

ManiVault: A Flexible and Extensible Visual Analytics Framework for High-Dimensional Data

Alexander Vieth^{*1}, Thomas Kroes^{*2}, Julian Thijssen^{*2}, Baldur van Lew², Jeroen Eggermont², Soumyadeep Basu², Elmar Eisemann¹, Anna Vilanova³, Thomas Höllt^{*1}, Boudewijn Lelieveldt^{*1,2}

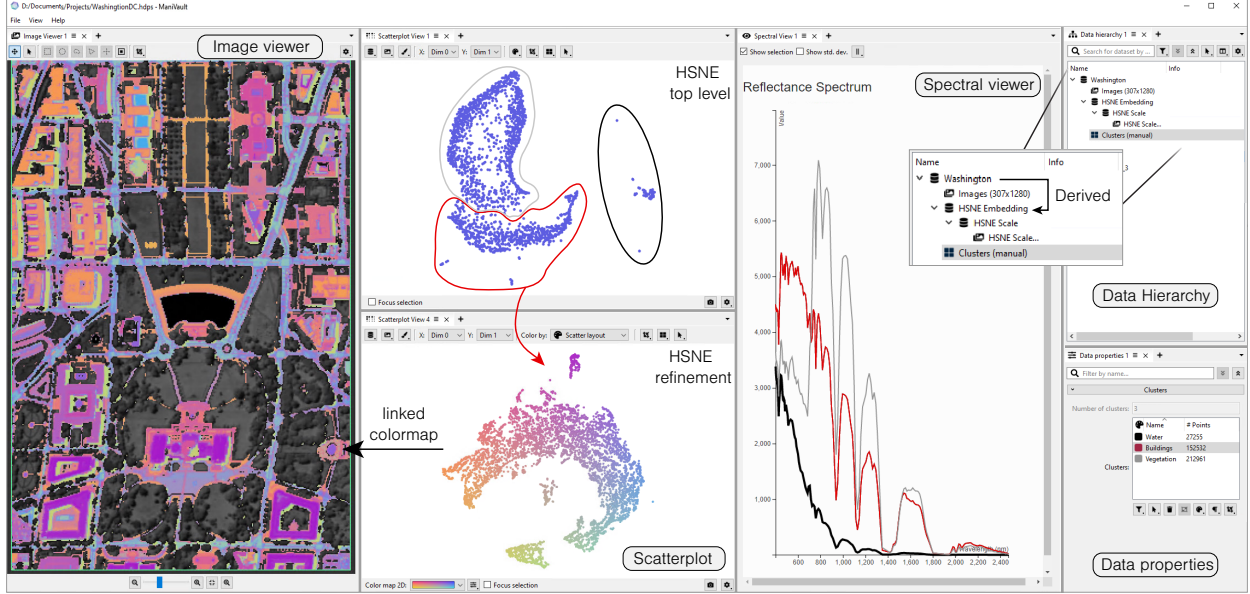


Fig. 1: Example screenshot of ManiVault used for the exploration of a hyperspectral imaging data set.

Abstract—Exploration and analysis of high-dimensional data are important tasks in many fields that produce large and complex data, like the financial sector, systems biology, or cultural heritage. Tailor-made visual analytics software is developed for each specific application, limiting their applicability in other fields. However, as diverse as these fields are, their characteristics and requirements for data analysis are conceptually similar. Many applications share abstract tasks and data types and are often constructed with similar building blocks. Developing such applications, even when based mostly on existing building blocks, requires significant engineering efforts. We developed ManiVault, a flexible and extensible open-source visual analytics framework for analyzing high-dimensional data. The primary objective of ManiVault is to facilitate rapid prototyping of visual analytics workflows for visualization software developers and practitioners alike. ManiVault is built using a plugin-based architecture that offers easy extensibility. While our architecture deliberately keeps plugins self-contained, to guarantee maximum flexibility and re-usability, we have designed and implemented a messaging API for tight integration and linking of modules to support common visual analytics design patterns. We provide several visualization and analytics plugins, and ManiVault's API makes the integration of new plugins easy for developers. ManiVault facilitates the distribution of visualization and analysis pipelines and results for practitioners through saving and reproducing complete application states. As such, ManiVault can be used as a communication tool among researchers to discuss workflows and results.

A copy of this paper and all supplemental material is available at osf.io/9k6jw, and source code at github.com/ManiVaultStudio.

Index Terms—High-dimensional data, Visual analytics, Visualization framework, Progressive analytics, Prototyping system.

1 INTRODUCTION

High-dimensional data has become important and ubiquitous in many applications. Yet, understanding this type of data remains challenging and poses many hurdles ranging from computational efficiency to interpretability. Combinations of automated analysis and interactive vi-

ualizations, visual analytics (VA) [12, 25], have proven to assist well in gaining insight for high-dimensional data. A variety of visual encodings and processing algorithms for high-dimensional data exist. At the same time, specialized application domains require specialized workflows for handling their data and often need to adapt established methods to their use case. Even though these domains encounter different domain-specific questions, they often deal with similar abstract data set types. Additionally, abstracting different domain-specific workflows regularly yields similar goals and user tasks [8, 30] which might be tackled with recurring visual encoding components like heatmaps or analytics methods such as dimensionality reduction. It is time-consuming and wastes development resources to reinvent the wheel by re-implementing, e.g., a linked selection mechanism for multiple coordinated views every time a domain-specific VA solution is needed [29, 34, 39, 57, 69]. We developed a visual analytics framework, ManiVault, as a flexible solution for VA software developers, application designers, and practitioners to im-

^{*•} These authors contributed equally.

¹ TU Delft, E-mail: {A.Vieth | E.Eisemann | T.Höllt-1} @tudelft.nl.

² Leiden University Medical Center, E-mail: {J.G.L.Thijssen | T.Kroes | J.Eggermont | B.van_Lew | S.Basu | B.P.F.Lelieveldt} @lumc.nl.

³ TU Eindhoven, E-mail: A.Vilanova@tue.nl.

Table 1: **Comparison with other visual analysis tools** that are most similar to ManiVault.

	ManiVault	XmdvTool [66]	GGobi [58]	Visplore [65]	Tableau [59]	ParaView [3]	Inviwo [23]
Focus on high-dim. data	•	•	•	•	•	—	—
Focus on field data	—	—	—	—	—	•	•
Extensible	•	• ^a	•	—	—	•	•
Visual Analytics	•	•	• ^b	•	•	• ^c	— ^d
Progressive Analytics	•	—	• ^b	•	—	— ^d	— ^d
VA system authoring	•	—	—	—	• ^d	• ^e	—
Active development	•	—	—	•	•	•	•
License	LGPL-3	Public domain	EPL	Commercial	Commercial	BSD-3	BSD-2

^a No dynamic extension loading ^b When used with its API, e.g., in combination with R ^c Via Trame [27] ^d The systems can be extended with Visual Analytics functionality by plugins or Python integration, but the focus is on interactive field visualization ^e Focus on dashboards with pre-populated data

plement algorithms and visual encodings, prototype workflow-specific tool sets, and perform their data exploration and analysis respectively.

Existing VA systems for exploring general multivariate data do not meet all of these goals. Commercial products like Visplore [40, 41] or Spotfire [1, 2] come with wide feature ranges but are closed-source and not easily extensible. Older open-source frameworks like XmdvTool [66] and GGobi [58] are mostly limited to visual analysis and lack analytics functions. ParaView [3] and Inviwo [23] are capable of displaying multivariate data as well but focus on field data and the representation of spatial structures. Business intelligence solutions like Tableau [56, 59] mostly focus on dashboard creation and chart recommendations. Other fast dashboard prototyping tools, like Keshif [70], provide infrastructure like linked selections of various data visualizations but lack analytics capability. With ManiVault we propose a visual analytics framework for general high-dimensional data that is easily extendable and lets both developers and practitioners re-use algorithmic and visualization building blocks for prototyping and reusing visual analytics systems.

Growing data sizes, both in the number of items and dimensions, increasingly complicate interactive analysis. Progressive visual analytics [55] intends to overcome this issue by continuously providing intermediate results of the current data analysis step. The ability to control the analysis based on continuous feedback is crucial for progressive VA systems [4]. In ManiVault we implement a data-centric and modular framework that facilitates continuous data updates and algorithm steering out of the box. The ManiVault core application manages data sets and plugins, which provide both analysis and visualization functionality. This architecture allows for fast data changes, selection updates, and overall flexible data exploration. Additionally, since each plugin is agnostic of any other, the system is easy to extend with new data types, visualizations, and analysis algorithms. ManiVault is written in C++, using the Qt framework [60] for cross-platform GUI development. OpenGL is used for high-performance rendering (e.g., our scatterplot plugin) but viewer plugins based on lower threshold JavaScript libraries like D3 [7] and Vega-Lite [50] are also possible. ManiVault is open source and can be found at github.com/ManiVaultStudio.

To summarize, in this paper we describe

- ManiVault, a modular and extensible visual analytics framework designed for high-dimensional data,
- several functionality extensions in the form of basic data-, viewer-, and analytics plugins, and
- three use cases ranging from plugin development to a practitioner’s workflow.

2 RELATED WORK

Visual analysis of high- and multidimensional data is broadly discussed in literature [17, 24, 68]. Here, we review the most relevant work on Visual Analytics (VA) systems for multidimensional data and visualization design environments with respect to our framework.

2.1 Visual Analysis and Analytics Systems

VA systems for the exploration and analysis of high-dimensional data are well established both in academia and industry [14, 19]. Table 1 gives an overview comparison between ManiVault and visual analysis tools that we deem most similar. Most VA systems employ coordinated

multiple views [47] with linked selections for data exploration, and we follow this approach with ManiVault as well. Chen et al. [10] discuss common practices and guidelines for the layout of multiple views.

