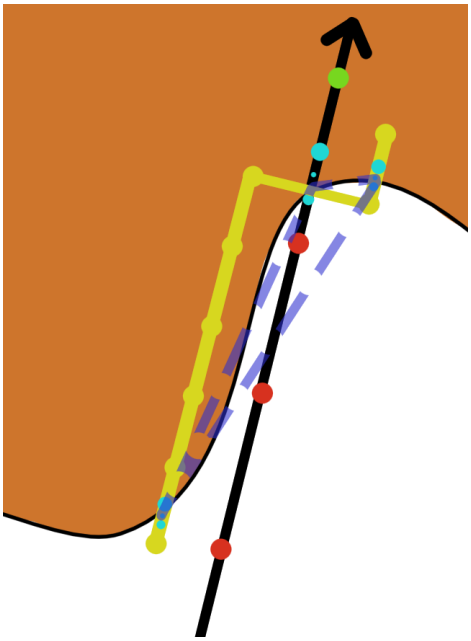


Figure 4.7: The surface triangle creation process (flattened for illustrative purposes). Green: material change detected. Light Blue: surface refinement steps. Yellow: offset rays seeking the surface. Dark Blue: constructed triangle using the located points

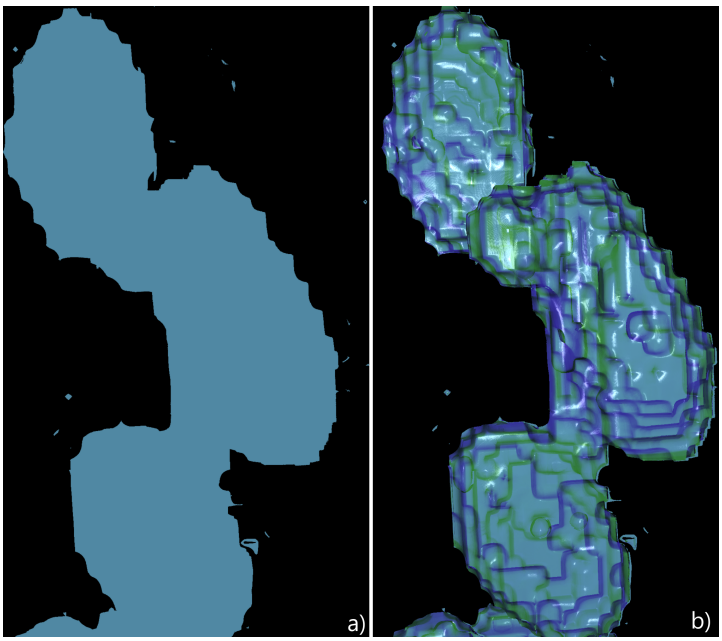


Figure 4.8: A render using the linear interpolation in the lower dimensional 2D space using the two-step TF. It has rays entering the cells set as green, and rays that leave as blue and homogeneous areas set as transparent. a) Without phong shading and b) with phong shading

material value at the beginning of a ray step differs from the value at its end. In this case, we construct a small triangle on the surface using the following procedure, which is also illustrated in Figure 4.7.

First, the surface intersection is refined using the method from Section 4.2.1. From this refined position $\vec{p}_0$, we define three offset points $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$ that form a triangle perpendicular to the view ray direction. These offsets are defined as:

$$\vec{v}_x = \text{normalize}\big(\vec{d} \times \vec{u}_1\big), \quad \vec{p}_x = \vec{p}_0 + \delta\, \vec{v}_x, \tag{4.1}$$

where $\vec{d}$ is the ray direction, δ is an offset scalar based on the current step size, and $\vec{u}_i$ are predefined offset vectors (e.g., $(0, 1, 0)$, $(-1, -1, 0)$, $(1, 1, 0)$).

From each of these offset points, we launch a short-range ray toward the surface using reduced step sizes, since we assume they are already close to a surface. If surface curvature is high, these rays may miss; to reduce this risk, they search up to a fixed number of steps (e.g., 3). If only one ray fails to detect a surface, its point is replaced with $\vec{p}_0$. If more than one ray fails, no shading is applied.

When three surface points are available, the surface normal is computed using the triangle cross product:

$$\vec{n} = \text{normalize}\big((\vec{p}_2 - \vec{p}_3) \times (\vec{p}_1 - \vec{p}_3)\big), \tag{4.2}$$

ensuring $\vec{n}$ faces the camera by flipping its sign if necessary.

This computed normal can then be used in the shading stage to produce Phong-lit surfaces, as illustrated in Figure 4.8.

4.3. Baseline Interpolation Methods

As discussed in Section 2.2, neighborhood-preserving dimensionality reduction (DR) methods are capable of visualizing complex multivariate data in low-dimensional spaces. They can reveal patterns, such

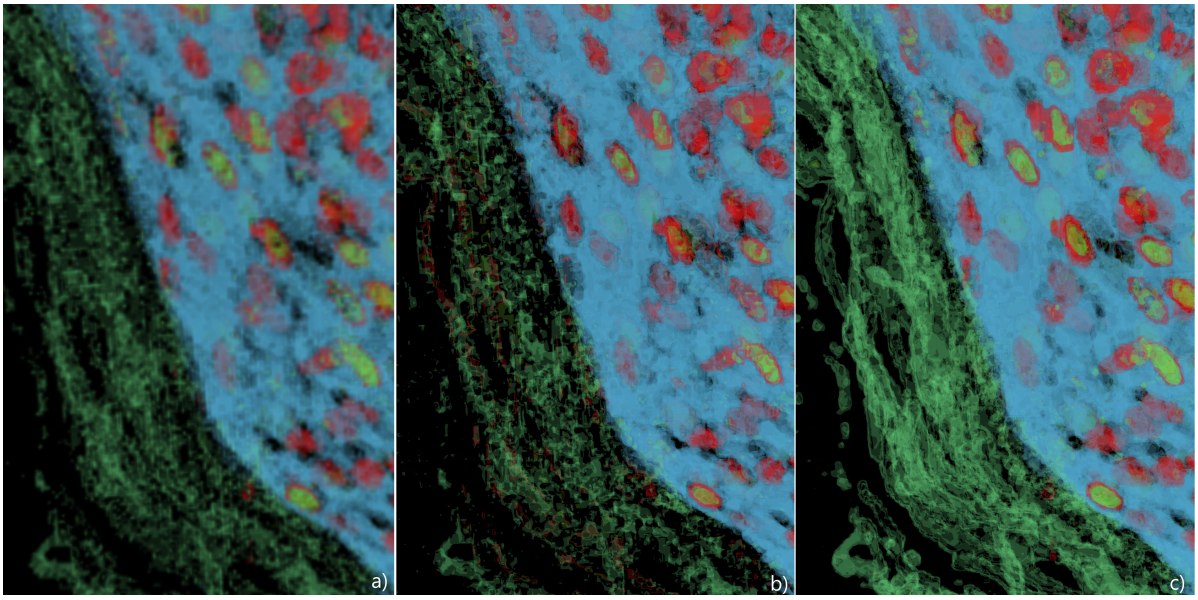


Figure 4.9: Comparison of the three render modes using linear interpolation with the basic TF: a) Interpolation in the color domain b) Interpolation in the 2D DR domain c) Interpolation in the full data domain with projected positions estimated using ANN

as gene expression clusters, that linear DR methods cannot capture effectively, making neighborhood-preserving DR essential in this context.

However, the transfer function (TF) is now defined in the embedded 2D space produced by a DR, and this is the domain in which interpolation must define new points during rendering, since we want to use post-classification as discussed in Section 2.1.4 (the reasons for using interpolation are detailed in Section 2.1.1). But because neighborhood-preserving DR does not have a consistent transformation across the data, interpolating directly in the low-dimensional domain leads to inaccurate results (see Section 2.2).

The methods introduced in this section represent these incorrect ways of applying linear interpolation. They provide useful reference points that highlight the challenges discussed earlier and help evaluate whether a more complex interpolation pipeline, introduced in a later section, is justified.

Color-Domain Rendering

In this method, each voxel in the volume is assigned a unique RGBA value based on the position it had in the embedded space and the TF applied to that location. These color values are then sampled using standard trilinear interpolation during rendering. As interpolation now occurs entirely in color space, the issues with the DR domain are sidestepped. However, since the TF is no longer applied during ray traversal, this constitutes a form of pre-classification.

The primary advantage of this approach is that the visualization roughly reflects the data as transformed by the DR and TF, since the colors are correct for the center of each voxel (the location where the actual measurement was taken), and interpolation of colors cannot result in high-frequency changes, and thus cannot deviate too much from those guaranteed correct values.

However, it also suffers from the same issues any method using pre-classification has (see Section 2.1.4). Namely, the lack of high-frequency changes results in a blurry visualization and the presence of black artifacts around the edges, as shown in Figure 4.9.

Additionally, this method does not support the two-step TF introduced earlier. As a result, any directional surface cues, such as shading or silhouette enhancement, cannot be applied using this approach.

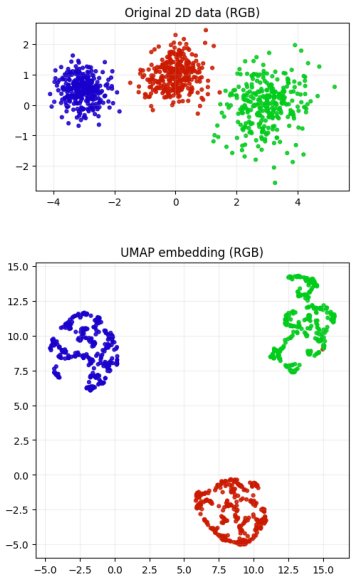


Figure 4.10: Due to displacement in the embedding, interpolation between clusters will not result in the same coloring in the two different spaces

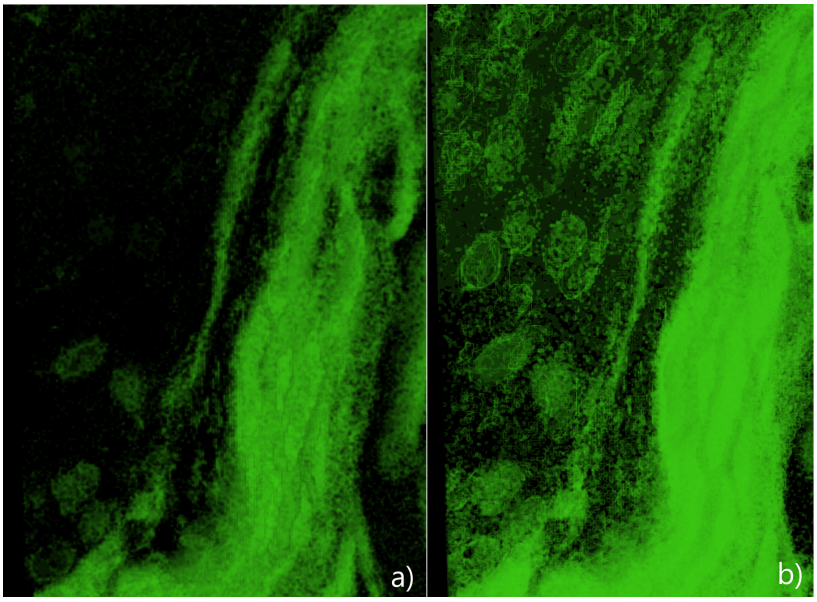


Figure 4.11: a) Color-domain-based rendering b) 2D domain-based rendering. In figure b), more structures are visible on the left side of the volume than in figure a).

2D Position-Domain Rendering

In this method, each voxel stores the position in the embedded space instead of the sampled original data. The resulting 2D coordinates are then directly interpolated and used to query the TF to get a color.

