Towards a Progressive Open Source Framework for SciVis and InfoVis

Gueunet Charles*
Kitware

Mazen François†
Kitware

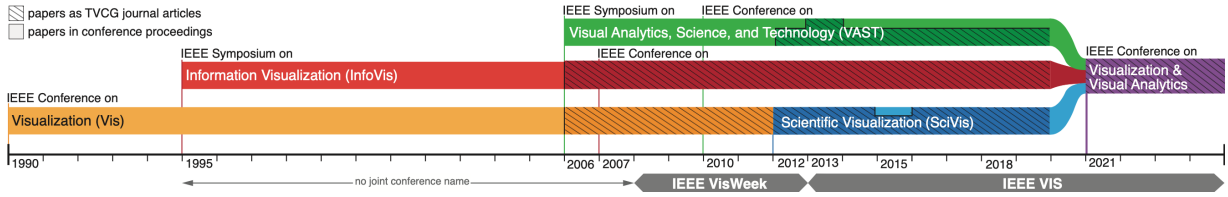

Figure 1: IEEE VIS terminology history from [1].

Abstract

In a world where data has become too large for direct human perception, scientists have developed methods for specific data exploration. Until recently, two main methodologies were used for their exploration: scientific visualization (SciVis) for data with inherent geometry (simulation/acquisition) and information visualization (InfoVis) for abstract data. Though these fields evolved in parallel, sharing journals and conferences, they had distinct challenges, methodologies, and experts. Recently, a visible transition has begun, with the two communities converging, exemplified by IEEE VIS conference removing distinct categories. In this context, we propose a high-level discussion on an open-source framework widely used in SciVis and how progressive processing and visualization could help bringing its abilities to InfoVis.

Index Terms: Scientific Visualization, Information Visualization, Methodology, Tools.

1 Introduction

As data grows in size and complexity, scientists continuously seek new methods for interactive visualization and exploration, leading to the creation of new domains. Taking IEEE VIS, the premiere conference in visualization research as an indicator of a more general trend, we observe a recent convergence between the fields of data visualization.

In this article, we present a free and open source (FOSS) frameworks widely used in scientific visualization, and we propose a discussion on how adapting this framework to progressive analysis would help reach a wider public in the visualization community.

2 Context and State of the Art

The fields of scientific visualization and information visualization have historically been distinct yet complementary domains within the broader context of data visualization. Taking as a reference, the terminology used for IEEE VIS [11], emphasized in Fig. 1, *Scientific Visualization* focuses on the representation and exploration of data coming from simulation or acquisition of a physical phenomenon, often dealing with complex two or three-dimensional data sets. In contrast, *Information visualization* primarily handles abstract, often high-dimensional data that may not have an inherent spatial structure.

*e-mail: charles.gueunet@kitware.com
†e-mail: francois.mazen@kitware.com

In the field of scientific visualization (**SciVis**), the volume of data to be analyzed has expanded over time in tandem with the increasing capabilities of supercomputers running simulations. Consequently, the tools employed in this field are usually developed with distributed computing in mind. One example is ParaView [3], a free and open-source (FOSS) application and framework widely used by the SciVis community. Recently, significant efforts have been made regarding in situ exploration, to visualize and process data directly from simulations as they execute, for example with Catalyst [4, 5] or Visit LibSim [8].

At the same time, information visualization (**InfoVis**) is meant to deal with more abstract data, like exploring customer market basket in retail data, and scientists in this domain are more used to build their own tailored application. A consequence is that the dominant technologies in the domains are more tools or frameworks meant to build not only the user interface, but the whole user experience with tailored selection and query and specific representations. Examples include language-based tools like R [10] and Vega-Lite [15] various Python-based solutions (IPython [14] and its Jupyter derivatives), as well as dashboard-based solutions like Tableau [13].

While both fields focus on (large) data analysis, they have adopted distinct methodologies due to differing priorities of their user bases. However, after years of exploration, the remaining challenges in both fields increasingly converge, with interactivity and data size being at the core of the attention. Therefore, we believe that adding progressive analysis capabilities into a **SciVis**-enabled solution could be a promising approach to support scientists in both fields.

3 ParaView and VTK: Current State

This section provides an overview of the ParaView (and VTK [12]) features that are already publicly available and could be used for data visualization in its broader definition, including InfoVis.

As ParaView relies on a client-server architecture, we will use the term **Backend** to denote everything happening on the server side, and usually related to the data processing. On the other end, the term **Frontend** will be used to denote the client side, usually related to pipeline definition, user interactions and data representation configurations.

3.1 Backend

From the start, the VTK data model has been designed with distribution in mind. Consequently, adding a new processing filter in ParaView requires minimal effort to support distributed data, and in many cases, no additional work is needed. We think this ability played an important role in the wide adoption of ParaView by the HPC community. Even though this data model mainly focuses on data having a geometry, graphs, trees and tables are also supported and can thus be processed and represented in a distributed fashion.