Pioneering visual analysis frameworks for *multidimensional data* include XmdvTool [66], Spotfire [1], GGobi [58] and the InfoVis toolkit [16]. These frameworks mostly focused on displaying data with a variety of visual idioms and enabled exploration with brushing tools and linked selections. XmdvTool was extended with several dimensionality reduction and clustering methods [13, 71, 72]. GGobi [58] integrates with the R language which enables users to apply analysis algorithms via scripting. Spotfire grew into a commercial, closed-source product with extensive analytics capabilities, while the others are open-source, albeit unmaintained. All of these tools predate *Progressive VA* and are not optimized for the specific needs of continuous updates and steering of analytics processes. ManiVault is designed around the principles of progressive VA from the start using a data-centric architecture. Data-producing and -transforming plugins can continuously update the data managed by the core, while data consumers get automatically notified about these changes. Tableau [59], building on the Polaris system [56], might be the most prominent and representative universal VA system. Marketing itself as a business intelligence tool, Tableau focuses on flexible visualization of various data types and more general analytics functions can be added via Python or R scripts. Similarly, Visplore [40, 41] implements a suit of statistical analysis and visualization methods for tabular data and aims at providing quick visual feedback for visual interactions and data queries. Its commercial offspring [65] offers a more direct integration of scripting languages to supplement built-in analysis functions.

The open-source ParaView [3], like many other analysis frameworks for spatial field data, e.g., volume data, [6, 11, 48, 52] is based on the VTK library [51], and provides a wide range of visualization and analysis functions in an extensible framework. ParaView follows VTK’s visualization pipeline and is designed around the flow of data through various transformations to their final visual presentation. Similarly, the commercial Amira Software [54, 61] offers a range of analysis functions for multidimensional volumetric data but it is not freely extensible. Many visual analysis systems traditionally target either geometric or abstract tabular data. However, in recent years, the analysis of spatial and non-spatial data has become increasingly integrated [53]. With ManiVault we create a system for general high-dimensional data that can be extended to handle arbitrary spatial or abstract data types. Our data-centric system design enables flexible exploration workflows instead of having practitioners concerned about data flow through each step of the visualization pipeline.

2.2 Visualization Design Environments

Visualization design environments or similarly visualization prototyping systems are tools for creating visualizations that provide a graphical user interface for specifying visual encodings of data and interaction dynamics. Many such systems exist, and here we provide an overview of the tools most similar to ManiVault.

Lyra [49] offers fine-grained design options for single plots through handles, drop-zones, and other interaction mechanisms for graphical setup of re-usable Vega or Vega-Lite [50] specifications. Lyra 2 [73] extends this framework by letting users define interactions like brushing and selection linkage between multiple plots. iVisDesigner [46]

follows similar principles but places emphasis on collections of data visualizations in a dashboard format. Keshif [70] focuses on a novice user audience by automatically aggregating data and selecting visual representations based on pre-defined mappings for various data types. In contrast to the above design environments for single or multiple visualizations, ManiVault is a design environment for complete visual analytics systems including automated analysis methods. While the above systems are focused on abstract data, Inviwo [23] presents a visualization prototyping system for spatial field data. Its design allows users to specify visualizations on various abstraction levels, from visual (connecting functional boxes) to conventional programming. Compared to Inviwo’s data-flow model, ManiVault is data-centric and focused on providing several visualizations and analytics tool building blocks. ManiVault’s core system coordinates views on the data and enables linked selections between views out-of-the-box.

From a plugin-in developer’s perspective, ManiVault resembles the *prefuse* [20] and *ComVis* [35] toolkits. They provide development environments and software components for building dynamic visualizations. Both focus on non-spatial data and target graph and tabular data set types. Scripting-based solutions like *Dash* [43] for creating dashboard applications or *Voilà* [28] for converting Jupyter notebooks into standalone web pages provide a GUI front-end to the wide offer of analysis libraries in the Python, R or Julia ecosystems. ManiVault is specifically laid out for progressive and high-dimensional data analysis. Our C++ implementation supports high-performance computations and interactions necessary for visual analytics.

3 DESIGN CONSIDERATIONS

We designed ManiVault as a VA framework with multiple user groups in mind. While these groups can overlap, their requirements for the effective and convenient use of ManiVault are varied.

3.1 General Setting

High-dimensional data has become ubiquitous in many domains and the analysis of such data plays a pivotal role in acquiring insights into complex systems. Analytics software in different domains targeted at such data generally utilizes comparable sets of analytical and visual tools, such as dimensionality reduction, clustering algorithms, scatterplots, or parallel coordinates plots. These generic tools are then combined with data-, user-, and domain-specific tools and customizations to create a specific application. The primary motivation for developing ManiVault is to facilitate rapid construction of visual analytics applications for high-dimensional data without the need to re-implement common functionality. Modularity is a key aspect for creating reusable tools, both on a code as well as a user-facing abstraction level. The second main motivation for ManiVault is a need for flexible exploratory analysis, but also subsequent sharing of results, as well as the means to recreate the corresponding workflows. We learned of the target user characteristics and design requirements during multiple collaborations with practitioners in various fields [21, 33, 44, 62] spanning several years.

3.2 Target Users

We identified three target user groups, each with specific requirements:

U1 Developers use ManiVault to implement new ideas and methods. These users, e.g., visualization researchers, interact with the system via code in order to create customized modules. Developers need the framework to provide a stable API that allows for the integration of their methods with little overhead. Further, they need existing modules to focus on their specific contribution; e.g., a developer of a dimensionality reduction method might want to visualize results in an existing scatterplot module without having to implement their own.

U2 Application designers combine and adapt existing modules to create stand-alone applications for specific use-cases. Not all options of a view (e.g., the point size in a scatterplot) might be necessary for a specific workflow, and providing all options in the GUI can be distracting. In these scenarios, ManiVault needs to support flexible GUI customization. To minimize the burden, the framework should support such customization directly in the GUI without programming.

U3 Practitioners and domain experts use the software to analyze their high-dimensional data. Practitioners need ManiVault to allow for a flexible data exploration process, to provide responsive user interfaces, and to offer domain-specific visualization and analysis modules. Once their analysis is finished, practitioners need the ability to easily share and reproduce the results and their workflow in ManiVault. Given a well-defined workflow, they also need easy access to specified presets of visualization and analysis layouts.

The boundaries between these user groups are fluid. E.g., a skilled practitioner might want to extend a pre-bundled application with a module or develop a module themselves.

3.3 System Requirements

Based on the general usage setting and needs of our target users, we define the following high-level requirements for a visual analytics platform such as ManiVault. The framework must be:

R1 Extensible: ManiVault has to provide an interface for adding new functionalities. It must be possible to create modules for new

- a data types,
- b visualizations,
- c analytics methods,
- d data transformations,
- e loading/writing data.

R2 Flexible: ManiVault must allow for workflows in multiple domains and specifically enable straightforward workflow adaption during use.

R3 Linkable: ManiVault must provide modules with an API to easily link data selections and synchronize parameters, such that no dependencies between modules are created.

R4 Configurable: ManiVault must provide options for GUI configuration during runtime through the user interface.

R5 Distributable: ManiVault must be able to save its current state, including layout, data sets, and settings and reproduce a saved state.

R6 Performant: ManiVault must be performant when handling large data, stay responsive and provide interfaces to interact with processes during calculation to support progressive VA.

4 MANI Vault ARCHITECTURE

In order to ensure easy extensibility (**R1**), ManiVault is implemented as a modular system, see Fig. 2a. The core application is a lightweight set of managers and any user-facing functionality is dynamically loaded from self-contained libraries, i.e., plugins, respectively discussed in Secs. 4.1 and 4.2 (**R6**). This compartmentalization into a core and extensions provides easier maintainability, better scalability, and faster development. Together with a data-centric system structure (Sec. 4.3), this enables flexible workflows (**R2**) with various analytics and visualization techniques. ManiVault features an intricate notification and parameter sharing system to allow for communicating between plugins, see Sec. 4.4 (**R3**). GUI management objects, called actions (Sec. 4.5), implement a part of the communication system and the configuration and serialization system, see Secs. 4.6 and 4.7 (**R4**, **R5**).