The benefit of this approach is that it is a straightforward and fast method of interpolation that can produce sharp transitions, unlike interpolation in the color domain. It is also still capable of using the two-step TF, and thus able to achieve its surface-based effects.

However, the main drawback lies in the nature of neighborhood-preserving DR: positions in the embedded space do not always correspond to relationships in the original high-dimensional space, especially between points in different clusters. As a result, interpolation can yield coordinates that return invalid data. This is demonstrated in Figure 4.10 with an example using 2D data. Here, interpolation between a green and a blue point could be classified as red in the original data, but interpolation in the embedded space will return a transparent color. At the same time, as long as it interpolates two points from the same neighbourhood, it will still have a high chance of returning the correct color.

These effects can clearly be seen in Figure 4.11b, where the left side of the volume displays green silhouettes, which are not present in Figure 4.11a, as a result from interpolation between clusters. However, it also correctly displays the actual highlighted structure on the right side of the volume, which visualizes the points inside a single cluster, without the issues present in the pre-classification method.

4.4. Linear Interpolation Approximation

As we have seen in Section 4.3, we can not simply interpolate in the embedded space (the 2D domain where the points are now defined). Instead, if we want correct linear interpolation, it must occur in the original high-dimensional space and then be projected into the embedded space. This would work for linear DR methods such as PCA [28], however, as discussed in Section 2.2, this is not feasible for neighborhood-preserving DR methods.

To overcome this, we approximate the projection of interpolated points by finding their nearest neighbors (NNs) in the high-dimensional space. We can use this to estimate an approximately valid position due to the nature of neighborhood-based DR algorithms. As stated in Section 2.2, points that are close together in the high-dimensional space should also be close in the low-dimensional space. Thus, the

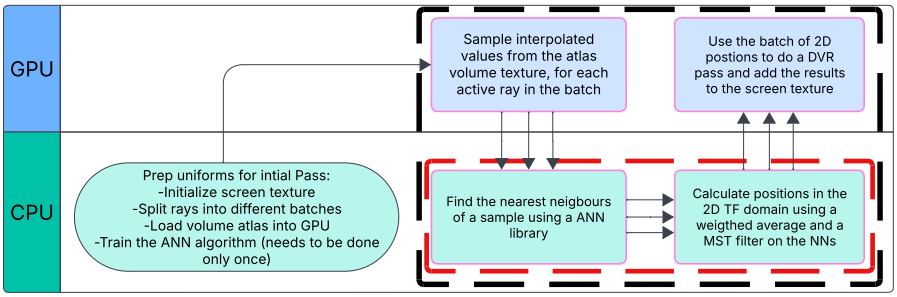


Figure 4.13: Steps in the Full Data Render Pipeline. Elements within the black dotted lines are executed once per batch. Elements within the red dotted lines are executed consecutively for each sample without waiting for other samples in the batch.

n NNs of the new point in the original data should also be near its theoretical low-dimensional position.

In this section, we discuss our design for a render pipeline that uses this fact to approximate the correct position of an interpolated point in the embedded space.

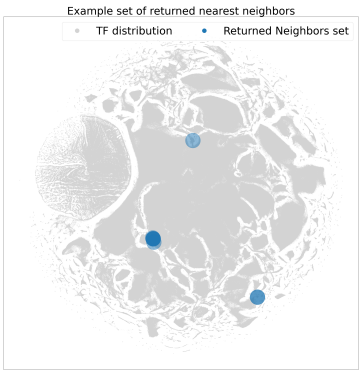


Figure 4.12: Illustration showing that the nearest neighbors an ANN algorithm returns are not always present in the same cluster.

4.4.1. Full Data Render Pipeline

This render pipeline, summarized in Figure 4.13, is executed in stages. First, we sample the interpolated point for which we need to approximate the embedded positions. This point is passed to an NN search algorithm to obtain its n high-dimensional NNs. Then, we collect the corresponding low-dimensional positions of these neighbors (which they have from the pre-calculated DR). Outliers are removed by creating an MST (minimum spanning tree) and cutting all connections above a certain length. This is necessary because neighborhood-preserving DR is not always able to place all points belonging to one cluster together in the embedded space, as seen in Section 4.1.2 or in Figure 4.12, which shows an example set of returned NNs. Since interpolating between points that are far apart produces incorrect positions, we only select the largest closely connected set of NN in the MST. We then apply a weighted average, based on closeness in the original space, to the remaining positions to obtain a single low-dimensional position. This position can then be used to sample the TF as seen in Figure 4.9.

This pipeline is executed in batches since it runs on both the CPU and GPU. This hybrid approach was chosen because a full GPU-based pipeline was not feasible, due to type of NN search libraries we use not being available on the GPU (further discussed in the next section). And while a full CPU-only approach, like a full GPU approach, would simplify the NN integration and allow features such as early stopping and the use of surface shading from Section 4.2.2, it is significantly slower due to limited threading capacity and the lack of hardware-accelerated interpolation. This interpolation overhead is more severe with multivariate datasets than with scalar datasets, due to the large number of dimensions that need to be interpolated. To reduce this cost, we use the GPU's built-in trilinear interpolation, which is possible when the input volume is stored as an atlas texture (as GPU textures normally handle at most 4 dimensions).

Thus, the hybrid approach was the only practical option. While not ideal for real-time systems, the CPU-GPU transfer latency is acceptable here, as ANN search is the primary bottleneck and dominates frame generation time, with large batches often taking dozens of seconds to process.

Due to memory limits (e.g., buffer sizes in OpenGL are capped and VRAM is often limited), storing all intermediate results in a single pass is often infeasible. For example, if the volume occupies the full screen, rendering at $1920 \cdot 1080$ resolution with 70-dimensional data and an average of 50 samples per ray would require $1920 \cdot 1080 \cdot 50 \cdot 70$ floats, or roughly 29 GB. To address this, we split the sampling step

into ray batches distributed evenly across the screen space, allowing partial results to be displayed progressively, as seen in Figure 4.14, while keeping memory usage within limits. A downside is that the steps in the black dotted box in Figure 4.13 must be repeated for each batch, increasing GPU-CPU transfer overhead.

As we do not perform early ray termination during data collection, the number of samples per ray is fixed and determined by the ray length divided by the step size. Both are known in advance and stored during a standard DVR preparation pass. This allows us to select which rays to include in a batch to make maximum use of available memory, thereby reducing the number of batches needed.

The list of low-dimensional positions obtained after the weighted average step (performed on the CPU) can then be rendered using a standard fragment shader similar to those used by other render modes, such as the two-step TF. The only modification is in the sampling step, it uses a buffer (containing aforementioned list) lookup based on a ray index, instead of a volume texture lookup.

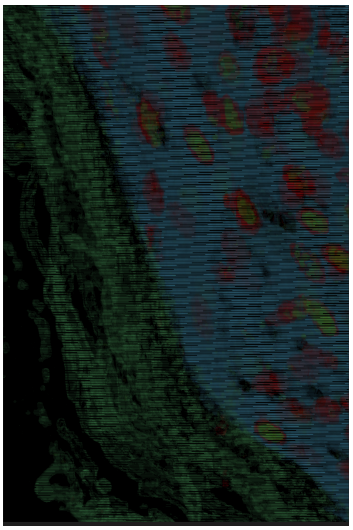


Figure 4.14: A snapshot of Figure 4.9 c) of how the gradual batch accumulation looks in progress

4.4.2. Nearest Neighbor Search Algorithms

Exact nearest neighbor algorithms are unsuitable for our use case, as even the fastest have a query time of $O(d \log N)$, which scales poorly for large, high-dimensional datasets [70]. Approximate nearest neighbor (ANN) methods produce approximate results with much better performance, which is essential for a visualization pipeline where low-latency feedback is important.

Among ANN methods, graph-based approaches are well suited for applications requiring high accuracy with tunable performance [45], though they have the drawback of increased memory usage to store the auxiliary graph.

Other approaches, such as hash-based methods like Locality-Sensitive Hashing (LSH) [4] or space-partitioning methods like IVF [5], use less memory but generally achieve lower accuracy and require more careful parameter tuning, making them less suitable here.

We require algorithms which are fast, easily tunable and have high accuracy for testing purposes. However, GPU graph-based ANN libraries that could be used directly in a custom shader require restrictive hardware and driver support. In addition, GPU VRAM is limited and would need to store both the auxiliary graph and the large multivariate dataset. Since CPU graph-based ANN libraries performed as well as or better than non-graph-based GPU implementations for our use case, and their performance is sufficient, we chose CPU-based graph ANN libraries.

4.5. NN smoothing methods

As we have just seen in Section 4.3 and 4.4, using linear interpolation in the embedded TF space has its issues, causing a misrepresentation of the data, while interpolation in the original data is not possible without a computationally expensive approximation pipeline. When using NN sampling (see Section 2.1.1), we do not have issues with artifacts since we only sample points that were present in the original dataset. However, NN sampling also returns blocky visualizations. To solve this downside, we propose two methods of smoothing the visualization that can be integrated into the rest of the visualization pipeline.

4.5.1. Integrated Surface Extraction Approach

To create the impression of smoother surfaces, we have integrated a part of the marching cubes (MC) [40] surface extraction algorithm into our DVR pipeline. We use MC as it is the least costly of the surface extraction algorithms to get a relatively smooth surface in a single ”cube”, which describes the eight voxels around a voxel corner and the area between their centers, as seen in Figure 4.16.

The MC algorithm works by sampling the eight voxels in a cube, which it passes to a lookup table that

tells the algorithm how the surface within the cube should look by providing a set of vertices and edge connections between said vertices. Using this, we can quickly retrieve a set of triangles that make up the extracted surface inside the sampled cube. We then calculate the intersection, if any, between the ray and each of these triangles. These intersections are then stored (along with the normal of the intersected triangle, for surface shading) in an array ordered based on their closeness to the start of the ray step, as that will be the order in which they are encountered. This ordering is important both for the compositing step and for how we deal with the issue of deciding which material the ray belongs to.

This algorithm needs to be run every time we take a sample, as we cannot detect at which voxels the ray could hit a surface anymore, since the surface smoothing can move the surface to neighboring voxels that do not contain any material change along the ray.

Material Assignment

This sampling strategy should be able to find all the extracted surface intersections along the ray, however, it still has a large issue. As discussed in Section 3.3.1, the original MC algorithm is built for binary classification, and thus, the lookup table it uses is not able to handle the multiple materials our TF can return. However, the generalized variant [24] is too slow for our purposes, and the multi-material variant [56] requires more than just the localized surfaces, as it relies on a follow-up smoothing step.

To mitigate this, we force a binary perspective: we see the material the ray currently has as zero and all other materials as one. This can lead to some misrepresentation of the surface, as can be seen in Figure 4.15. The left side is an example surface of how each corner treats the rest of the materials, while the right side shows the surface that would actually be produced if all corners had this perspective, which contains large gaps not filled in by any material. However, it does not result in large visual artifacts in practice, due to the relative rarity of a cube having more than two different materials and this misrepresentation only happening from one side (as we only look at it from a single angle), resulting in something closer to the left side of the figure, which is not as visually impactful.

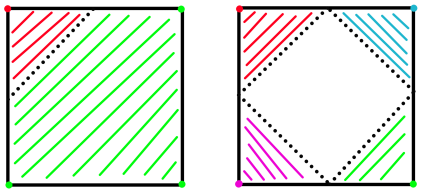


Figure 4.15: Illustration showing binary surface assumptions. Colored corner dots represent materials. Left: surface with binary input perspective. Right: incorrect result when applying it from all material perspectives.