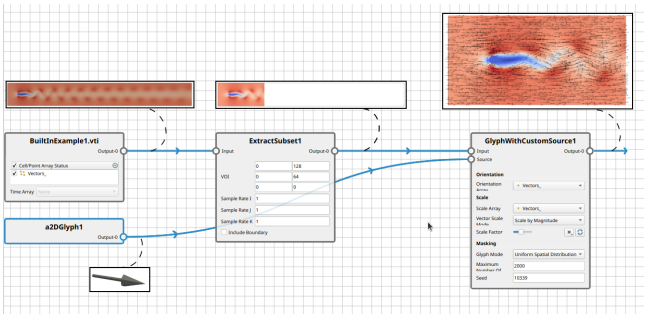

Figure 2: Processing pipeline in ParaView's node editor with data flowing from left to right. First a CFD data set is read, then a subset is extracted by applying the corresponding filter and finally glyphs are added to the result, taking a custom generated arrow model as the second input.

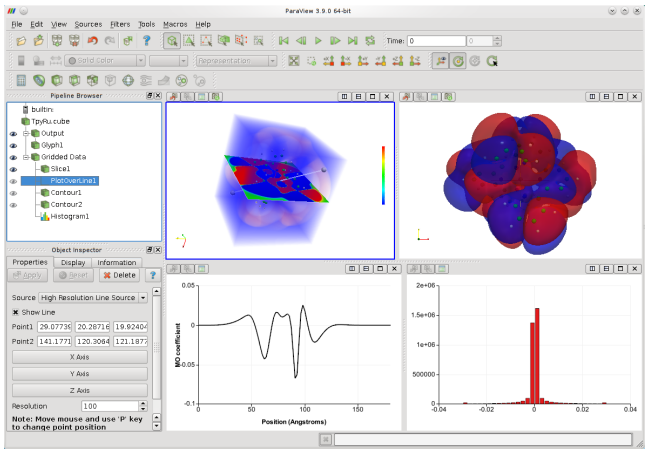

Figure 3: Example of a ParaView interface with the pipeline on the leftmost part and an analysis results display in four windows mixing 3D representations and charts.

To process those data, VTK is based on a *processing and rendering pipeline*. The processing pipeline as highlighted in Fig. 2 is a direct acyclic graph where a node is a processing filters and an edge represent a connection between one of the outputs of a filter to one of the inputs of another one. Some key features of this pipeline are:
On-Demand processing: Filters are only executed when their output is required, preventing unnecessary computations. This includes partial pipeline updates: as each filter maintains its most recent output(s) in cache, it is possible to only process the minimum amount of filters when executing the pipeline after a change. Using the pipeline from Fig. 2, changing the arrow model would only update the glyph filter without re-executing the extraction one.
Parallel Processing: Filters and other pipeline components can handle distributed datasets efficiently. Multi-threads parallelism is also used to leverage multi-core architecture.

The cleverness around the data flow in the pipeline graph and the execution of the filters is handled in specific classes named *executives*. For example, the simplest executive already ensures *On-Demand processing* as described previously, while the more advanced executives bring additional functionalities regarding data distribution, time management, or even to limit the size of cached data along the pipeline. The ParaView pipeline can operate in a *streaming* manner, processing and visualizing data as it is generated. In this mode, the pipeline is regularly re-executed and specific filters can be used to aggregate the data incrementally as it is the case with the SLAM filter of LidarView for example. This feature is important as it is a key feature for future progressive analysis.

Once the processing is done, we may follow up with serialization of the output on the disk. Thanks to VTK, distributed IO are natively supported including for tables. Distributed tables are also available for in situ processing. Another important feature it the rendering, either in an interactive scene, or through screenshots / cinema databases generations. Unfortunately, as we will detail in the next section, in its current state the rendering of distributed tables often involve to transfer most of the data to the client side.

3.2 Frontend

Thanks to the client-side architecture mentioned previously, it is possible to use the ParaView server with various clients, ranging from a Qt client (cf. Fig. 3) to a python shell or notebook, or even a web based interface. Regarding the python client, it is worth mentioning that ParaView also has a Jupyter Notebook kernel, or an integration trough Trame, a framework made to quickly build web application with data visualization in mind.

For developers that want to quickly prototype an analysis pipeline based on VTK, it is possible to use Python thanks to the native wrapping, although this is not the most convenient way. In order to help analysts, VTK brings some useful utilities to converts its data object to NumPy arrays back and forth, and a more pythonic syntax is currently being added. It is worth mentioning here PyVista [16], a Python package that greatly simplifies VTK usage and provides another Jupyter Notebook integration.