4.1 Core Application

ManiVault’s core is modularized into a set of managers, actions, and utilities as shown in Fig. 2a. ManiVault comprises a data-centric architecture: a data manager stores and administers access to data sets. All data sets are organized hierarchically, such that derived data sets like clusterings, embeddings, or proper subsets are marked as children of their respective source data. This enables simple access to properties of the parent data set and propagation of selections from derived to source data sets. Analysis, transformation, visualization, and loading/writing functionality as well as the definition of data types themselves are separated into plugins. A plugin manager loads plugins into the core and makes them available to the user. Each plugin can *consume data*, i.e., process existing data in the core and/or *produce data*, i.e., store a new or alter an existing data set in the core. While each plugin is self-contained, communication between plugins is made possible

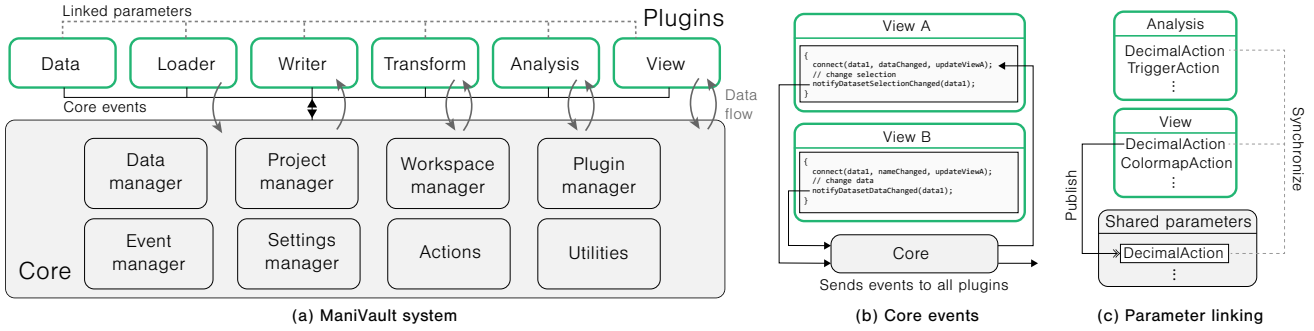


Fig. 2: **ManiVault’s system architecture.** The core manages data and events, provides GUI management (actions), etc. Green ● borders indicate plugins, a light-grey background the core. Data flow from the core to data consumer plugins and from data producer plugins to the core is indicated with \rightarrow arrows. (b) View A listens to `notifyDatasetDataChanged` emitted by View B. View B does not listen to the `notifyDatasetSelectionChanged` event triggered by View A, but any plugin could. (c) a view plugin published a `DecimalAction`, moving the action in a shared parameters space and immediately subscribes to it. Now, an analytics plugin can connect to the shared action, enabling synchronization across plugins.

using two messaging systems (Sec. 4.4). An event manager in the core administers globally defines notifications while actions are used for run-time configurable notifications (see Figs. 2b and 2c).

The general application layout is handled by a workspace manager which takes care of the arrangement of all GUI widgets provided by view plugins. The core contains two main system view plugins, a data hierarchy, and a data properties viewer. The former displays the internal hierarchical data structure, while the latter shows properties of the data (number of data points, dimensions, active selections) and gives users access to the settings of analytics plugins, as discussed in more detail in Sec. 5. ManiVault provides a number of actions, GUI management objects, and administers any user-defined linking between them, see Sec. 4.5. Further, a project manager is responsible for saving and loading the current state of the application, including loaded data sets, the GUI layout, opened plugins, and linked parameters. Global settings applicable to, e.g., all plugins or the general application layout are handled by a dedicated settings manager.

Additionally, ManiVault’s core supplies a set of utilities like dedicated renderers, shaders, color maps, mathematical helper classes, such as vectors and matrices, as well as common algorithms like mean shift clustering. These tools can be used to create a more coherent visualization and analysis setup across plugins. E.g., developers can rely on the availability of a standard set of color map types in every view plugin, while maintaining the ability to introduce custom ones.

4.2 Plugin Types

ManiVault works with six distinct plugin types that bundle various types of functionality. The system can be easily extended with new functionality by writing a new plugin that will automatically be loaded on start-up (R1). In combination with the data-centric core architecture, this enables a user to perform flexible workflow changes (R2).

Data plugins enable extending the types of data the system can handle. ManiVault provides a base data plugin class that developers can extend to define a custom data format. E.g., we provide an image data type that extends our basic point data type with image dimensions and thus a mapping of points to image coordinates. The system can generally be extended with arbitrary data formats.

View plugins provide a view on the data and allow interaction, such as selection of data elements. Views can be fully-fledged visualizations or simpler views such as lists. View plugins are primarily *data consumers*, i.e., they take a data set as input for visualization, but can also function as *data producers*, e.g., by providing means for annotating data. We provide example plugins with diverse backends, like OpenGL and D3.

Analytics plugins allow for the implementation of data analytics modules such as dimensionality reduction. As such, they are primarily *data producers* but also follow the *data consumer* API to receive the input data on which they perform calculations.

Transformation plugins resemble analytics plugins in code but are semantically different. They are also primarily *data producers*, but while analytics plugins derive new properties, e.g., an embedding, that

can have an arbitrary shape, transformation plugins produce data of the same shape, i.e., with identical items and attributes. An example of such a transformation is a normalization of the original data.

Loader/Writer plugins respectively load specific types of data into the system (*data producer*) or write it back to file (*data consumer*).

4.3 Data Handling

The data handling in ManiVault follows a model-view pattern. Internally, the core’s data manager keeps a list of raw data models, data set views, and selection views. A data plugin has to define both a raw data model and data set view — the selection view is simply another instance of the same data set view on the raw data. The raw data model holds the physical data values of a set and is never exposed directly to non-data plugins. Therefore, for most intents and purposes, the data set views can be regarded as the actual data sets present in the system. They define access to the raw data for all non-data plugins by providing, e.g., views on or copies of it. Each raw data object is associated with exactly one selection object to ensure straightforward selection sharing across all plugins that access a data set. Selection and set views can be separately requested and adjusted. This model-view pattern allows for a simple API and to create and use subsets with minimal overhead.

New data sets can be marked as derived from existing ones, e.g., when a new data set is created by an analytics plugin. The derived data also functions as the user-facing entry point through which the analytics settings can be accessed. This operation will create new data set and raw data objects but no new selection view. Instead, selection views are shared between parent and derived data sets. This simplifies the propagation of selections between views, e.g., a derived embedding shown in a scatterplot and the original data in a parallel coordinates plot. To enable selection sharing between arbitrary data sets, ManiVault lets users group data sets in the hierarchy view. Selections of any data sets within a group and with the same number of data points are then automatically synchronized.

We implemented a set of base data plugins in ManiVault, including plugins for point data, multichannel images, clusters, color, and text data. The development of ManiVault so far primarily targeted the point data type, which can store various high-dimensional integer and floating point formats. Our image data plugin shows the versatility of ManiVault’s data handling and the point data type. When loading an image, two data sets are created: a point data set whose raw data object stores the actual pixel values and a child image data set whose raw data object stores metadata like image size. The image data set view provides access to the parent’s raw data. This configuration ensures compatibility with analytics, transformation, and view plugins that expect point data to process multichannel images.

The implemented data handling system is lightweight. Besides the basic ManiVault core (< 90 MB), the data manager and hierarchy require < 8 MB of memory (on Windows). Each loaded data set produces less than 1.5 MB overhead in addition to its binary size, stemming from the plugin instance and core integration. More details can be found in Supplemental Material S1.

4.4 Plugin Communication

Coordinated Multiple Views (CMVs) [47] are the basis for virtually any visual analytics application. While the individual views in a CMV system naturally map to modules in a modular architecture, an essential part of CMV systems is the integration of those views. This enables techniques like brushing and linking [9], where selections on the data are propagated to all views in the system, or the synchronization of parameters, like the viewport in an Overview+Detail system [42]. Enabling such linking of views, without breaking the system's modularity (R3) is no trivial task. A plugin should be self-contained with respect to its functionality. Yet, at the same time, plugins need to be able to communicate, such that they can inform other plugins about data changes and that their parameters can be linked and synchronized throughout the application.

We have designed and implemented two interfaces to solve the issue of inter-plugin communication. First, an event-based communication API to cover common system-wide types of events related to data set changes (Sec. 4.4.1) and second a parameter-sharing API (Sec. 4.4.2) as part of our GUI building blocks (Sec. 4.5).

4.4.1 Core Events

The ManiVault core API provides an event-based system for inter-plugin communication using the **publish-subscriber** pattern. Plugins send predefined events to the core, which distributes them, and all subscribers (typically plugins) can digest these events as depicted in Fig. 2b. To efficiently support linking and brushing (R3), we have implemented such events for any changes of data values like addition (`notifyDatasetAdded`), updates (`notifyDatasetDataChanged`), removal (`notifyDatasetRemoved`), changes to data selections (`notifyDatasetSelectionChanged`) and several other data related changes. A plugin can choose to listen to all events of a certain type or subscribe only to certain events concerning a specific data set.

An example of a linked selection is shown in Fig. 3. The figure shows a screenshot with three views, a scatterplot and a density plot on the left, and the properties of a clustering analysis on the right. Clicking any cluster in the clusters list (Fig. 3a) will update the selection set attached to the data set and notify the core of these changes with the `notifyDatasetDataSelectionChanged` event. The core will then emit the `dataSelectionChanged` event with the changed data as an argument and subscribed plugins will receive a notification that triggers a refresh of the view with the updated selection (red points in Fig. 3b).

4.4.2 Shared Parameters

We designed a complementary API to share parameters between modules (R3) using GUI actions (Sec. 4.5). With this system, a plugin parameter is exposed to other plugins by placing it in a public shared parameter pool, i.e., the parameter is *published* (Fig. 2c). From there, other plugins can *subscribe* to published parameters (provided that the parameter types match). Any change to a published parameter will be synchronized with all subscribed parameters. We provide common GUI elements with ManiVault, that developers can integrate into their plugins such that the user can publish a parameter or subscribe to any published parameter at run-time through the GUI (R4).

Figure 3 presents an example in the form of the kernel bandwidth (sigma) parameter used in kernel density estimation (KDE) employed in density plot visualizations (Fig. 3c) but also mean-shift clustering. We have implemented plugins for both that allow real-time changes of the sigma parameter, based on Lampe and Hausers real-time KDE [31]. Linking this parameter between the density plot and the clustering module enables visually finding a suitable density estimation while the clustering is updated on-the-fly. To link the parameters the user simply clicks on the underlined label in the GUI (Fig. 3d), e.g., in the density plot view, and chooses "publish". After defining a suitable name for the parameter, the user can then click on the corresponding label in the settings widget of the mean shift clustering plugin (Fig. 3e) and click subscribe to be presented with a list of suitable parameters, including the just defined one. After subscribing, the connection is indicated by the italic font of the *Sigma* label.