However, even if we now have a surface to calculate intersections with, we still need a way to assign which material the ray will switch to after the intersection. The material we switch to is simply the first material sampled in the cube that is not the same as the current one. This guarantees some consistency since the cube always samples its voxels in the same order, no matter the ray direction. The materials along the ray, inside the sampled cube, are then defined by swapping between these two materials at every intersection point to represent the changing surface. Figure 4.17 shows the resulting visualization of this process.

Sampling Changes

To integrate this new sampling strategy into our pipeline, we have changed the way the ray collects samples from the volume, as visualized in Figure 4.16.

First, we now need to sample every cube (not a voxel) along the ray, since we change the way we look at the volume from a voxel-centric view to a cube-centric view, as indicated by the blue dotted line in Figure 4.16.

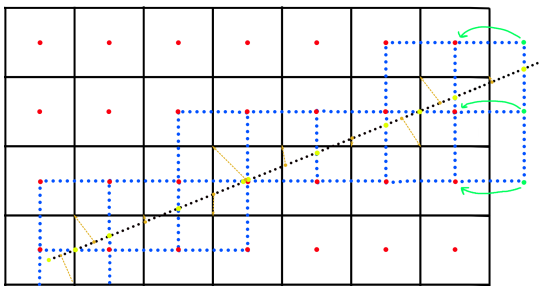


Figure 4.16: 2D view of modified DDA sampling. Blue dotted lines show MC cubes, red dots are stored voxels, yellow dots mark sample points, orange shows the actual cubes sampled, and green dots indicate clamped values.

Second, since we are now using NN sampling, we can use the DDA sampling variant first proposed by Amanatides et al. [3]. This ray traversal algorithm does not use a fixed step size but visits each voxel along the ray exactly once, ensuring we collect any impactful sample, as any extra sample would be redundant due to the use of NN sampling. Our implementation makes a slight change to this algorithm: instead of keeping track of a list of voxel indices, we keep

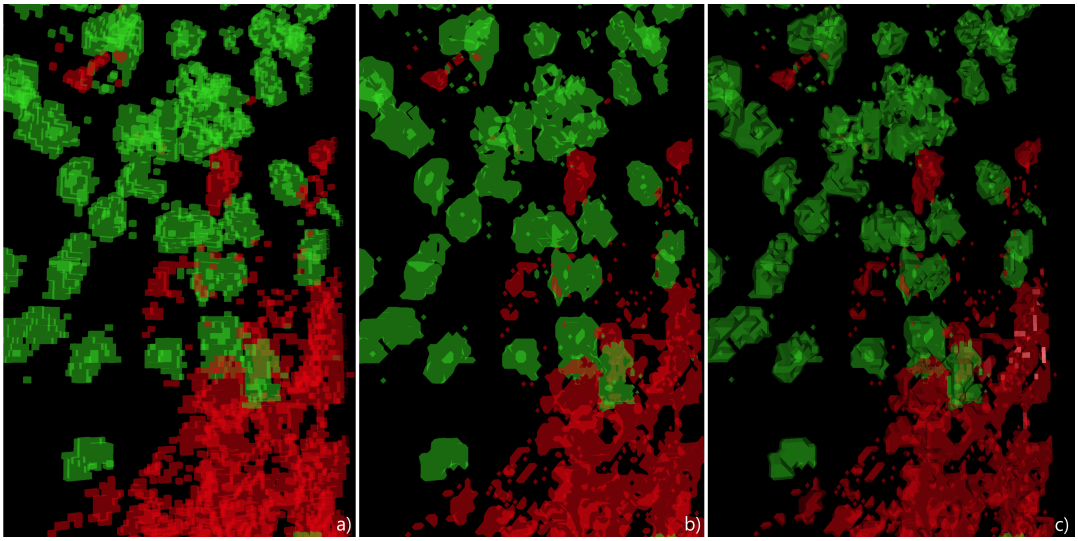


Figure 4.17: Comparison of NN sampling with and without smoothing: a) no smoothing b) smoothing c) smoothing plus surface shading

track of the traversed ray length, as we need this for the compositing step and some of the proposed algorithms. The change in perspective also influences the DDA sampling, as to get all the necessary information along the ray, we need to sample each cube once and not the voxels themselves. This can be done by sampling the grid with an offset, as indicated by the yellow dots in Figure 4.16.

4.5.2. Pre-processed Surface Extraction Approach

The method discussed in Section 4.5.1 suffers from two large issues: the extracted surface still has many flaws, and the performance is lacking. The flaws in the extracted surface are due to the lack of scalar values to guide the vertex positions, combined with the fact that the MC algorithm can only place vertices on the cube edges, and that MC cannot handle more than two materials. The reason the performance leaves something to be desired is the amount of unnecessary samples and calculations due to the inability to both reuse calculations and identify which regions of the volume actually require surface extraction.

Both of these issues can be solved by having a pre-render step that extracts a surface using SurfaceNets [10]. The SurfaceNets algorithm provides a better surface representation that can handle multiple materials and includes surface smoothing. By pre-calculating the surface, we know where the surface is located (and thus avoid performing extra operations in homogeneous regions), and we can avoid redoing the same calculations multiple times by reusing previous results.

Surface Representation in Storage

We do not want to pass a mesh to the DVR renderer, as the techniques to render those are completely different from our approach and will not integrate into our method well. Instead, we make use of the SurfaceNets property that every cube on the surface is represented by a single vertex. This means we can store all these vertices in a 3D texture, with each voxel representing a cube and storing the position of its corresponding vertex. This volume also doubles as an indicator of where a surface is located, and thus where extra surface calculations need to be performed.

For the connection of these vertices, we can create a lookup table as is done for MC. Since these vertices are connected around the cube edges, and cubes that neighbor an edge linking two voxels of different materials have their vertices connected, we can represent the edges as a binary pattern. Meaning there are at most 4096 possible ways for this vertex to be connected to its neighbors. This value can be stored in the fourth channel of the same volume as the vertices, reducing the need for sampling the volume containing the materials.

Material Assignment

Now we have the a set of triangles, and the rest of the algorithm roughly follows the process discussed in Section 4.5.1. However, SurfaceNets can handle multiple materials in its extracted surface, unlike MC. For the material assignment step, we find the two materials that are separated by the intersected triangle surface, and pick the one that is not the current ray material. This is correct since the surface is the boundary between those two materials, and as the ray traverses one of them, it will enter the other at the surface.

Downsides

The downsides are that, without using an additional complex detection step, the SurfaceNets algorithm has no way of knowing which parts of the volume are actually relevant to the current render. This requires it to extract the surface of the entire volume, even when the current rendering only visualizes a very small part of it.

It would also require extra storage to store all these results, and due to the often limited amounts of VRAM in a GPU plus the need to also store the volume (though for this render mode we only store a pre-classification of the volume, not the entire multivariate dataset), this might be an issue.

Finally, while it will speed up the render step, it does require some extra setup time due to the need to recalculate the surface every time the TF is changed.

4.6. Implementation Details

The entire software is implemented as a plugin for the ManiVault Studio framework [65]. Since this framework is designed to visualize multivariate data, it already contains several plugins with useful features, including ones missing from our pipeline, such as DR algorithms [54, 64, 65].

4.6.1. Framework Details

The core ManiVault framework [65], which handles most of the data sharing and preparation and serves as the back end for all plugins, is built using C++, though it also supports the integration of JavaScript libraries such as D3. The GUI and visualizations are built using Qt and OpenGL. These are the specifications that all plugins, including the one proposed in this thesis, must adhere to.

Each plugin can contribute multiple functions to the ManiVault framework. DR plugins add functions that can be applied to a dataset to create a new dataset, such as the embedded space into which the points are projected. Some plugins focus on data processing or define new dataset types and formats for storing data, but the functionality introduced by this thesis is primarily view widgets. These widgets allow the program to generate windows that users can scale and reposition freely, and visualize data in a specified way when provided with the necessary datasets.

4.6.2. Data Management and Linking

While all widgets are managed by the ManiVault framework [65], they operate as self-contained programs. This limits the interaction possibilities between components. The only allowed communication between widgets involves passing a reference to a dataset and optionally linking compatible UI parameters so that they share the same values. However, linking UI parameters must be done manually for each element, which is cumbersome and ideally avoided. When a widget modifies a referenced dataset, it is expected to emit an event notifying all other widgets linked to that dataset that it has changed.

The benefit of this design is that as long as widgets support the same dataset type, they can interact with one another, regardless of internal implementation details. This makes our approach agnostic to how the embedded space was computed, ensuring compatibility with any DR algorithm. In our application, this means that if two datasets share the same number of data points and one contains 2D data, then the corresponding indices are assumed to represent the same point, with the 2D dataset representing the lower-dimensional space into which the points are projected, and the other dataset representing the original high-dimensional data.

5 Results

In this chapter, we perform tests on the proposed methods and evaluate how they perform on different types of datasets, as well as how certain configurable parameters affect the resulting visualization. In Section 5.1, we discuss the testing setup and introduce the datasets used. Section 5.2 examines the visualizations produced by our methods and evaluates their respective strengths and weaknesses. Section 5.3 presents performance statistics and identifies bottlenecks. Finally, Section 5.4 briefly goes over what these results tell us about the best way to use our application.

5.1. Setup and Datasets

All tests were run on a desktop computer equipped with an AMD Ryzen 7 9800X3D 8-Core CPU at 4.70 GHz, 64 GB of DDR5 RAM, an NVIDIA GeForce RTX 5080 GPU with 16 GB of GDDR7 VRAM, and operating on a 64-bit architecture.

5.1.1. Used Datasets

All example datasets are derived from two primary transcriptomics datasets: one of a mouse brain [67], and the other a high-plex 3D CyCIF imaging dataset of a human tissue sample [68].

The mouse brain dataset is stored as a point cloud with 550 dimensions, as shown in Figure 5.1. Since we can not directly use a point cloud with our methods, we first convert it to a voxel grid. The user provides the grid dimensions, after which the plugin maps each point to its respective voxel and averages values when multiple points fall into the same voxel. While this new format is usable, it is not ideal since the original point cloud has a highly irregular sample density. To avoid excessive holes in the volume, the voxel grid resolution is kept relatively low, especially along the x-axis, where the slice spacing creates large gaps, requiring stretching of voxels. Due to this limitation, we prefer using the tissue dataset, though we do still include results from the brain datasets. We use 2 different grid dimensions, one being more in line with the actual dimensions of the underlying data (65×50×60) and the other with the sample density along each axis (50×150×200).