In the following sections, we outline the common features of most existing interfaces, noting that they usually offer additional functionalities tailored to specific user needs. In terms of data representation, a wide range of methods is available for exploring geometric datasets, ranging from fast rasterisation techniques to photo-realistic but computationally heavy ray tracing. Abstract data is usually also supported through several representations such as line charts, histogram, parallel coordinates, or plot matrix view box. An example mixing 3D views with charts is presented in Fig. 3.

In these views, beyond the standard interactions for updating the viewpoint, special objects called *widgets* can be added to the scene (or chart) to enhance interactivity. Widgets are typically used to help with parameters configuration for filters in the pipeline; for example, an interactive plane can be manually positioned to set up a slice. Another form of interactivity is the *selection*: the VTK data model includes a class specifically designed to handle complex subsets of data. It is based on a tree, where each node represents an expression defining a subset, connected by edges corresponding to simple operators like **AND**, or **OR**. This class itself does not assume any geometry on the data. The frontend is responsible for offering tools to the user, potentially utilizing widgets, to generate these selections. Currently, most frontends provide both geometric tools and a straightforward request mechanism to fill the expression nodes of the previously defined tree.

3.3 Open Communities

While the previous paragraphs were focusing on the technical aspect, it is worth mentioning that both VTK and ParaView being FOSS, they have fostered a wide community of users and contributors. Through public forums like their Discourse [1,2], or tools like Gitlab [3,4] to open issues and submit merge requests, scientists collaborate, support one another, and actively participate in shaping the future of these software tools.

[1] https://discourse.vtk.org/
[2] https://discourse.paraview.org
[3] https://gitlab.kitware.com/vtk/vtk
[4] https://gitlab.kitware.com/paraview/paraview

4 Limitation

In order to meet InfoVis expectations, SciVis tools are facing several limitations, either on backend side or frontend side.

First, most of the VTK filters assume that the data has geometry and topology. For example, in order to process a point cloud the user has to create a *polydata* data model where vertices correspond to the point locations. A polydata is a commonly used data model to represent a geometric structure consisting of vertices, lines, polygons, and/or triangle strips, with point and cell attribute values. Using this structure is sub-optimal for InfoVis because it leads to extra memory usage and provides no easy way to manage high-dimensional data. The usual work-around is to map some dimensions to point coordinates and other dimensions to Point Data. This is because most of the SciVis filters assume few dimensions, usually up to 6 for mechanical tensor processing. The Topology Toolkit [6] (TTK), available in VTK and ParaView, adds specific filters, like the Rips-complex or the dimension reduction [7], to manage high-dimensional data and maps the result to VTK's low-dimensional model.

An other important limitation in the SciVis processing filters is the blocking call mechanism, which means that while the processing is running, the host application must wait for the end of the process before visualizing any result. In addition, when VTK filters are chained in a processing pipeline, there is a strict barrier in-between filters to ensure that inputs are valid. This barrier currently prevents any real progressive analysis.

On the user interaction side, most of InfoVis data exploration implies selecting the relevant data and dimension to display. In existing frontends, interactive selection is usually based on geometry, for example filtering the cells or the points to display with the mouse, or combining simple expressions based on fields values by basic operations. Hence, VTK lacks advanced SQL-like request capability where the user can combine several criteria to reduce dimensionality and the number of elements to display.

In addition, while it is possible to define custom processing filters with Python, there is currently no high level grammar to define user specific request and visualization, like Vega and Vega-Lite provide. Thankfully, the python wrappings of VTK and ParaView tend to help interoperability with classic InfoVis tooling like Jupyter Notebooks. Unfortunately, in case of non-geometric data the current implementation of VTK requires to fetch the whole data set client side for visualization, which is not scalable.

5 Perspectives

The lack of progressiveness in SciVis has been identified as a key factor that limits InfoVis usages. To fulfill InfoVis requirements, this progressiveness should be improved at the processing level (Backend) and at the user level (Frontend).

At the backend level, the data model could itself be progressive. For example, the actual Adaptative Mesh Refinement (AMR) Data Models, very popular in numerical simulation at scale like CFD or Astrophysics, is a progressive data model. This hierarchical model makes it possible to process and render at different levels of precision, sometimes pre-computed, which could be generalized to process large high-dimensional InfoVis datasets. For example, `t8code` [9] from DLR is a current implementation of this approach. In addition, the computation itself could be progressive and output approximate result in a timely fashion. A confidence interval could be provided in order to refine in an iterative way up to an acceptable answer. Ultimately, the system makes sure to output a result at any given computing time, in order to create responsiveness for the users. At VTK level, this new mechanism would be implemented as a new execution model, with specific filters able to deliver results at regular time intervals. Ultimately, the frequency of the output could be driven by the executive in order to ensure progressiveness in the user experience.