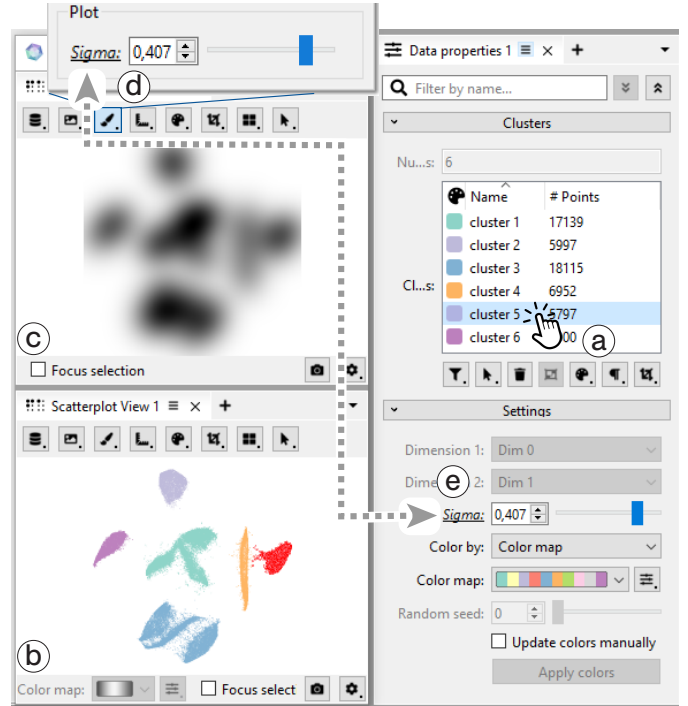


Fig. 3: **Parameter sharing** by connecting two actions of the same type in the GUI. Both, the Mean-Shift plugin and Scatterplot plugin use a `DecimalAction` to steer their computation and view respectively.

4.5 Actions

To support sharing of parameters as described above, but also to make it easy to capture the state of a plugin, configure the GUI and unify the look and feel between plugins, we have devised and implemented a number of building blocks we call *actions* on top of the standard Qt GUI widgets. These include simple actions for decimal and integral values as well as strings but also more complex elements such as colors, color maps, file-pickers, etc.. In addition to those standard GUI elements we implemented a number of custom actions targeting typical VA applications. These include a general-purpose selection action, that supports different modalities (brushing, rectangle, lasso, etc.) and Boolean combinations (replace, add, remove), and a dimension picker action that provides a consistent way to select one or multiple dimensions of a data set, e.g., to limit the input to a dimensionality reduction plugin. Although we believe that we provide large coverage of commonly required tasks with the built-in actions, we also provide an API for plugin developers to create custom actions.

By using our actions API, sharing of parameters as described in Sec. 4.4 is automatically available through the GUI. In addition, actions can also be attached to data objects, to expose their functionality to other plugins. A data producer plugin can, e.g., attach an action to trigger a calculation within the plugin. Other plugins can query these attached actions and provide the corresponding GUI elements within their scope. We showcase this in our Hierarchical Stochastic Neighbor Embedding (HSNE) [37] analytics plugin. The plugin creates a hierarchical embedding structure that can be refined interactively. We attach an action for triggering the refinement to the produced embedding data set. When viewing the embedding in a scatterplot, the scatterplot view plugin exposes the refine action and other attached actions through the context menu. The user can then trigger the refinement directly from the scatterplot visualization, even though the actual calculation is carried out by the HSNE plugin.

Besides serving as GUI building blocks, we have also implemented support for serialization in the action system. Each action can be serialized into a `QVariant` object, including its complete current state, consisting of whether it is active, visible, writable, and the parameter itself. All actions that belong to a plugin form a hierarchy that can again be serialized into a `QVariant` object and from there into a JSON

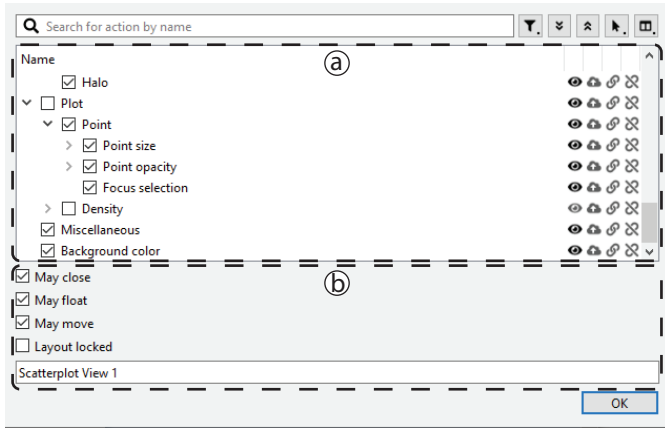


Fig. 4: Example of the plugin **GUI configuration editor** which allows application designers to edit the properties of the plugin actions hierarchy from within the application.

object in memory or file on disk. As such, a plugin that has consistently been implemented with the actions API supports saving and loading of the state out-of-the-box. Currently, we use this to create presets of a plugin’s configuration and to save the complete state of the application to a project file. In the future, we intend to extend this to a complete provenance mechanism.

An example of a simple decimal action is the implementation of the Sigma parameter discussed above and shown in Fig. 3d. The GUI for this parameter consists of the label, a spinbox, and a slider. Rather than manually creating the GUI elements, the desired elements can be specified when creating the action. An example of a customization that we integrated in the decimal action is to show a spinbox or slider individually or both, as in this example. The action then creates the GUI elements on-the-fly and also makes sure they are synchronized by creating them as linked views on the parameter itself. The underlined label indicates that the parameter is publishable and/or ready to subscribe, while the italics font indicates that it is already linked. Clicking the label opens a GUI interface for setting up parameter linking.





4.6 Projects and Workspaces

To save the entire state of the application and fully restore it at a later point in time ManiVault uses projects (R5). Projects extend the serialization of actions, described in Sec. 4.5, to the core framework, capturing settings and the layout of the CMV system. In addition, a project contains a complete snapshot of the data hierarchy. We implemented projects as self-contained, compressed archives that are a combination of human-readable JSON files and binary files. Two JSON files are used to save the entire state of the application. A `workspace.json` contains the CMV layout and actions state and a `project.json` saves the data hierarchy and additional project metadata. The actual data sets are saved as raw binary blobs, with unique identifiers referenced in `project.json`, to minimize load and save times. As such, a project is completely self-contained and can be easily distributed to share findings or simply used to come back to an analysis at a later point in time.

We split the description of the project into `project.json` and `workspace.json` to add an additional feature, i.e., the definition of user-defined workspaces. As described above, the workspace contains the complete spatial arrangement of views (layout configuration) and their complete state. A workspace is used to set up a complete tailor-made CMV VA application, including customized GUI elements, but without preset data, as a project would. To enable easy tailoring of layouts and cross-plugin connections directly in the application, even without programming, we designed the *Studio Mode* for ManiVault.

4.7 Studio Mode

For the configuration of actions, workspaces, and complete projects, ManiVault can be put into *Studio Mode*. This mode of operation allows application designers to create complete tailor-made applications and data viewers from within the GUI of ManiVault itself.

A plugin editor, shown in Fig. 4, enables fine-grained control over the user interface. It lists an overview of all actions that are currently available for opened plugins (Fig. 4a). Therein each action can be enabled or disabled as a whole ☒, but also customized with respect to its visibility  or whether it can be published , connected , or disconnected . Additionally, the editor lets a user configure general options like the name of a plugin instance, shown in its title bar, or whether the GUI of the plugin may be moved or closed (Fig. 4b).

The plugin editor is an essential tool for application designers, to create a completely customized user experience for a specific application. At the same time, it provides the possibility for advanced users of the system to create presets of views. Besides saving a complete project, users can adjust the interface of an individual plugin to their needs and save the resulting configuration as a template for future instances of that plugin. Using the serialization described above, these templates can be saved to disk, providing persistent access across sessions.

For a user-definable flexible layout of the application, we incorporate the Qt advanced docking system [22] into ManiVault. The system allows users and application designers to re-arrange the entire layout according to their needs and preferences.

5 MANI Vault IMPLEMENTATION

The ManiVault core is implemented in C++ and the Qt [60] cross-platform application development framework. ManiVault provides a plugin API for data types, view, analytics, transformation, and writer/loader modules. For each of these types we provide template implementations to lower the entry barrier for developers. In addition, we have already implemented a number of plugins for various use cases, including some of the core functionality of ManiVault such as the basic data types, and the data hierarchy and data properties view plugins.

The **data hierarchy view** (Fig. 5a) functions as the central access point to any data loaded or created in ManiVault. It displays the data hierarchy in a searchable tree widget where derived data, such as a clustering, are added as children to the original data. A data set can be loaded into a viewer plugin by simply dragging it from the hierarchy onto the view (Fig. 5c). Alternatively, the user can also interact with each data set through a context menu providing access to all compatible data consumer plugins. For a fast setup of plugins that expect more than a single input, users can select multiple data sets in the hierarchy and open them through the same menu. The info panel shows additional information like an analytics progress bar, status messages from plugins or data group affiliation. If a data set is associated with an analytics plugin, selecting the hierarchy entry will open the analytics settings in the properties view.