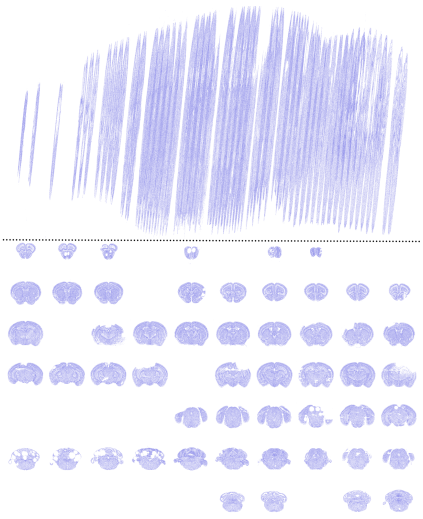


Figure 5.1: Mouse brain dataset mapped in a scatterplot: top image shows a side view, bottom image shows each individual slice as a flat image

The tissue sample dataset is already a dense voxel grid and consists of 70 dimensions. However, we do not use the entire dataset, as not all dimensions are informative, and the full resolution of the dataset is 10908×5508×194, resulting in 1.5 TB, which is impossible to use directly. We remove the redundant dimensions that, for example, encode co-occurring markers. Based on selections made by the dataset's

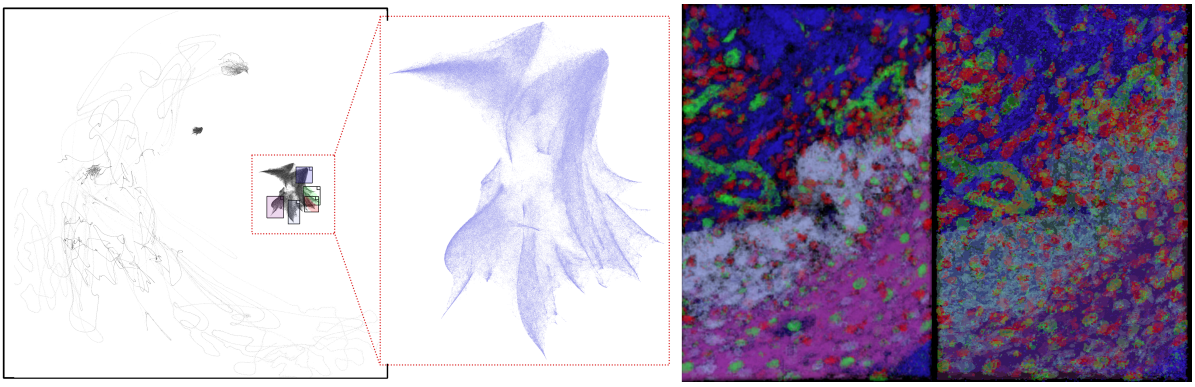


Figure 5.2: UMAP visualization of tissue dataset under compression level 3

original authors, we narrowed the used markers to the following 27: 5hmc, β-actin, CD103, CD11b, CD11c, CD163, CD20, CD206, CD31, CD3E, CD4, CD8a, FOXP3, HLA-AB (replaced with MCI-1), IRF1, Ki67, laminABC, MART1, panCK, PD1, podoplanin, S100A, S100B, SOX10, SOX9, pMLC2, and γH2AX, ensuring some features of the tissue dataset are not overrepresented.

The original data provided a multi-resolution pyramid with 6 layers, each subsequent layer halving the resolution of the x and y axes. The datasets we use are cutouts of this volume at various resolution layers, with the z-axis resolution also decreased, though one level less, so that the resolution of each axis becomes uniform (the original voxel dimensions are 140×140×280 nm) [68].

5.1.2. Used Dimensionality Reduction Algorithms

We mostly focus on the use of the GPGPU t-SNE [46, 52] DR algorithm provided by the ManiVault framework [65] to generate the transfer functions (TF) we used in the results. However, it is also possible to make use of other neighborhood-preserving DR algorithms such as UMAP [42, 48], as seen in Figure 5.2. However while it clusters points well, the UMAP implementation we have access to is slower than the t-SNE implementation, and a large area of the resulting embedded space is utilized by values representing empty space, which is detrimental to its usability.

5.2. Visualization Quality

By combining one of the interpolation methods discussed in Chapter 4 with one of the proposed transfer functions, we arrive at a specific way of rendering the data, referred to from here on as a render mode. Figure 5.3 shows which combinations have been implemented. The shading and clutter reduction options are not considered for classification as a specific render mode.

We do not present a single best render mode, as each has its own strengths and weaknesses, and no one mode is applicable to all situations. Instead, this section goes over the benefits and downsides of using a particular TF or sampling strategy.

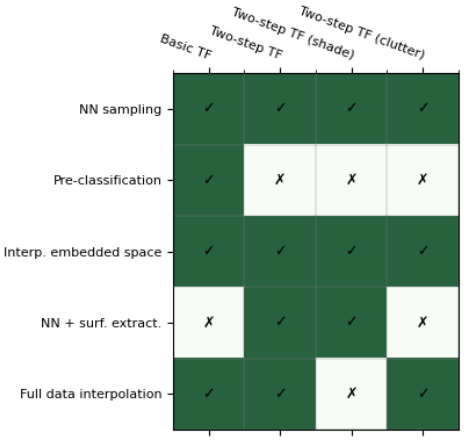


Figure 5.3: A table with all the implemented TFs on the x-axis and the sampling methods on the y-axis. It shows which specific combination has been implemented in our visualization software

5.2.1. Transfer Functions

How the program converts the taken samples into a color has a large effect on the resulting visualization. In this thesis, we have proposed two main types of TF: the basic TF in Section 4.1.1 and the two-step TF in Section 4.2.

As can be seen in images d) and h) of Figure 5.6a, the basic TF gives a less cluttered visualization of the dataset than the two-step TF, which makes it easier for

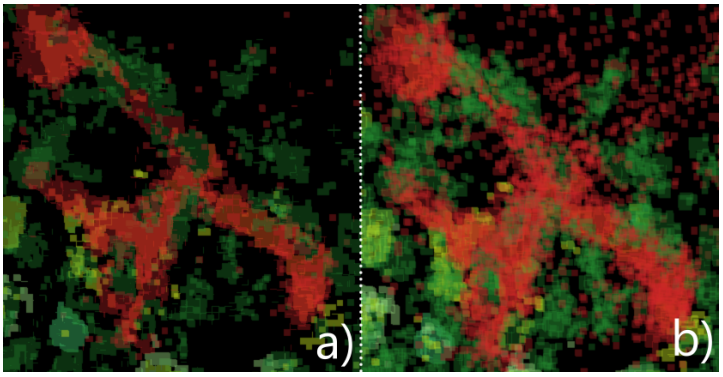


Figure 5.4: Clutter reduction comparison on a medium-resolution (compression 3) slice of the tissue dataset, a) with clutter reduction, b) without clutter reduction

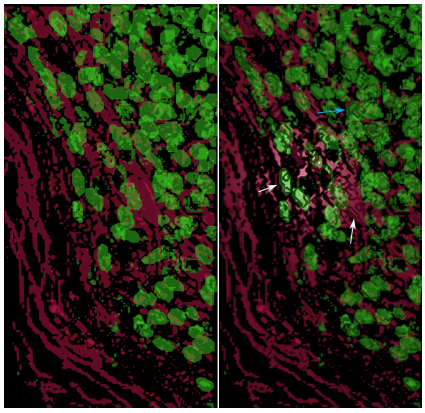


Figure 5.5: Shading comparison on a medium-resolution (compression 3) slice of the tissue dataset

the user to see the underlying structure of the data. However, as can be seen in images d) and h) of Figures 5.6b and 5.6c, this clutter is not as much of an issue in datasets containing a smaller number of points. In these figures, the two-step TF performs better at visualizing the structure of the data, particularly the structures that are otherwise occluded in the basic TF, as seen by the arrows numbered 1 in image h) of Figure 5.6b). These results follow the issues discussed in Sections 4.1.2 and 4.2.

Figures 5.6a and 5.6b already make use of the clutter reduction option proposed in Section 4.2.1. Since Figure 5.6a still has clear issues with clutter, it can be stated that it does not completely resolve the issue if the DR quality drops low enough. However, as is visualized in Figure 5.4, its use does still have a positive effect in general.

Finally, as can be seen in Figure 5.5, the surface shading option adds depth cues and highlights the surface. This can be beneficial in areas that lack sufficient detail themselves (white arrows). However, it also increases visual complexity and may reduce clarity when many surfaces overlap (blue arrow).

5.2.2. Sampling Modes

Before the TF can convert the samples it is given into a color, we need to sample and process these voxels. Here, we go over all the ways that have been proposed in this thesis, as shown in Figure 5.7, and discuss their effects on the resulting visualization.

NN Sampling

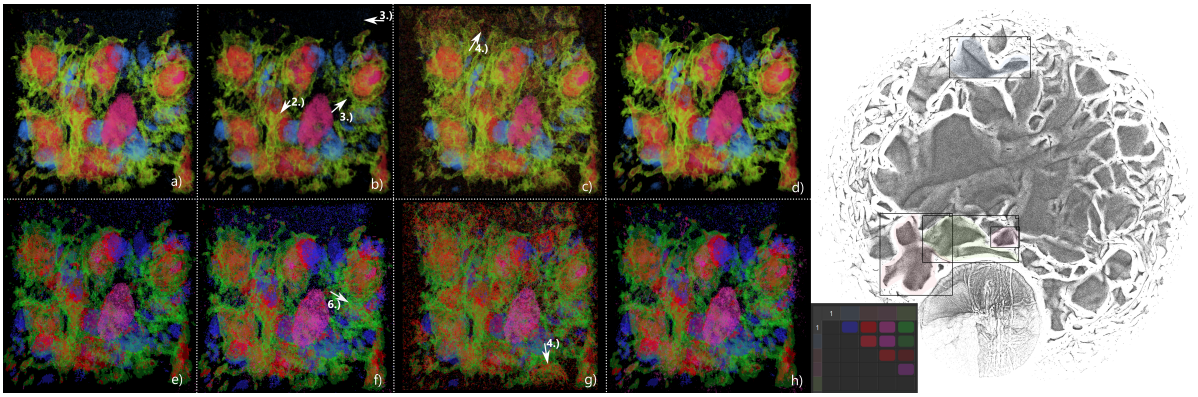
As can be seen in Figure 5.7c, NN sampling results in blocky surfaces, as expected. Though, as seen in image a) or e) in any of the subfigures of Figure 5.6, this blockyness does not significantly impact the visualization, especially when zoomed out. In Figure 5.4b, it can also be seen that when the clutter reduction is turned off, it gives a faithful visualization of the current TF, including clutter, which could be useful information.

Pre-classification

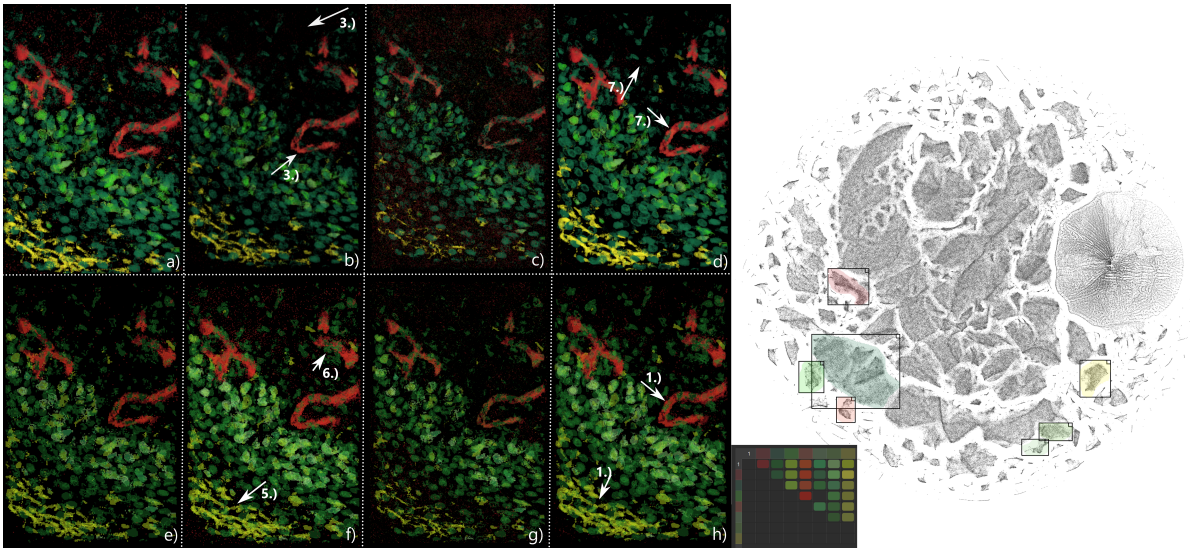
Figure 5.7d shows that when zooming in, the pre-classification method results in a blurry visualization with many black artifacts around its surfaces. However, as can be seen in image b) of Figures 5.6a and 5.6c at the arrows numbered 2, these black artifacts sometimes function as a sort of silhouette, which can have a positive effect on the understandability of the visualization. Image b) in Figure 5.6a and Figure 5.6b also show that pre-classification is good at filtering out clutter, as seen in the areas near the arrows numbered 3.