Most of InfoVis tooling are usable in high level scripting languages like python. Hence, SciVis tools should be also driven by python scripts. VTK and ParaView are developed in C++, but a python wrapping is available. It allows seamless integration with InfoVis tools like SciKit learn or Jupyter Notebook, and Deep Learning frameworks like Pytorch. However the syntax of the VTK python API is basically a reflection of the C++ API, which is not intuitive for InfoVis scientists. Pyvista [16] is a tentative to add such pythonic layer on top of the existing VTK python wrapping. The SciVis tools could implement similar sugar syntax to help integration in InfoVis pipelines.

Modularity is also key in InfoVis, and is also lacking in SciVis tools due to historical reasons. ParaView Async [2] is an initiative to bring asynchronous computation to the ParaView framework with a micro-service approach. It strongly decouples the processing service, the data service and the rendering service to improve the user experience with more responsiveness while the computing is in progress. For other VTK-based application, multi-process approach could be leveraged to guarantee similar interactivity for the InfoVis users.

6 Conclusion

Following the convergence of the InfoVis and SciVis communities, we analyzed the current state of a popular SciVis framework. From this convergence view, we identified several limitations which are holding the adoption of SciVis tools for InfoVis works. One of the major factors is the lack of progressiveness either on the processing side and on the user interaction side. This approach has to be developed to bring SciVis and InfoVis to a common playground, with strategic advantages at scale like reducing computing resources needed.

We invite researchers, practitioners, and enthusiasts interested in having a progressive analysis enabled FOSS solution to reach forward to us or contribute.

Acknowledgments

The authors wish to thank Louis Gombert and Léon Victor for the internal review.

References

[1] IEEE VIS 2021 website. https://ieeevis.org/year/2024/blog/things-are-changing-2021. Accessed: 2024-06-11.

[2] ParaView Async website. https://gitlab.kitware.com/async/async-paraview. Accessed: 2024-07-12.

[3] J. Ahrens, B. Geveci, and C. Law. ParaView: An end-user tool for large data visualization. In *Visualization Handbook*. Elesvier, 2005. ISBN 978-0123875822.

[4] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV 2015)*, pp. 25–29, November 2015. doi: 10.1145/2828612.2828624

[5] U. Ayachit, A. C. Bauer, B. Boeckel, B. Geveci, K. Moreland, P. O'Leary, and T. Osika. Catalyst revised: Rethinking the paraview in situ analysis and visualization API. In *High Performance Computing*, pp. 484–494, June 2021. doi: 10.1007/978-3-030-90539-2_33

[6] T. Bin Masood, J. Budin, M. Falk, G. Favelier, C. Garth, C. Gueunet, P. Guillou, L. Hofmann, P. Hristov, A. Kamakshidasan, C. Kappe, P. Klacansky, P. Laurin, J. Levine, J. Lukasczyk, D. Sakurai, M. Soler, P. Steneteg, J. Tierny, W. Usher, J. Vidal, and M. Wozniak. An Overview of the Topology ToolKit. In *TopoInVis*, 2019.

[7] H. Doraiswamy, J. Tierny, P. S. Silva, L. Nonato, and C. Silva. Topomap: A 0-dimensional homology preserving projection of high-dimensional data. *IEEE Transactions on Visualization amp; Computer Graphics*, 27(02):561–571, feb 2021. doi: 10.1109/TVCG.2020.3030441

[8] C. Harrison. *Visualization and analysis of hpc simulation data with visit*. PhD thesis, PhD thesis, Lawrence Livermore National Laboratory, 2021.

[9] J. Holke, C. Burstedde, D. Knapp, L. Dreyer, S. Elsweijer, V. Ünlü, J. Markert, I. Lilikakis, N. Böing, P. Ponnusamy, and A. Basermann. t8code v. 1.0 - modular adaptive mesh refinement in the exascale era. In *SIAM International Meshing Round Table 2023*, March 2023.

[10] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. doi: 10.1080/10618600.1996.10474713

[11] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko. vispubdata.org: A Metadata Collection about IEEE Visualization (VIS) Publications. *IEEE Transactions on Visualization and Computer Graphics*, 23, 2017. To appear. doi: 10.1109/TVCG.2016.2615308

[12] Kitware, Inc. *The Visualization Toolkit User's Guide*, January 2003.

[13] D. G. Murray. *Tableau your data!: fast and easy visual analysis with tableau software*. John Wiley & Sons, 2013.

[14] F. Pérez and B. E. Granger. Ipython: a system for interactive scientific computing. *Computing in science & engineering*, 9(3):21–29, 2007.

[15] 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, 2016.

[16] C. B. Sullivan and A. Kaszynski. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, may 2019. doi: 10.21105/joss.01450