The **data properties view** (Fig. 5b) provides information for a data set selected in the data hierarchy. For a loaded data set this can be additional metadata created by the loader, e.g., the extents of an image data set. More importantly, the data properties view also functions as the user interface for analytics and transformation plugins. These

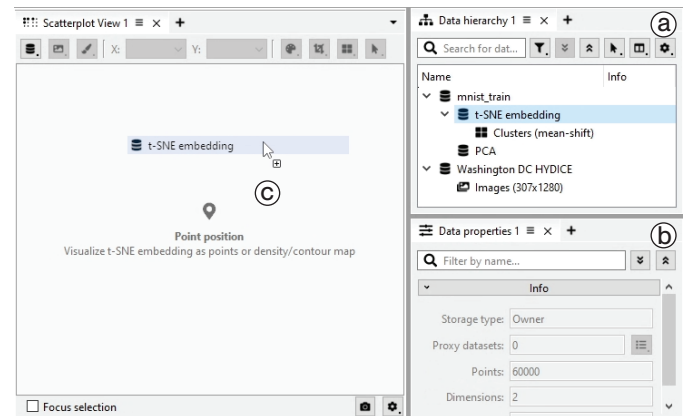


Fig. 5: **Data hierarchy** (a) and **data properties** view (b) in ManiVault. Data sets can easily be shown in views via drag and drop (c).

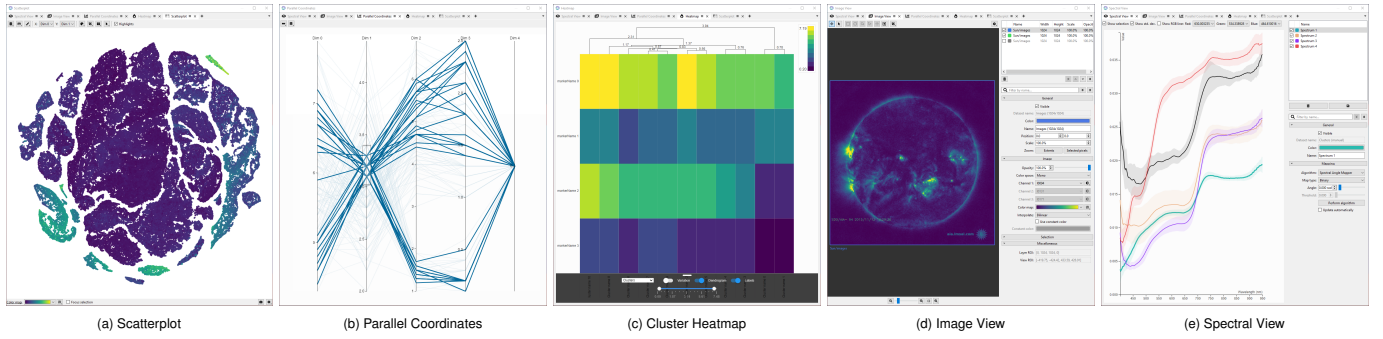


Fig. 6: A selection of **viewer plugins** in ManiVault.

plugins are instantiated through the context menu of a data set, which then functions as their input; their output data sets are then created as children of the input. Selecting an output data set provides access to the parameters of the analytics or transformation plugin. Fig. 5b shows the data properties view of an embedding data set, created with our t-SNE plugin. From here, the user can at any time interact with the t-SNE algorithm, e.g., to pause the calculation, change parameters or compute more iterations.

The data hierarchy and data properties views are integral parts of the system. More specific functionality is implemented in a number of further plugins. Dimensionality reduction, integral to high-dimensional data analysis, is provided by Principal Component Analysis (PCA), t-distributed Stochastic Neighbor Embedding (t-SNE) [63], and Hierarchical Stochastic Neighbor Embedding (HSNE) [37] plugins. The t-SNE and HSNE plugins wrap the high-performance HDI library [36] and as such scale to millions of data points using its GPU-based implementations [38]. For clustering, we provide an interactive **mean-shift clustering** plugin, based on real-time kernel density estimation [31].

For visualization, we provide a number of plugins for common plots, including a **scatterplot** (Fig. 6a), **parallel coordinates plot** (Fig. 6b), and **cluster heatmap** (Fig. 6c). If performance is not a major concern, developers can use web views in combination with Qt’s **webchannel API** for communicating between the C++ back-end and web-technology-based front-end. This allows for easily integrating the vast amount of available visualizations in languages like D3 [7] and Vega-lite [50]. Our heatmap and parallel coordinates plot are based on this technology. While the webchannel introduces some overhead, such plugins are generally limited by the performance of the JavaScript rendering libraries. If the scalability of a visualization is of high priority, developers can implement custom high-performance views, e.g., using OpenGL. We have done so with our scatterplot and image view (Sec. 5.1) plugins. The scatterplot enables visualization and interaction with millions of points in real-time. In the default point rendering mode, the different visual channels (point size, color, opacity, etc.) are fully configurable either using fixed values or based on any fitting data available. Additionally, we implemented a density representation, to provide more visual scalability.

Finally, for data loading and writing, we currently provide support for basic formats in the form of a comma-separated value (CSV) loader/writer and a **binary** loader/writer.

5.1 High-Dimensional Imaging

Besides traditional abstract high-dimensional data analytics, we target a number of applications related to high-dimensional imaging (e.g., the workflow presented in Sec. 6.2). As such, we developed a number of plugins targeting such image data.

Central to these efforts is the **image data type** plugin. The image data type extends the point data type by the extent of the image. Consequently, the image data type is compatible with all data consumer plugins that take point data as input; e.g., this allows to calculate a t-SNE using the pixels of a high-dimensional image as input.

We implemented a sophisticated **image view** plugin (Fig. 6d). Inspired by widely used image editors, we opted for a layer-based approach. Users can simply drag multiple data sets into the view, where they are added as layers. From here, users can define the transparency,

as well as the position of each layer, e.g., to stack multiple properties of a single data set as semi-transparent layers or arrange complementing data sets next to each other. These interactions are possible through standard navigation tools for zooming and panning, while selection is implemented using the action described in Sec. 4.5. The actual visualization of the image is fully configurable: One or two attributes can be displayed by using 1D and 2D color mapping, and three attributes by directly mapping them to the three channels of *RGB*, *HSL*, or *CIELAB* color spaces.

Next to the image viewer, we also provide a **spectral view** plugin (Fig. 6e), specifically for hyperspectral images. The viewer is based on a simple D3 line plot and shows spectra of individual pixels or, in the case of groups (e.g., selections or clusters), a mean spectrum and a variation as a band around it.

To load image data into ManiVault, we currently provide two options. The first one is a versatile general **image loader** plugin. Hyperspectral image data is commonly available as a stack of grayscale images, where each image represents a specific wavelength, also interpreted as a dimension of a high-dimensional space. Our image loader detects such stack in a folder containing common image formats (including .png, .jpg, .tiff), and also allows direct loading of other common image formats (grayscale, RGB, ARGB). Dimensions can be interactively included or excluded from the data set in the loading menu. We also support re-sampling of the data before loading and the creation of image pyramids to enable analysis at varying levels of detail, depending on the features of interest or time available for the analysis. Specific to hyperspectral images, we also provide an **ENVI loader** plugin compatible with L3Harris’ geospatial analysis software ENVI [67].

6 APPLICATION EXAMPLES

ManiVault has already been used for several projects across four universities and several partners. Popa et al. [44] and Li et al. [33] describe the design of complete VA systems for analysis of cultural heritage and biological data, respectively. Vieth et al. [64] and Thijssen et al. [62] developed VA approaches for dimensionality reduction and explaining projections as ManiVault plugins. Here, we walk through exemplary usage scenarios for our framework from the perspective of our three target user groups (Sec. 3.2): software developers (Sec. 6.1), practitioners (Sec. 6.2) and application designers (Sec. 6.3).

6.1 Writing ManiVault Plugins – Developer Perspective

ManiVault provides developers of VA modules with a comprehensive API for data set access, the event notification system, and the other core managers (Sec. 4.1). Extending the functionality of ManiVault through new plugins thus comes with minimal overhead. Example code for each plugin type is available at github.com/ManiVaultStudio/ExamplePlugins.

Here, we present two examples of the necessary steps for creating basic plugins (R1). First, we create an analytics plugin based on the high-performance t-SNE library HDI [36]. In addition, we discuss the implementation of a parallel coordinates plot (PCP) plugin using an existing D3 implementation. Together with the existing image viewer and scatterplot, these plugins combine into a complete GUI-based application shown in Fig. 7 that is usable by domain expert users without programming knowledge.

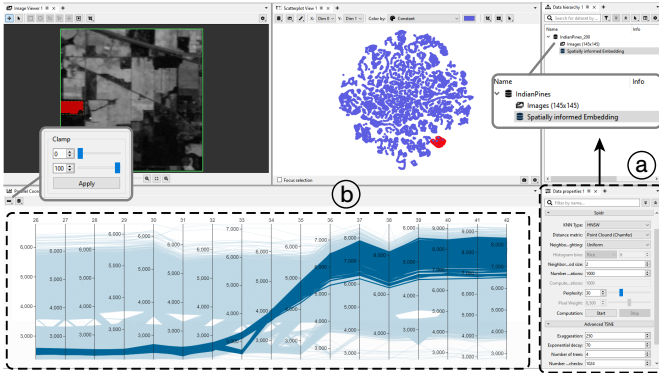


Fig. 7: The **Spidr analysis and parallel coordinates plot** as implemented with the plugin setups from Figs. 8 and 9.