Interpolation in the Embedded Space

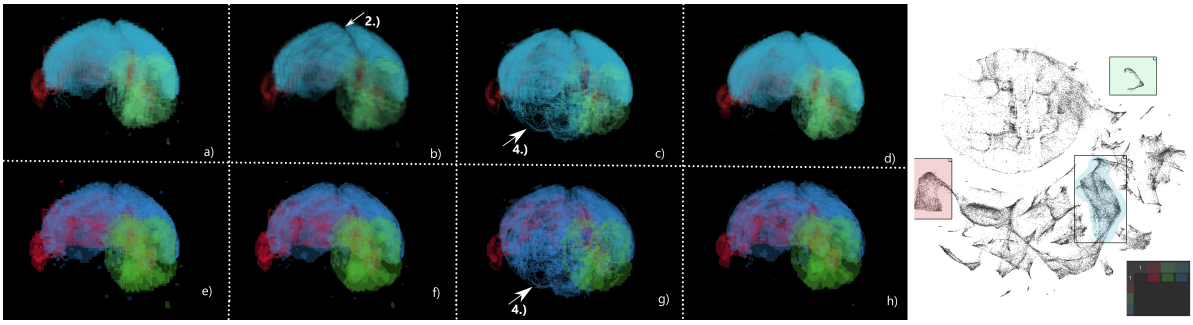
As seen in images c) and g) of Figures 5.6c and 5.6a, this mode can misrepresent the data in significant ways. In the spots indicated by the arrows numbered 4, additional or missing outlines appear compared to the other modes. This does not indicate a better representation of the data, but instead reflects the deformation in the 2D space, as explained in Section 4.3.



(a) High-resolution slice (compression 1) of the tissue dataset



(b) Medium-resolution (compression 3) slice of the tissue dataset



(c) Low-resolution variant of the brain dataset

Figure 5.6: Comparison of all render modes: a) NN sampling with classic TF, b) pre-classification sampling, c) linear interpolation in the embedded space with classic TF, d) full data interpolation pipeline with classic TF, e) NN sampling with material TF, f) NN sampling with integrated surface extraction using material TF, g) linear interpolation in the embedded space with material TF, h) full data interpolation pipeline with material TF. On the right side, the TF that is used for rendering the examples is displayed. The clutter reduction is turned on, and the surface shading is turned off where applicable.

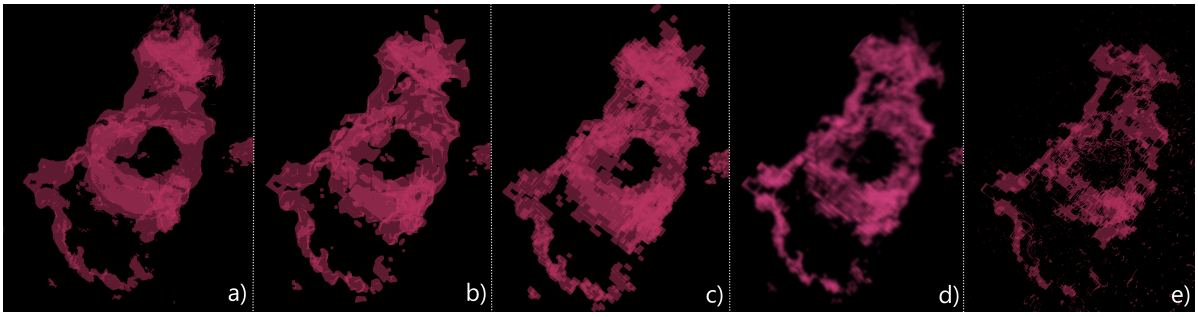


Figure 5.7: Surface comparison between different modes of interpolation in a close-up visualization of the tissue dataset. It makes use of the two-step TF where possible. a) full data interpolation approximation, b) NN sampling with surface extraction, NN sampling, pre-classification, interpolation in the embedded space

Furthermore, as seen in Figure 5.7e, the surfaces are also not always smooth, as misclassification of materials still causes a surface with many holes and spikes, along with floaty bits not attached to any surface.

NN Sampling with Surface Extraction

Figure 5.7b and image f) of Figure 5.6b (at the arrow numbered 5) show that NN sampling with surface extraction can provide a nice visualization with smooth surfaces. Though when zoomed in or when the resolution of the dataset is low, like in image f) of Figure 5.6c, it becomes apparent that the extracted surface it uses is still quite triangular with sharp edges instead of a roundish surface we would expect. It also does not handle clutter well, as can be seen in image f) of Figures 5.6a and 5.6b when looking at the area the arrows numbered 6 are pointing and comparing it to the visualization with image b). This is about what we expect from this visualization mode, not because of any inherent issues with the surface extraction, but because it can only make use of the two-step TF and does not have any clutter reduction implemented, making it suffer from its downside.

Full Data Interpolation Approximation

Figure 5.7 shows that the full data interpolation visualization returns the smooth, natural-looking surfaces that we would expect from the data. As can be seen in images d) and h) of Figure 5.6a, it also reduces the clutter in the visualization compared to the other sampling modes. This effect is less pronounced in the lower resolution dataset (as they already contain less clutter, see Section 4.1.2). However, when comparing the areas the arrows numbered 1 and 7 point at to the same area in the images representing the other render modes, it can be seen that it still has the least clutter of all sampling modes. This may be due to interpolation pushing samples toward surrounding material classifications, or because the NNs from the clutter samples are placed back into the correct regions, correcting initial misplacements introduced by the DR.

Figure 5.8 compares the effect of pairing it with several different ANN algorithms: HNSW with different parameters [44], and a single IVF [18][1]. In the figure, we place the returned sample positions in the TF and aggregate them in hexbins, which are then shown. It only plots the samples from around the same location as the points highlighted in green, which is every part of the volume excluding the zero values. It can be seen that the choice of ANN algorithm has a noticeable effect on the correctness of the returned positions, as shown by the results of the IVF algorithm and the HNSW algorithm with low build parameters. In those cases, the distribution of returned points does not cover the entire DR, which causes the full data interpolation to misrepresent the dataset.

The effect this has on the visualization is shown in Figure 5.9, which compares several renders of the tissue and brain datasets using ANN algorithms with varying accuracy (different parameters for the two datasets, since ANN performs well with less stringent settings on the tissue dataset). It can be seen that in both datasets, the visualization using the lowest accuracy ANN actively misrepresents the data in the highlighted areas. The IVF algorithms are already nearly identical to the most accurate visualization, though still showing small differences. It is thus important to choose an ANN algorithm

[1] Due to the inefficiency of exact NN finding algorithms, we were not able to calculate the recall of the larger datasets used in our tests. Instead, we opted to perform comparative tests using practical cases that could influence future design decisions.

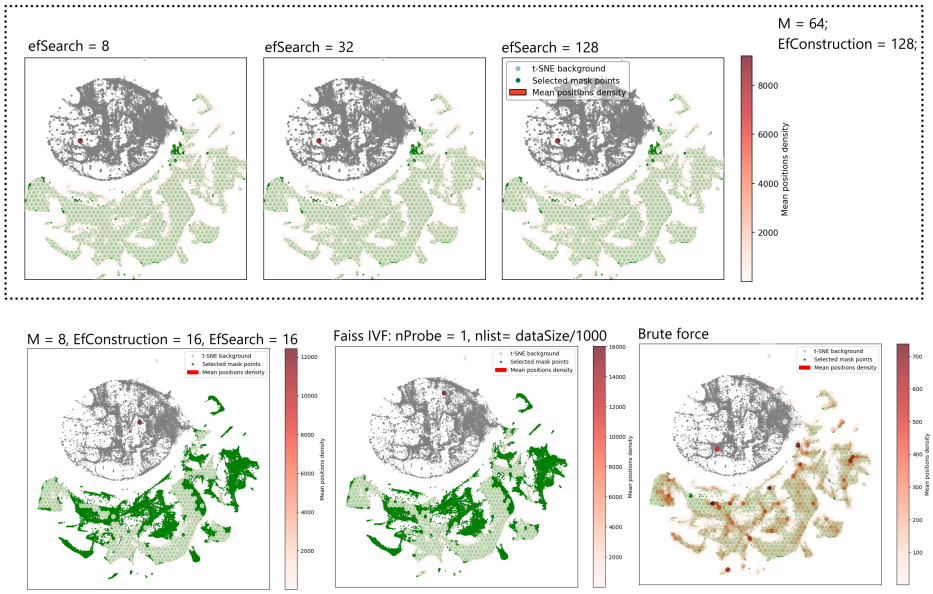


Figure 5.8: Effect of the ANN algorithm on the density of the returned mean positions in the brain dataset

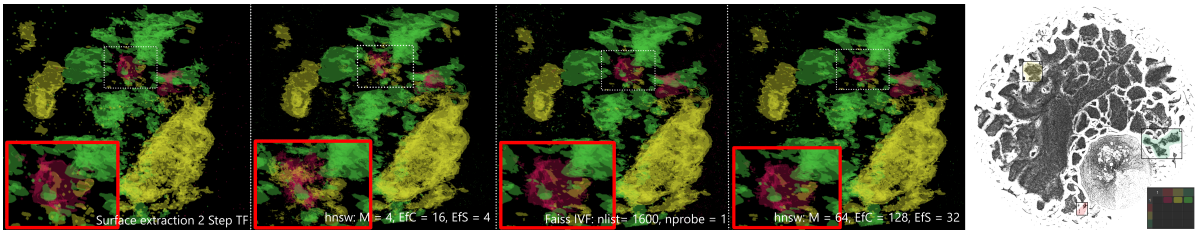
that is sufficiently accurate to represent the dataset. However, this is different for each dataset, and simply using the most accurate ANN algorithm possible will hurt performance.

We can also see in Figure 5.8 that accuracy, or at least the area of the DR that is covered, largely depends on the build parameters, not the search parameters (at least for the HNSW ANN), which have little effect on search times. This means that for optimal efficiency, it is ideal to use low search parameters. The reason we look at the covered area is that, since the relationship between points carries little meaning in the resulting 2D space, users are incentivized to group entire clusters using a single color/material. Thus, as long as the returned neighbor is in the same cluster as the actual neighbor, the visualization remains unaffected.

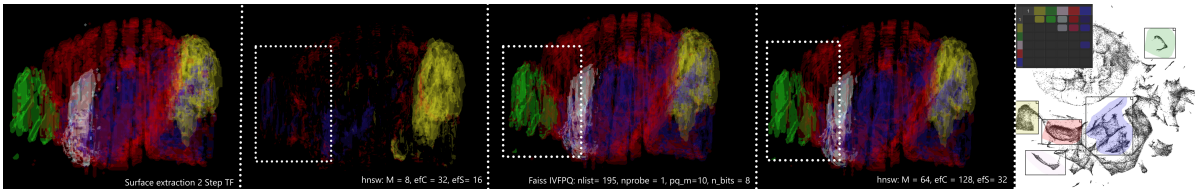
It can also be seen in Figure 5.8 that even the most accurate ANN tested does not match the exact NN search results. We are uncertain if this will impact the visualization in a significant way, as we were unable to render an actual image with the exact NN algorithm. However, due to the reasoning discussed in the previous paragraph, we assume the visual difference between a sufficiently accurate ANN algorithm and an exact NN algorithm will not be significant.