To implement the analytics plugin, we follow the steps laid out in Fig. 8. In step 1, we create the output data set by deriving a new data set from the input data, for which the plugin is opened in ManiVault. In this case, we will create a two-dimensional t-SNE embedding containing x- and y-coordinates for all of the points in the input data set. As such, the output data set will be a points data set that has the same number of points and two dimensions. Next, we add a settings action to the created data set and define GUI elements using ManiVault's action system. The actions are added to the output data and listed in the data properties view as shown in Fig. 7a (step 2). We create TriggerActions which add pushbuttons to the GUI, to start, pause, and resume the calculations and a number of categorical OptionActions and numerical DecimalActions, e.g., to expose t-SNE parameters like the distance metric (OptionAction) or perplexity (DecimalAction) (R4). Finally, in step 3, calls and reactions to library functions need to be defined. Here, we notify the core and thereby other plugins about updated output data, in particular, as the t-SNE optimization iteratively progresses, we notify the core after every iteration, such that the viewer plugins can show the progress live. The result is a lightweight wrapper with no notable performance overhead. Comparing the performance to running the HDI library using its own Python wrapper showed no performance regression (Supplemental Material S1), even when including progressive updates in ManiVault.

To implement the PCP viewer plugin, we need to set up a view widget that shows the PCP chart in addition to settings, like with the analytics plugin. Here, the settings are displayed in the same windows as the view widget (Fig. 7b). Since we build a JavaScript-driven plot, we derive this widget from ManiVault::WebWidget and introduce all HTML and JavaScript resources that are used for the PCP through a Qt resource file, pcp.qrc (step 1, Fig. 9). Step 2 is to simply set the existing pcp.html file in the existing viewWidget. All JavaScript resources are automatically included through the HTML file. At this point, the viewer is only able to show the content of the provided HTML page. To establish interactions to and from the C++ side, we set up a ManiVault::WebCommunicationObject, which uses a QWebChannel. Within this communication object, we define

```
void AnalyticsPlugin::init() {
    // 1. Derive output from input data set
    setOutputDataset(_core->createDerivedDataset("outData"));
    // 2. Add settings actions to output data set
    outDataset->addAction(_settings->getSettings());
    // 3. Connect GUI interactions (e.g. button press)
    // and library callbacks (e.g. progress or finish)
    connect(_settings->getStart(), press, this, runTask);
    connect(_lib, finishedTask, this, updateCore);
}
```

Fig. 8: Bare bone **analytics plugin setup** for wrapping a C++ library. Notifying of output data change (step 4) can be called progressively during the calculation of or on finishing a task.

```
[ViewWidget.cpp]
ViewWidget::ViewWidget() : WebWidget() {
    // 1. Init resources and communication bridge
    Q_INIT_RESOURCE(pcp);
    init(_comObj);
}

[ViewPlugin.cpp]
void ViewPlugin::init() {
    // 2. Init web widget (set HTML contents)
    viewWidget->setPage("res/pcp.html", "qrc:/res/");
    layout->addWidget(viewWidget);
}

[CommunicationObject.h]
class ComObj : public WebCommunicationObject {
    // 3. Init signals for communication from cpp to js
    signals:
        void setData(QVariantList& data);
    // 5. Init slots for communication from js to cpp
    public slots:
        void updateSelection(QVariantList& selection);
}

[qwebchannel.tools.js]
// 4. Register signals sent by the view widget
bridge.setData.connect(function(){initPlot(arguments[0])})
```

Fig. 9: Bare bone **viewer plugin setup** for wrapping a JavaScript library. Some boilerplate code is left out for brevity; complete implementation is available alongside other example plugins online.

signals and slots for communication. E.g., the setData signal (step 3) is used to send the data, provided as a QVariantList object, to a receiver on the JavaScript side. This receiver, i.e., the initPlot function is connected in step 4 to receive the signal. Vice versa, slots defined in the communication object can be called directly in JavaScript code, e.g., here we define an updateSelection slot, that can be called from the JavaScript side with a list of selected items. The plugin then handles any related computations in the corresponding C++ function.

6.2 Data Exploration – Practitioner Perspective

Practitioners in various disciplines work with high-dimensional data sets. Here, we consider the exemplary case of exploring remote sensing data using ManiVault. Similar to other application areas, visual exploration of geospatial data is considered important but challenging [18]. While specific considerations and final insights will differ from domain to domain, we can follow the task abstraction by Lam et al. [30] to create a partial workflow that will be representative of many fields (R2).

We want to explore a hyperspectral image data set, the HYDICE image of the National Mall [32], showing 307 by 1280 pixels, each attached to 191 spectral bands covering the 0.4 μm to 2.4 μm region of the light spectrum reflected by the objects in view. Each band can be interpreted as an image channel. A major objective when exploring hyperspectral images is the identification of surface cover classes. It is typical to manually define class labels for a small subset of pixels that afterwards are used in semi-supervised automated classification for the rest of the data. Connecting any derived features from the spectrum back to the spatial image layout is essential during these analysis steps. More specifically, our goals are now to (I) explore the data, connected to the task of *discovering and describing observations*, and to (II) explain these observations by *identifying main causes*. These steps will yield well-justified classes that can be used in downstream analysis.

First, in ManiVault, we load the HYDICE data set using an *image loader plugin*. To inspect the loaded image we can open it in an *image viewer plugin*, which provides single-channel and false-coloring visualizations based on any three channels. We additionally open a *spectral view plugin* which shows the full spectrum of a single pixel or the averaged spectrum of a selection that we define in the image viewer, resulting in the setup of Fig. 10a. Then, to easily discover a hierarchical class structure, we use the *HSNE analytics plugin* to create a hierarchical embedding of the data employing angular distance: we open

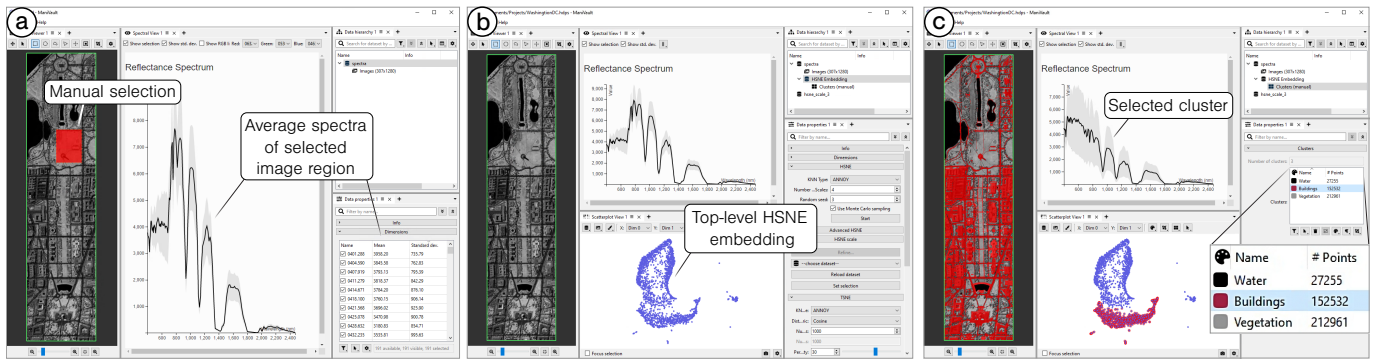


Fig. 10: **A typical exploration workflow with ManiVault:** A user can open and re-arrange views on the fly, derive new data sets using analytics plugins and connect parameters between plugins. Linked colormaps of the scatterplot and image viewer are shown in Fig. 1.

the analysis through a context-menu of the data set entry in the *data hierarchy*, select the cosine distance metric, start the embedding and display it in a *scatterplot* as seen in Fig. 10b. Next, we manually outline three clusters that are apparent in the top-level HSNE embedding as shown in Fig. 1 (center top). To inspect their spectra, we drag and drop the new cluster data set from the data hierarchy into the spectral viewer, Fig. 1 (right). Additionally, we might inspect the cluster sizes in the *data properties*. Clicking on a specific cluster displayed in the data properties will select corresponding data points in the embedding and highlight corresponding pixels in the image (Fig. 10c). Thus, we can quickly relate the cluster spectra to image positions and define the main pixel classes water, vegetation, and buildings. We want to focus on a single cluster — the one corresponding to buildings. Therefore, we refine the cluster of interest to a lower HSNE hierarchy level through a context menu opened by clicking inside the embedding — the HSNE plugin added an action to the data set that is displayed there as well as in the data properties window. To establish a visual connection between the spatial data layout and embedding, we drag the new embedding data set to the image viewer, which automatically infers the proper image dimensions for the data subset from its parent in the data hierarchy and converts it into an additional image layer. Further, we can link the colormaps of this image layer and the embedding through the parameter-sharing system by publishing one and connecting the other to it (R3). Zooming into a spatial area of interest, Fig. 1 (left), we can discriminate between several building structures like houses and streets, and even create sub-classes of roofs that immediately stand out thanks to the embedding-based recoloring.

The above procedure intertwined the accomplishment of goals (I) and (II). ManiVault made it easy to connect various views on the data, i.e., a spatial layout, high-dimensional pixel attributes, and derived features in the form of embedding positions. We quickly discriminated between classes in the data and identified differing spectral characteristics as their cause. A video that walks through the full procedure can be found as supplemental material.

6.3 Sharing Analysis Setups – Designer Perspective

ManiVault’s workspace and project features can be used to save and continue an analysis session but also enable dissemination of results and complete workflows. To showcase this, we re-implemented the Cytosplore Viewer application [15] dedicated to sharing the results of Bakken et al. [5] in ManiVault, shown in Fig. 11. Instead of having to write an entire stand-alone application to share an interactive environment alongside data to explore related insights in, we can use a ManiVault project to bundle both views and data (R5).

The viewer application depicts RNA sequencing data on brain cells from three vertebrate species. The viewer aims to highlight differences in the expression of genes and cell types that are shared across the species as described in the original paper. The main elements of the viewer application are three scatterplots showing t-SNE embeddings of the gene data of each species, a hierarchical cluster viewer showing cell types, and a table view showing statistical properties of the expression data. To create the viewer, we configure ManiVault’s GUI from within

the GUI (R4). We start with loading all data sets and setting up a single scatterplot plugin. We link scatterplot parameters like its colormap to a global settings panel that lets users configure all three scatterplots, like in the original application. Its settings can be saved as a preset which we use for the other two scatterplot instances. Similarly, we populate the cluster hierarchy view and table viewer with data. Figure 11 shows a configuration in which a user-selected entry in the table view defines the data attributes (here a gene’s expression) used to recolor the scatterplot data points (here tissue samples).

ManiVault’s Studio Mode allows us to lock this setup of views and parameter connections. This is achieved by simply publishing the current view layout, loaded data, and parameter linkage through the "File" menu tab. We can now share the viewer with other parties.

7 DISCUSSION AND CONCLUSION

This paper describes the design considerations for and implementation of ManiVault, an extensible visual analytics framework for high-dimensional data. Due to its modular architecture and data-centric design, the software enables flexible exploration and analysis workflows. We presented various plugins that provide visualization and analytics functionalities to the system. To build upon these, we showed how existing libraries can be easily incorporated into the system. ManiVault’s action and event systems allow users to adjust plugins and their interplay, enabling the creation of fully customized applications.

Currently, the system provides data plugins that cover a wide range of applications. New data types like multivariate graph data [26] can be introduced into the system as new data plugins without changes to the application’s core. We plan to extend the current serialization mechanism, used for saving the state of the system, to handle information about interaction history and other kinds of provenance [45]. Finally, we would like to include analytics plugins that run code in interpreted languages like Python or R, to easily integrate the vast amount of data science tools available in those languages.

We believe that ManiVault has great potential in aiding with the creation and use of visual analytics applications for visualization developers, practitioners, and application designers.

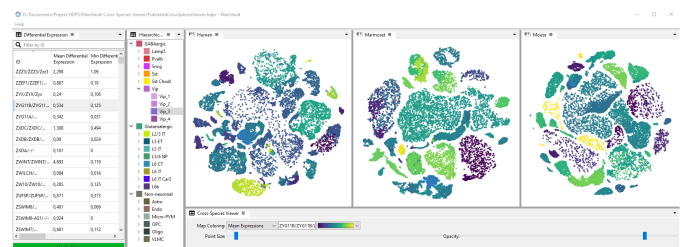


Fig. 11: Screenshot of a re-implementation of a **Cytosplore Viewer** for comparative cellular analysis of motor cortex in human, marmoset, and mouse [5]. The viewer shows embeddings of cells from the three species in combination with a shared cluster hierarchy and the option to calculate differential gene expression. See Suppl. S2 for a larger figure version.

SUPPLEMENTAL MATERIALS

All supplemental materials are available on OSF at <https://osf.io/9k6jw/>, released under a CC BY SA 4.0 license. In particular, they include (1) benchmark results, S1, and a larger version of Fig. 11, S2, (2) Excel files containing the data presented in S1, (3) Python scripts to run the nptsne benchmark from S1, (4) two videos showcasing ManiVault and (5) a full version of this paper.

ACKNOWLEDGMENTS

Author contributions: Alexander Vieth: Writing and Plugin Development; Thomas Kroes: Lead Developer (Core and Plugins) & Architect; Julian Thijssen: Developer & Architect of initial core, Plugin Developer; Baldur van Lew: Build Infrastructure; Jeroen Eggermont and Soumyadeep Basu: Viewer & Plugin Development; Elmar Eisemann, Anna Vilanova, Thomas Höllt, and Boudewijn Lelieveldt: project conception, manuscript writing, general supervision.

This work received financial support from the NWO TTW project 3DOMICS (NWO: 17126), the NWO Gravitation project BRAINSCAPES: A Roadmap from Neurogenetics to Neurobiology (NWO: 024.004.012), and the NIH Brain Initiative Cell Atlas Network (UM1MH130981).

REFERENCES

- [1] C. Ahlberg. Spotfire: An information exploration environment. *ACM SIGMOD Record*, 25(4):25–29, 1996. [tibco.com](https://doi.org/10.1145/245882.245893), archived webpage. doi: 10.1145/245882.245893
- [2] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proc. CHI*, pp. 313–317. ACM, New York, 1994. doi: 10.1145/191666.191775
- [3] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large-data visualization. In C. D. Hansen and C. R. Johnson, eds., *Visualization Handbook*, pp. 717–731. Butterworth-Heinemann, Burlington, MA, USA, 2005. doi: 10.1016/B978-012387582-2/50038-1
- [4] S. K. Badam, N. Elmqvist, and J.-D. Fekete. Steering the craft: UI elements and visualizations for supporting progressive visual analytics. *Computer Graphics Forum*, 36(3):491–502, 2017. doi: 10.1111/cgf.13205
- [5] T. E. Bakken, N. L. Jorstad, and Q. Hu et al. Comparative cellular analysis of motor cortex in human, marmoset and mouse. *Nature*, 598(7879):111–119, 2021. doi: 10.1038/s41586-021-03465-8
- [6] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Enabling interactive multiple-view visualizations. In *Proc. VIS*, pp. 135–142. IEEE, New York, 2005. doi: 10.1109/VISUAL.2005.1532788
- [7] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185
- [8] M. Brehmer and T. Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, 2013. doi: 10.1109/TVCG.2013.124
- [9] A. Buja, J. McDonald, J. Michalak, and W. Stuetzle. Interactive data visualization using focusing and linking. In *Proc. VIS*, pp. 156–163. IEEE, New York, 1991. doi: 10.1109/VISUAL.1991.175794
- [10] X. Chen, W. Zeng, Y. Lin, H. M. Al-manee, J. Roberts, and R. Chang. Composition and configuration patterns in multiple-view visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1514–1524, 2021. doi: 10.1109/TVCG.2020.3030338
- [11] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, et al. *VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data*. Chapman and Hall/CRC, 2012. doi: 10.1201/b12985-29
- [12] K. A. Cook and J. J. Thomas. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2005. osti.gov/biblio/912515.
- [13] Q. Cui, M. Ward, E. Rundensteiner, and J. Yang. Measuring data abstraction quality in multiresolution visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):709–716, 2006. doi: 10.1109/TVCG.2006.161
- [14] W. Cui. Visual analytics: A comprehensive overview. *IEEE Access*, 7:81555–81573, 2019. doi: 10.1109/ACCESS.2019.2923736
- [15] J. Eggermont, B. van Lew, N. Pezzotti, A. Mahfouz, A. Vilanova, T. Höllt, and B. Lelieveldt. Cytosplore Viewer. viewer.cytosplore.org, archived webpage.
- [16] J.-D. Fekete. The InfoVis Toolkit. In *Proc. INFOVIS*, pp. 167–174. IEEE, New York, 2004. doi: 10.1109/INFOVIS.2004.64
- [17] R. Fuchs and H. Hauser. Visualization of multi-variate scientific data. *Computer Graphics Forum*, 28(6):1670–1690, 2009. doi: 10.1111/j.1467-8659.2009.01429.x
- [18] M. Gahegan. Visual exploration and explanation in geography analysis with light. In H. J. Miller and J. Han, eds., *Geographic Data Mining and Knowledge Discovery*, chap. 11, pp. 291–324. CRC Press, 2nd ed., 2009. doi: 10.1201/9781420073980-11
- [19] A. Ghosh, M. Nashaat, J. Miller, S. Quader, and C. Marston. A comprehensive review of tools for exploratory analysis of tabular industrial datasets. *Visual Informatics*, 2(4):235–253, 2018. doi: 10.1016/j.visinf.2018.12.004
- [20] J. Heer, S. K. Card, and J. A. Landay. Prefuse: A toolkit for interactive information visualization. In *Proc. CHI*, pp. 421–430. ACM, New York, 2005. doi: 10.1145/1054972.1055031
- [21] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive immune cell phenotyping for large single-cell datasets. *Computer Graphics Forum*, 35(3):171–180, 2016. doi: 10.1111/cgf.12893
- [22] [githubuser0x00000000](https://github.com/0x00000000/Advanced-Docking-System). Advanced docking system for Qt. [github.com/Qt-Advanced-Docking-System](https://github.com/0x00000000/Advanced-Docking-System), archived webpage.
- [23] D. Jönsson, P. Steneteg, E. Sundén, R. Englund, S. Kotttravel, M. Falk, A. Ynnerman, I. Hotz, and T. Ropinski. Inviwo - a visualization system with usage abstraction levels. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3241–3254, 2020. doi: 10.1109/TVCG.2019.2920639
- [24] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):495–513, 2013. doi: 10.1109/TVCG.2012.110
- [25] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. In A. Kerren, J. T. Stasko, J.-D. Fekete, and C. North, eds., *Information Visualization: Human-Centered Issues and Perspectives*, Lecture Notes in Computer Science, pp. 154–175. Springer, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-70956-5_7
- [26] A. Kerren, H. C. Purchase, and M. O. Ward, eds. *Multivariate Network Visualization*, vol. 8380 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, Switzerland, 2014. doi: 10.1007/978-3-319-06793-3
- [27] Kitware. Trame. kitware.github.io/trame, archived webpage.
- [28] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. jupyter.org, archived webpage, github.com/voila-dashboards/voila, 2016.
- [29] R. Krueger, J. Beyer, W.-D. Jang, N. W. Kim, A. Sokolov, P. K. Sorger, and H. Pfister. Facetto: Combining unsupervised and supervised learning for hierarchical phenotype analysis in multi-channel image data. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):227–237, 2020. doi: 10.1109/tvcg.2019.2934547
- [30] H. Lam, M. Tory, and T. Munzner. Bridging from goals to tasks with design study analysis reports. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):435–445, 2018. doi: 10.1109/TVCG.2017.2744319
- [31] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Proc. PacificVis*. IEEE, New York, 2011. doi: 10.1109/pacificvis.2011.5742387
- [32] D. A. Landgrebe. HYDICE image of washington dc mall. engineering.purdue.edu, archived webpage.
- [33] C. Li, J. Thijssen, T. Abdelaal, T. Höllt, and B. Lelieveldt. Spacewalker: Interactive gradient exploration for spatial transcriptomics data. *bioRxiv*, 2023. doi: 10.1101/2023.03.20.532934
- [34] Y. Ma, T. Xie, J. Li, and R. Maciejewski. Explaining vulnerabilities to adversarial machine learning through visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1075–1085, 2020. doi: 10.1109/TVCG.2019.2934631
- [35] K. Matkovic, W. Freiler, D. Gracanin, and H. Hauser. ComVis: A coordinated multiple views system for prototyping new visualization technology. In *Proc. IV*, pp. 215–220. IEEE, New York, 2008. doi: 10.1109/IV.2008.87
- [36] N. Pezzotti. High dimensional inspector. github.com/Nicola17/High-Dimensional-Inspector, 2018. doi: 10.5281/zenodo.1303855
- [37] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. *Computer Graphics Forum*, 35(3):21