However, the accuracy of the ANN algorithms is not the only factor that impacts the visualization when making use of the full data interpolation sampling mode. How many NNs we use to calculate our position estimate in the embedded space also has an impact. In Figure 5.10, we have visualized the effects of using multiple neighbors versus few, as well as the effect of the MST cluster filtering (as explained in Section 5.2.2) on the spread of the returned points. In the figure, we plot all 2D positions we estimated from the linearly interpolated points from a single render of the tissue dataset in red. We do this for various amounts of returned NNs, as well as toggling the MST filtering on or off. When zooming in on the area highlighted by the black dotted box, we observe that without MST cluster filtering, more neighbors simply result in more noise, with the estimated points often being placed outside of any defined clusters. The cause of this has already been discussed in Section 4.4.1 and can clearly be observed in the area highlighted by the blue dotted box, namely, that the NNs returned by the ANN are not always confined to a single cluster, and interpolating between clusters leads to incorrect results due to the non-linear domain in which the TF is defined.

In Figure 5.11, we visualize a slice of the tissue dataset using either 9 or a single NN. We can see that when using the MST cluster filtering, using 9 NNs results in a similar output to using only a single NN. The visual differences between the two are minimal. We have highlighted some of the more noticeable differences, and even there, the changes are so small that we cannot confidently conclude



(a) High-resolution (compression 1) tissue dataset



(b) Low-resolution brain dataset

Figure 5.9: Comparison of how ANN algorithms with different accuracies affect the visualization. The leftmost image is a baseline to compare against. The rest of the images are sorted from left to right based on how accurate the used ANN is.

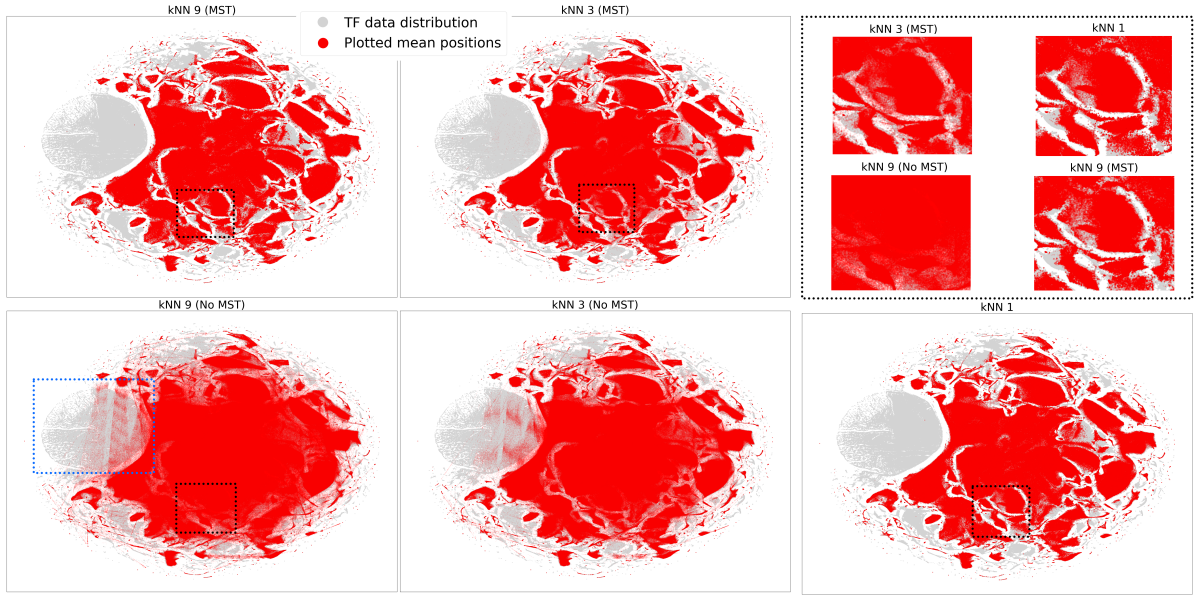


Figure 5.10: Effect of varying the number of nearest neighbours (NNs) and enabling/disabling MST cluster filtering on 2D position estimates for the tissue dataset. Red points show estimated positions from linearly sampled locations in a single render. The black dotted box highlights an area where, without MST filtering, increasing the number of NNs produces more noise and places estimates outside of defined clusters. The images in the top right are magnifications of these areas highlighted by the black dots. The blue dotted box marks a clear case of incorrect results due to linear interpolation in the non-linear TF domain.

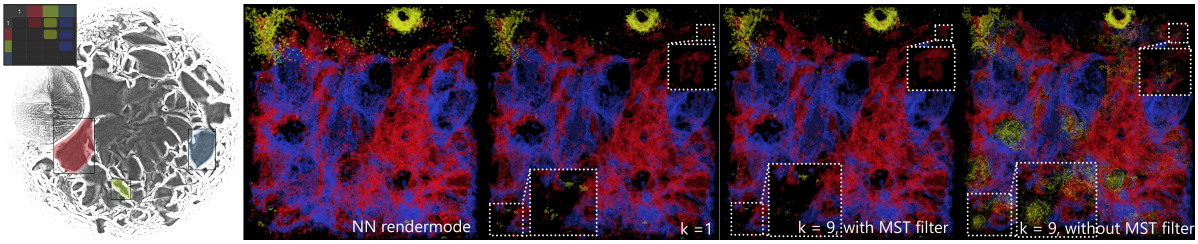


Figure 5.11: Comparison of visualizations of a slice of the tissue dataset using either 9 NNs or a single NN for 2D position estimation with MST cluster filtering applied.

which representation is more accurate. However, we can conclude that using 9 NNs without MST filtering results in a significant misrepresentation of the data.

The reason for this again aligns with the reasoning used before: it does not significantly matter where within a cluster a point is placed, as long as it falls inside the correct cluster. Since we select the largest cluster of NNs, calculating the weighted mean from them will likely yield the same result as taking only a single NN. Apart from scenarios in which the user does not select an entire cluster for some reason. The only difference is that when using a single NN, it is possible that the selected NN has not been placed correctly, i.e., as clutter. Increasing the number of NNs and identifying the largest cluster among them reduces the risk of relying on misplaced clutter to determine the position of our interpolated point. Conversely, the MST filtering only takes into account the distance between points, so interpolation between close clusters can still occur. We can therefore not conclude which method returns more accurate results.

Overall, it can be said that the full data interpolation returns the best overall visualization of all the sampling modes, though its quality depends on the ANN algorithm it is paired with.

5.3. Algorithm Performance

With interactive visualizations, algorithm speed is important, as a slow algorithm can result in a choppy visualization when changing the camera and/or TF. This reduces the software's interactivity, which is undesirable. Here, we test the speed of each render mode.

As shown in Figures 5.13a and 5.13b, most render modes remain comfortably within real-time rendering territory. Even methods that add significant extra steps, such as the NN sampling mode with surface extraction or the shading option when interpolating in the embedded space, do not slow down the render time enough to loose the interactivity of the application.

However, as shown in Figures 5.14a and 5.14b, the full data interpolation approximation mode is very slow, with render times ranging from a dozen seconds to several minutes. To explain this, we measured each component of the full pipeline separately, as seen in Figure 4.13 (excluding the ANN training stage, which can be pre-calculated). The bottleneck lies in the search and collect stages of the pipeline. The performance of the search stage depends on the speed of the ANN algorithms, which can be optimized for higher accuracy at the expense of performance. The collect stage is heavily dependent on data dimensionality, as higher dimensionality increases the amount of data transferred between the CPU and GPU, since each sample requires more storage. It also increases the number of texture sampling operations, as we use an atlas texture to store the whole dataset. We do this since a GPU texture can have at most 4 dimensions, thus the entire dataset is split into multiple datasets with 4 dimensions each. However, disabling these multiple texture lookups only speeds up the collection stage by about 10% in the brain dataset. Therefore, if the pipeline were fully on the GPU, the performance drop in the collection stage caused by increased dimensionality would largely disappear.

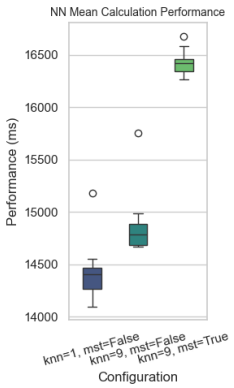
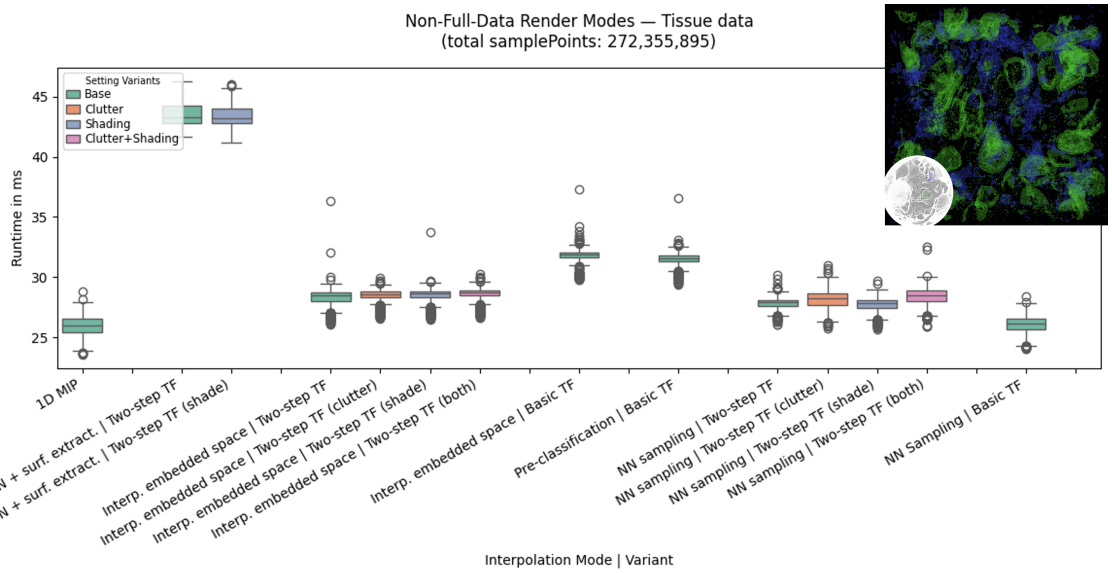


Figure 5.12: Performance metrics for the mean calculation step of the full data pipeline on the tissue dataset

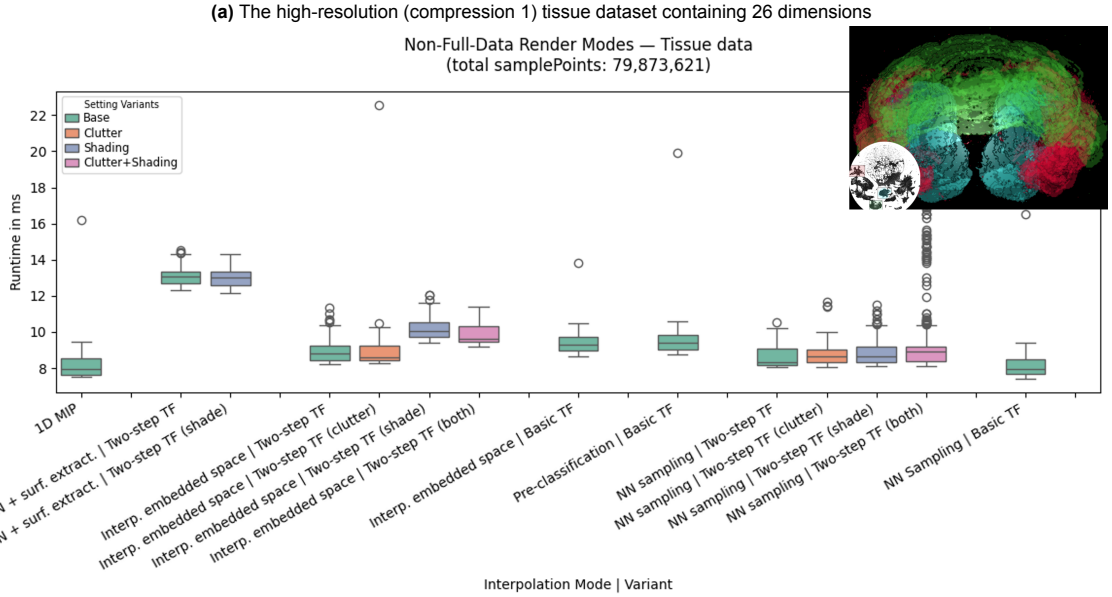
Finally, in Figure 5.12, we see that the performance impact of using one versus many NNs in the mean calculation step of the full data interpolation approximation does have some impact, around 15% for 1 NN versus 9 NNs with MST filter, with most of that performance decrease being caused by the MST filter.

5.4. Recommendation

Using all the results above, we can conclude that the full data linear approximation pipeline has the potential to deliver the best visual results overall. However, its slower rendering time prevents interactive use, making it better suited for final image generation. And using the real-time rendering modes for navigation and real-time interaction. For these purposes, the pre-classification render mode is likely the most suitable, as it provides a decent visualization under most circumstances. Depending on the dataset, we might also want to use either the two-step TF NN sampling mode with its clutter reduction,

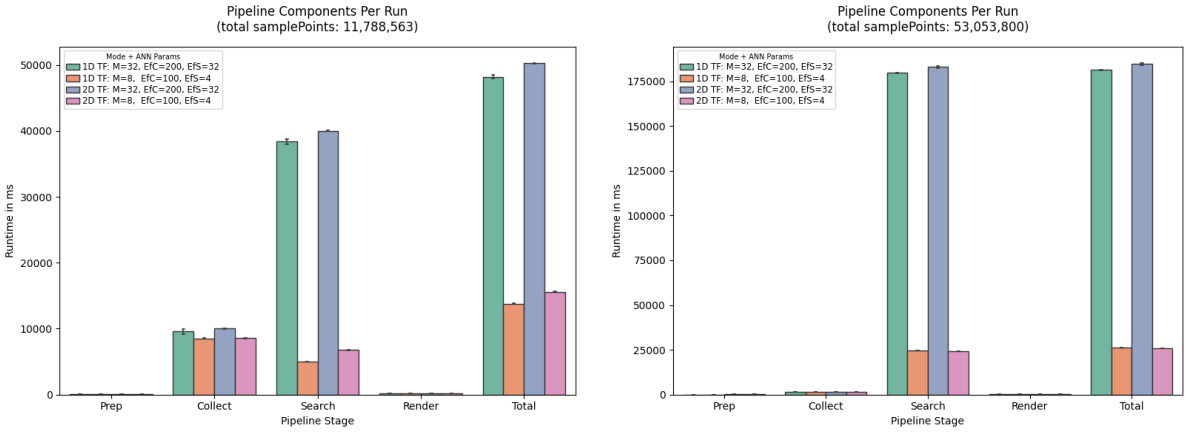


(a) The high-resolution (compression 1) tissue dataset containing 26 dimensions



(b) The high-resolution brain dataset containing 550 dimensions

Figure 5.13: Box plot performance benchmarks of the real-time render modes on a high-resolution screen (4k)



(a) The high-resolution brain dataset containing 550 dimensions

(b) The high-resolution (compression 1) tissue dataset containing 26 dimensions

Figure 5.14: Bar plot performance benchmarks of the full data interpolation render modes on a low-resolution screen (1080p)

or the two-step TF with surface extraction, as they have the potential to provide better visualization than the pre-classification method. They are, however, not suitable as the go-to render mode for all circumstances, as visualization quality depends on the dataset and TF used.

Regarding which ANN algorithm to use, the best option would be a graph-based ANN with its built parameters optimized to cover the entire data distribution while keeping the search parameters low. This provides fast query times along with accuracy that should be sufficient for most datasets, avoiding the need for heavy fine-tuning.

We also recommend limiting the number of returned NNs to a single one, as it is unclear whether there is any improvement in visualization quality from using more neighbors. It is therefore most likely not worth the performance trade-off in most scenarios.

6

Conclusion and Discussion

In this work, we provide a transfer function (TF) for visualizing high-dimensional multivariate volume datasets. We accomplish this by making use of neighborhood-preserving dimensionality reduction (DR), which presents an TF that does not increase in complexity as the dimensionality of the data increases, offering a simple overview of points sharing high similarity.

We have proposed four different methods for interpolating the points in the non-linear domain, where the points are now defined. These interpolation methods are: nearest neighbour (NN) sampling, interpolation in the color domain (pre-classification), interpolation in embedded space (the 2D non-linear domain where the points are defined), and finally using linear interpolation in the original space and estimating its position in the embedded space using approximate nearest neighbors (ANN).

Each of the proposed interpolation modes has its weaknesses, such that there is no single method that is always best to use. The full data approximation method, utilizing an ANN algorithm, provides the best visualization and representation of the data. However, it is very slow and not suitable for real-time rendering. Interpolation in the embedded space gives a misrepresentation of the underlying data and is thus the least useful of the methods. The pre-classification and NN sampling methods both give a reasonable representation and include mechanisms for dealing with clutter in the given data, though their visualizations suffer from either blurry or jagged lines. That said, they are very useful for interactively working with the data before switching to a potentially better visualization, such as the approximation method. The pre-classification method especially would be a good default mode for this, as it seems to provide at least a decent, though not always the best, visualization for any dataset and TF.

We also introduced a modified version of the two-step TF [25] that can efficiently simulate surface-based effects such as opacity modulation and surface shading, which are no longer possible in multivariate data due to a lack of usable gradient data. It also introduced a way to combine the NN sampling method with a surface extraction algorithm, resulting in smoother edges. These methods do not always improve the visualization, however, as they may highlight any clutter in the data.

Thus, the user must adapt their rendering method based on their dataset and needs, as there is no single best way of visualizing the dataset, among the proposed methods.

6.1. Future Work

There are still several improvements that can be made to enhance the current rendering pipeline.

From our testing, we have seen that in very high-dimensional datasets, the GPU-CPU transfer times of the full data approximation pipeline can become non-trivial. This means that a full GPU-based implementation of that render mode could greatly speed up the process, in addition to potential speed-ups gained from using a GPU version of the ANN algorithm. It would also open up the possibility of estimating the 2D position immediately after sampling the high-dimensional position, which would allow for the integration of the proposed surface shading method or enable features such as early ray termination.

There is also the method we proposed in Section 4.5.2, which we were unfortunately unable to implement. This method could be further improved by using the pre-processing step to filter out clutter, with its strength tunable via a parameter. This would address the main downsides of the surface extraction render mode and might result in a real-time render mode that works well across many different datasets.

Lastly, further developing the neighborhood-preserving dimensionality reduction algorithm, which we did not explore in depth as it was not intrinsic to the proposed method, could improve visualization quality. Improvements that reduce clutter or enable the processing of larger datasets would be especially beneficial. It may also be interesting to investigate how our method could be combined with a hierarchical DR approach such as HSNE [51].

References

[1] M. Agus et al. “Volume Puzzle: visual analysis of segmented volume data with multivariate attributes”. In: *2022 IEEE Visualization and Visual Analytics (VIS)*. 2022, pp. 130–134. DOI: 10.1109/VIS54862.2022.00035.

[2] Mohammed Ali et al. “TimeCluster: Dimension Reduction Applied to Temporal Data For Visual Analytics”. In: *The Visual Computer* 35.6 (2019), pp. 1013–1026. DOI: 10.1007/s00371-019-01673-y. URL: https://doi.org/10.1007/s00371-019-01673-y.

[3] John Amanatides and Andrew Woo. “A Fast Voxel Traversal Algorithm for Ray Tracing”. In: *Proceedings of EuroGraphics* 87 (Aug. 1987).

[4] Alexandr Andoni and Piotr Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 117–122. ISSN: 0001-0782. DOI: 10.1145/1327452.1327494. URL: https://doi.org/10.1145/1327452.1327494.

[5] Kazuo Aoyama, Kazumi Saito, and Tetsuo Ikeda. *Inverted-File k-Means Clustering: Performance Analysis*. 2020. arXiv: 2002.09094 [stat.ML]. URL: https://arxiv.org/abs/2002.09094.

[6] Ayan Biswas et al. “An Information-Aware Framework for Exploring Multivariate Data Sets”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2683–2692. DOI: 10.1109/TVCG.2013.133.

[7] Ingwer Borg and Patrick J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[8] Rita Borgo et al. “Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications”. In: *Eurographics State of the Art Reports*. EG STARs (May 2013). http://diglib.eg.org/EG/DL/conf/EG
063.pdf, pp. 39–63. URL: https://www.cg.tuwien.ac.at/research/publications/2013/borgo-2013-gly/.

[9] Alexander Broersen and Robert van Liere. “Transfer Functions for Imaging Spectroscopy Data using Principal Component Analysis.” In: Jan. 2005, pp. 117–123. DOI: 10.2312/VisSym/EuroVis05/117-123.

[10] P. Bruin et al. “Improving Triangle Mesh Quality with SurfaceNets”. In: vol. 1935. Nov. 2004, pp. 69–102. ISBN: 978-3-540-41189-5. DOI: 10.1007/978-3-540-40899-4_83.

[11] Shenghui Cheng and Klaus Mueller. “Improving the fidelity of contextual data layouts using a Generalized Barycentric Coordinates framework”. In: *2015 IEEE Pacific Visualization Symposium (PacificVis)*. 2015, pp. 295–302. DOI: 10.1109/PACIFICVIS.2015.7156390.

[12] Daniel Jie Yuan Chin et al. “GPU-Accelerated Enhanced Marching Cubes 33 for Fast 3D Reconstruction of Large Bone Defect CT Images”. In: *Advances in Visual Informatics*. Ed. by Halimah Badioze Zaman et al. Cham: Springer International Publishing, 2021, pp. 374–384. ISBN: 978-3-030-90235-3.

[13] Hyunghoon Cho, Bonnie Berger, and Jian Peng. “Generalizable and Scalable Visualization of Single-Cell Data Using Neural Networks”. In: *Cell Systems* 7.2 (2018), 185–191.e4. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2018.05.017. URL: https://www.sciencedirect.com/science/article/pii/S2405471218302357.

[14] Matthew C. Cieslak et al. “t-Distributed Stochastic Neighbor Embedding (t-SNE): A tool for eco-physiological transcriptomic analysis.” In: *Marine genomics* (2019), p. 100723. URL: https://api.semanticscholar.org/CorpusID:208498829.

[15] Carlos A. Dietrich et al. “Marching Cubes without Skinny Triangles”. In: *Computing in Science & Engineering* 11.2 (2009), pp. 82–87. DOI: 10.1109/MCSE.2009.34.

[16] George Dimitriadis, Joana Neto, and Adam Kampff. *T-SNE visualization of large-scale neural recordings*. Nov. 2016. DOI: 10.1101/087395.

[17] Zhiyu Ding et al. “Visual inspection of multivariate volume data based on multi-class noise sampling”. In: *The Visual Computer* 32 (Mar. 2015). DOI: 10.1007/s00371-015-1070-6.

[18] Matthijs Douze et al. “The Faiss library”. In: (2024). arXiv: 2401.08281 [cs.LG].

[19] Soumya Dutta et al. “Pointwise information guided visual analysis of time-varying multi-fields”. In: SA ’17. Bangkok, Thailand: Association for Computing Machinery, 2017. ISBN: 9781450354110. DOI: 10.1145/3139295.3139298. URL: https://doi.org/10.1145/3139295.3139298.