- 30, 2016. doi: [10.1111/cgf.12878](https://doi.org/10.1111/cgf.12878)
- [38] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Höllt, B. V. Lew, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. GPGPU linear complexity t-SNE optimization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1172–1181, 2020. doi: [10.1109/tvcg.2019.2934307](https://doi.org/10.1109/tvcg.2019.2934307)
- [39] M. Pi, H. Yeon, H. Son, and Y. Jang. Visual cause analytics for traffic congestion. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2186–2201, 2021. doi: [10.1109/TVCG.2019.2940580](https://doi.org/10.1109/TVCG.2019.2940580)
- [40] H. Piringer, W. Berger, and H. Hauser. Quantifying and comparing features in high-dimensional datasets. In *Proc. IV*, pp. 240–245. IEEE, New York, 2008. doi: [10.1109/IV.2008.17](https://doi.org/10.1109/IV.2008.17)
- [41] H. Piringer, C. Tominski, P. Muigg, and W. Berger. A multi-threading architecture to support interactive visual exploration. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1113–1120, 2009. doi: [10.1109/TVCG.2009.110](https://doi.org/10.1109/TVCG.2009.110)
- [42] C. Plaisant, D. Carr, and B. Shneiderman. Image-browser taxonomy and guidelines for designers. *IEEE Software*, 12(2):21–32, 1995. doi: [10.1109/52.368260](https://doi.org/10.1109/52.368260)
- [43] Plotly Technologies Inc. Dash. dash.plotly.com, archived webpage.
- [44] A. Popa, F. Gabrieli, T. Kroes, A. Krekeler, M. Alfeld, B. Lelieveldt, E. Eisemann, and T. Höllt. Visual analysis of ris data for endmember selection. In *Proc. GCH*. The Eurographics Association, Eindhoven, NL, 2022. doi: [10.2312/gch.20221233](https://doi.org/10.2312/gch.20221233)
- [45] E. D. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purpose. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):31–40, 2016. doi: [10.1109/TVCG.2015.2467551](https://doi.org/10.1109/TVCG.2015.2467551)
- [46] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2092–2101, 2014. doi: [10.1109/TVCG.2014.2346291](https://doi.org/10.1109/TVCG.2014.2346291)
- [47] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proc. CMV*, pp. 61–71. IEEE, New York, 2007. doi: [10.1109/CMV.2007.20](https://doi.org/10.1109/CMV.2007.20)
- [48] N. Sakamoto and K. Koyamada. KVS: A simple and effective framework for scientific visualization. *Journal of Advanced Simulation in Science and Engineering*, 2(1):76–95, 2015. doi: [10.15748/jasse.2.76](https://doi.org/10.15748/jasse.2.76)
- [49] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, 2014. doi: [10.1111/cgf.12391](https://doi.org/10.1111/cgf.12391)
- [50] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: [10.1109/TVCG.2016.2599030](https://doi.org/10.1109/TVCG.2016.2599030)
- [51] W. Schroeder, K. Martin, and B. Lorensen. *The visualization toolkit*. Kitware, 4th ed., 2006. gitlab.kitware.com/vtk/textbook, archived pdf.
- [52] Slicer Community. 3D Slicer. slicer.org, archived webpage.
- [53] J. Sorger, T. Ortner, H. Piringer, G. Hesina, and E. Gröller. A taxonomy of integration techniques for spatial and non-spatial visualizations. In *Proc. VMV*. The Eurographics Association, Eindhoven, NL, 2015. doi: [10.2312/vmv.20151258](https://doi.org/10.2312/vmv.20151258)
- [54] D. Stalling, M. Westerhoff, and H.-C. Hege. amira: A highly interactive system for visual data analysis. In C. D. Hansen and C. R. Johnson, eds., *Visualization Handbook*, pp. 749–767. Butterworth-Heinemann, Burlington, 2005. doi: [10.1016/B978-012387582-2/50040-X](https://doi.org/10.1016/B978-012387582-2/50040-X)
- [55] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014. doi: [10.1109/TVCG.2014.2346574](https://doi.org/10.1109/TVCG.2014.2346574)
- [56] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002. doi: [10.1109/2945.981851](https://doi.org/10.1109/2945.981851)
- [57] D. Sun, R. Huang, Y. Chen, Y. Wang, J. Zeng, M. Yuan, T.-C. Pong, and H. Qu. PlanningVis: A visual analytics approach to production planning in smart factories. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):579–589, 2020. doi: [10.1109/TVCG.2019.2934275](https://doi.org/10.1109/TVCG.2019.2934275)
- [58] D. F. Swayne, D. T. Lang, A. Buja, and D. Cook. GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43(4):423–444, 2003. doi: [10.1016/S0167-9473\(02\)00286-4](https://doi.org/10.1016/S0167-9473(02)00286-4)
- [59] Tableau Software, LLC. Tableau. tableau.com, archived webpage.
- [60] The Qt Company. Qt. qt.io, archived webpage.
- [61] Thermo Fisher Scientific. Amira. thermofisher.com, archived webpage.
- [62] J. Thijssen, Z. Tian, and A. Telea. Scaling Up the Explanation of Multidimensional Projections. In M. Angelini and M. El-Assady, eds., *Proc. EuroVA*. The Eurographics Association, 2023. doi: [10.2312/eurova.20231098](https://doi.org/10.2312/eurova.20231098)
- [63] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. jmlr.org/vandermaaten08a.
- [64] A. Vieth, A. Vilanova, B. Lelieveldt, E. Eisemann, and T. Höllt. Incorporating texture information into dimensionality reduction for high-dimensional images. In *Proc. PacificVis*, pp. 11–20. IEEE, New York, 2022. doi: [10.1109/PacificVis53943.2022.00010](https://doi.org/10.1109/PacificVis53943.2022.00010)
- [65] Visplore GmbH. Visplore. visplore.com, archived webpage.
- [66] M. Ward. XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proc. VIS*, pp. 326–333. IEEE, New York, 1994. davis.wpi.edu/xmdv, archived webpage. doi: [10.1109/VISUAL.1994.346302](https://doi.org/10.1109/VISUAL.1994.346302)
- [67] J. D. Wolfe and S. R. Black. Hyperspectral analytics in envi target detection and spectral mapping methods. Technical report, Harris Corporation, 2018. l3harrisgeospatial.com/Whitepaper.pdf, archived pdf.
- [68] P. C. Wong and R. D. Bergeron. 30 years of multidimensional multivariate visualization. In *Scientific Visualization, Overviews, Methodologies, and Techniques*, p. 3–33. IEEE, New York, 1997.
- [69] J. Wu, D. Liu, Z. Guo, Q. Xu, and Y. Wu. TacticFlow: Visual analytics of ever-changing tactics in racket sports. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):835–845, 2022. doi: [10.1109/TVCG.2021.3114832](https://doi.org/10.1109/TVCG.2021.3114832)
- [70] M. A. Yalçın, N. Elmqvist, and B. B. Bederson. Keshif: Rapid and expressive tabular data exploration for novices. *IEEE Transactions on Visualization and Computer Graphics*, 24(8):2339–2352, 2018. doi: [10.1109/TVCG.2017.2723393](https://doi.org/10.1109/TVCG.2017.2723393)
- [71] J. Yang, M. O. Ward, and E. A. Rundensteiner. Interactive hierarchical displays: A general framework for visualization and exploration of large multivariate data sets. *Computers & Graphics*, 27(2):265–283, 2003. doi: [10.1016/S0097-8493\(02\)00283-2](https://doi.org/10.1016/S0097-8493(02)00283-2)
- [72] J. Yang, M. O. Ward, E. A. Rundensteiner, and S. Huang. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *Proc. VisSym*. The Eurographics Association, Eindhoven, NL, 2003. doi: [10.2312/VisSym/VisSym03/019-028](https://doi.org/10.2312/VisSym/VisSym03/019-028)
- [73] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):304–314, 2021. doi: [10.1109/TVCG.2020.3030367](https://doi.org/10.1109/TVCG.2020.3030367)