[20] Klaus Engel and Thomas Ertl. “Interactive high-quality volume rendering with flexible consumer graphics hardware.” In: *Eurographics (State of the Art Reports)*. 2002.

[21] Sarah Frisken. “SurfaceNets for Multi-Label Segmentations with Preservation of Sharp Boundaries”. In: *The Journal of computer graphics techniques* 11 (Feb. 2022), pp. 34–54.

[22] Hanqi Guo, He Xiao, and Xiaoru Yuan. “Scalable Multivariate Volume Visualization and Analysis Based on Dimension Projection and Parallel Coordinates”. In: *IEEE Transactions on Visualization and Computer Graphics* 18 (Mar. 2012), pp. 1397–1410. DOI: 10.1109/TVCG.2012.80.

[23] Xiangyang He et al. “Multivariate Spatial Data Visualization: A Survey”. In: *Journal of Visualization* 22 (2019), pp. 897–912. DOI: 10.1007/s12650-019-00584-3. URL: https://doi.org/10.1007/s12650-019-00584-3.

[24] Hans-Christian Hege et al. “A Generalized Marching Cubes Algorithm Based On Non-Binary Classifications”. In: *ZIB Preprint SC-97-05* (Jan. 1997).

[25] Oleg Igouchkine, Yubo Zhang, and Kwan-Liu Ma. “Multi-Material Volume Rendering with a Physically-Based Surface Reflection Model”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.12 (2018), pp. 3147–3159. DOI: 10.1109/TVCG.2017.2784830.

[26] Jochen Jankowai and Ingrid Hotz. “Feature Level-Sets: Generalizing Iso-Surfaces to Multi-Variate Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.2 (2020), pp. 1308–1319. DOI: 10.1109/TVCG.2018.2867488.

[27] Paulo Joia et al. “Local Affine Multidimensional Projection”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2563–2571.

[28] Ian T. Jolliffe. “Principal Component Analysis”. In: *Springer Series in Statistics* (2002).

[29] Arie Kaufman. “Voxels as a Computational Representation of Geometry”. In: *SIGGRAPH Course Notes*. 1996.

[30] Han Suk Kim et al. “Dimensionality Reduction on Multi-Dimensional Transfer Functions for Multi-Channel Volume Data Sets”. In: *Information Visualization* 9.3 (2010). PMID: 21841914, pp. 167–180. DOI: 10.1057/ivs.2010.6. eprint: https://doi.org/10.1057/ivs.2010.6. URL: https://doi.org/10.1057/ivs.2010.6.

[31] J. Kniss, G. Kindlmann, and C. Hansen. “Multidimensional transfer functions for interactive volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 8.3 (2002), pp. 270–285. DOI: 10.1109/TVCG.2002.1021579.

[32] Dmitry Kobak and Philipp Berens. “The Art of Using t-SNE for Single-Cell Transcriptomics”. In: *Nature Communications* 10.1 (2019), pp. 1–10. DOI: 10.1038/s41467-019-13056-x. URL: https://doi.org/10.1038/s41467-019-13056-x.

[33] J. Kruger and R. Westermann. “Acceleration Techniques for GPU-based Volume Rendering”. In: IEEE Computer Society, 2003.

[34] Ayush Kumar et al. “RadVolViz: An Information Display-Inspired Transfer Function Editor for Multivariate Volume Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 30.8 (2024), pp. 4464–4479. DOI: 10.1109/TVCG.2023.3263856.

[35] M. Levoy. “Display of surfaces from volume data”. In: *IEEE Computer Graphics and Applications* 8.3 (1988), pp. 29–37. DOI: 10.1109/38.511.

[36] M. Levoy. “Display of surfaces from volume data”. In: *IEEE Computer Graphics and Applications* 8.3 (1988), pp. 29–37. DOI: 10.1109/38.511.

[37] Chang Li et al. “SpaceWalker enables interactive gradient exploration for spatial transcriptomics data”. In: *Cell Reports Methods* 3.12 (2023), p. 100645. ISSN: 2667-2375. DOI: https://doi.org/10.1016/j.crmeth.2023.100645. URL: https://www.sciencedirect.com/science/article/pii/S2667237523003168.

[38] Xiaotong Liu and Han-Wei Shen. “Association Analysis for Visual Exploration of Multivariate Scientific Data Sets”. In: *IEEE Transactions on Visualization and Computer Graphics* (Aug. 2015). DOI: 10.1109/TVCG.2015.2467431.

[39] Patric Ljung et al. “State of the Art in Transfer Functions for Direct Volume Rendering”. In: *Computer Graphics Forum* 35.3 (2016), pp. 669–691. DOI: https://doi.org/10.1111/cgf.12934. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12934. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12934.

[40] William E. Lorensen and Harvey E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422. URL: https://doi.org/10.1145/37402.37422.

[41] Kewei Lu and Han-Wei Shen. “Multivariate volumetric data analysis and visualization through bottom-up subspace exploration”. In: *2017 IEEE Pacific Visualization Symposium (PacificVis)*. 2017, pp. 141–150. DOI: 10.1109/PACIFICVIS.2017.8031588.

[42] Aaron Lun. *umappp: C++ port of the UMAP algorithm*. https://github.com/libscran/umappp. Accessed: 2025-07-14. 2023.

[43] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[44] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[45] Yu. A. Malkov and D. A. Yashunin. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. 2018. arXiv: 1603.09320 [cs.DS]. URL: https://arxiv.org/abs/1603.09320.

[46] ManiVaultStudio. *t-SNE-Analysis*. https://github.com/ManiVaultStudio/t-SNE-Analysis. Accessed: 2025-07-14. 2024.

[47] S.R. Marschner and R.J. Lobb. “An evaluation of reconstruction filters for volume rendering”. In: *Proceedings Visualization ’94*. 1994, pp. 100–107. DOI: 10.1109/VISUAL.1994.346331.

[48] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML]. URL: https://arxiv.org/abs/1802.03426.

[49] C. Montani, R. Scateni, and R. Scopigno. “Discretized marching cubes”. In: *Proceedings of the Conference on Visualization ’94*. VIS ’94. Washinton, D.C.: IEEE Computer Society Press, 1994, pp. 281–287. ISBN: 0780325214.

[50] A.V. Oppenheim, R. Schafer, and C. Yuen. “Digital Signal Processing”. In: *Systems, Man and Cybernetics, IEEE Transactions on* 8 (Mar. 1978), pp. 146–146. DOI: 10.1109/TSMC.1978.4309917.

[51] N. Pezzotti et al. “Hierarchical Stochastic Neighbor Embedding”. In: *Computer Graphics Forum* 35.3 (2016), pp. 21–30. DOI: https://doi.org/10.1111/cgf.12878. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12878. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12878.

[52] Nicola Pezzotti et al. “GPGPU Linear Complexity t-SNE Optimization”. In: *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE VAST 2019)* 26.1 (2020), pp. 1172–1181. DOI: 10.1109/TVCG.2019.2934307.

[53] Pavlin G. Poličar, Martin Stražar, and Blaž Zupan. “Embedding to reference t-SNE space addresses batch effects in single-cell classification”. In: *Mach. Learn.* (2021), pp. 721–740.

[54] Andra Popa et al. “Visual Analysis of RIS Data for Endmember Selection”. In: *Eurographics Workshop on Graphics and Cultural Heritage*. Ed. by Federico Ponchio and Ruggero Pintus. The Eurographics Association, 2022. ISBN: 978-3-03868-178-6. DOI: 10.2312/gch.20221233.

[55] Yubo Tao Qirui Wang and Hai Lin. “FeatureNet: automatic visual summarization of major features in multivariate volume data”. In: *Journal of Visualization* 21.3 (2018), pp. 443–455. DOI: 10.1007/s12650-017-0459-x. URL: https://doi.org/10.1007/s12650-017-0459-x.

[56] Bernhard Reitinger, Alexander Bornik, and Reinhard Beichel. “Constructing Smooth Non-Manifold Meshes of Multi-Labeled Volumetric Datasets.” In: Jan. 2005, pp. 227–234.

[57] Christof Rezk Salama. “Visual Parameters and Transfer Functions”. In: Oct. 2008. DOI: 10.1007/978-1-84800-269-2_5.

[58] Penny Rheingans and David Ebert. “Volume Illustration: Nonphotorealistic Rendering of Volume Models.” In: *IEEE Trans. Vis. Comput. Graph.* 7 (Jan. 2001), pp. 253–264.

[59] Rickard Sandberg. “Entering the era of single-cell transcriptomics in biology and medicine”. In: *Nature Methods* 11.1 (2014), pp. 22–24. DOI: 10.1038/nmeth.2764. URL: https://doi.org/10.1038/nmeth.2764.

[60] Mathias Schott et al. “A Directional Occlusion Shading Model for Interactive Direct Volume Rendering”. In: *Computer Graphics Forum* 28.3 (2009), pp. 855–862. DOI: https://doi.org/10.1111/j.1467-8659.2009.01464.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01464.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01464.x.

[61] David Schroeder and Daniel F. Keefe. “Visualization-by-Sketching: An Artist’s Interface for Creating Multivariate Time-Varying Data Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 877–885. DOI: 10.1109/TVCG.2015.2467153.

[62] Will Schroeder et al. *A High-Performance SurfaceNets Discrete Isocontouring Algorithm*. 2024. arXiv: 2401.14906 [cs.GR]. URL: https://arxiv.org/abs/2401.14906.

[63] Tran Van Long and Lars Linsen. “MultiClusterTree: Interactive Visual Exploration of Hierarchical Clusters in Multidimensional Multivariate Data”. In: *Computer Graphics Forum* 28.3 (2009), pp. 823–830. DOI: https://doi.org/10.1111/j.1467-8659.2009.01468.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01468.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01468.x.

[64] A. Vieth et al. “Incorporating Texture Information into Dimensionality Reduction for High-Dimensional Images”. In: *2022 IEEE 15th Pacific Visualization Symposium (PacificVis)*. 2022, pp. 11–20. DOI: 10.1109/PacificVis53943.2022.00010.

[65] Alexander Vieth et al. “ManiVault: A Flexible and Extensible Visual Analytics Framework for High-Dimensional Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 30.1 (2024), pp. 175–185. DOI: 10.1109/TVCG.2023.3326582.

[66] Feiran Wu et al. “EasyXplorer: A Flexible Visual Exploration Approach for Multivariate Spatial Data”. In: *Computer Graphics Forum* 34.7 (2015), pp. 1–11. DOI: 10.1111/cgf.12755. URL: https://doi.org/10.1111/cgf.12755.

[67] Zizhen Yao et al. “A high-resolution transcriptomic and spatial atlas of cell types in the whole mouse brain”. In: *Nature* 624.7991 (2023), pp. 317–332.

[68] Clarence Yapp et al. “Highly Multiplexed 3D Profiling of Cell States and Immune Niches in Human Tumours”. In: *bioRxiv* (2025). DOI: 10.1101/2023.11.10.566670. eprint: https://www.biorxiv.org/content/early/2025/04/11/2023.11.10.566670.full.pdf. URL: https://www.biorxiv.org/content/early/2025/04/11/2023.11.10.566670.

[69] Liang Zhou and Charles Hansen. “GuideME: Slice-guided Semiautomatic Multivariate Exploration of Volumes”. In: *Computer Graphics Forum* 33.3 (2014), pp. 151–160. DOI: 10.1111/cgf.12371. URL: https://doi.org/10.1111/cgf.12371.

[70] Richard Zhu. *Fast Exact Retrieval for Nearest-neighbor Lookup (FERN)*. 2024. arXiv: 2405.04435 [cs.CL]. URL: https://arxiv.org/abs/2405.04435.