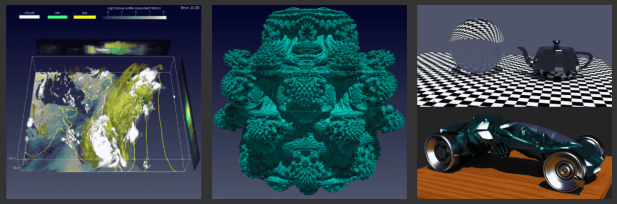
Which visualization tool is right for me?

ALEX RAZOUMOV
alex.razoumov@westdri.ca

SIMON FRASER UNIVERSITY

Digital Research Alliance of Canada

About us: the National Visualization Team



- Operational since 2014
- Part of the Alliance Federation, aka the distributed support team at Canadian universities coast to coast
- 7-8 HPC / advanced research computing staff interested in visualization tools and techniques
- The only national team with a website `https://ccvis.netlify.app`
  - gallery | click on any visualization for details
  - Visualize This! contests
  - resources | ∼30 past visualization webinars with recordings `https://training.westdri.ca/tools/visualization`
  - resources | national visualization documentation `https://docs.alliancecan.ca/wiki/Visualization`
- To reach us: support@tech.alliancecan.ca and mention “visualization” in the subject line

Winter Series

- Why?
  1. the National Visualization Team has accumulated significant expertise across many areas well outside the scope of any single workshop, and we rarely teach these topics
  2. certain topics are quite niche or advanced, making it impractical to cover them within a single institution or region
  3. as tools continue to evolve and much online information becomes outdated, we sought to present a 2026 snapshot of what we consider accurate and useful
- Web page `https://folio.vastcloud.org/winterseries` (EN)
  `https://folio.vastcloud.org/ecolehivernale` (FR)
- Format
  - 15 (mostly) hands-on workshops in English, 3 workshops in French
  - workshops progress from basic plotting to more advanced topics
    - please pay attention to Prerequisites, e.g. advanced ParaView topics require basic ParaView knowledge
  - sessions will not be recorded in order to encourage interaction; however, slides and other materials will be posted on the Winter Series page shortly before each session
  - not an annual event: whether it is repeated in a few years will depend on how this iteration goes and on your feedback

Workshops

1. Basic plotting in Python 
2. Beginner's ParaView: workflows, scripting, animation, remote / large-scale 
3. Advanced ParaView: Programmable Filter, in-situ 
4. Specialty topics: VMD, Gephi 
5. Modern web visualization 

Basic plotting in Python

Figure: Nicolas Rougier's famous adaptation of the Python Visualization Landscape slide from Jake VanderPlas's keynote at PyCon 2017

Source: `https://github.com/rougier/python-visualization-landscape`

Python plotting libraries in this Series

Trade-offs between features and simplicity, speed and beauty, static and dynamic interface

Library	Date	Gist	✔ Pros	✘ Cons
Matplotlib gallery cheatsheet	today	grandfather of Python plotting, low-level, originally static plots (1)	infinite flexibility	verbose, dated default styling
Seaborn gallery	Jan-27	for statistical data exploration, built on top of Matplotlib, static plots	understands Pandas dataframes, needs very little code, beautiful default styling and plot types	harder to customize beyond default templates
Plotly gallery and some Dash	Jan-30	web-based output, interactive (hover for data, zoom, toggle elements)	interactive, great for 3D plots and geo-maps	slow for very large datasets, steeper learning curve for complex layouts
Vega-Altair gallery	Feb-03	web-based output, "Grammar of Graphics" (describe the links between data columns and their visuals)	clean/concise code, very consistent syntax, basic interactivity	slow for very large datasets
Plotnine gallery + R's ggplot2	Feb-10	implementation of R's ggplot2 ("Grammar of Graphics") for Python	ggplot style of layering components (data + geometry + facets), splitting one chart into multiple sub-charts based on a category	less popular than Matplotlib or Plotly

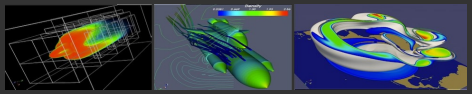
(1) Check a Matplotlib animation `https://ccvis.netlify.app/gallery/voronoi-mesh`

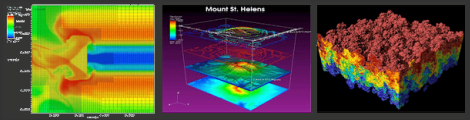
Multi-dimensional visualization

- In 3D:
  1. harder to navigate, multiple overlapping fields $\Rightarrow$ need layers and interactive filters
  2. larger datasets $\Rightarrow$ processing times can be large $\Rightarrow$ GPU and distributed rendering
- Requirements:
  - open-source + multi-platform (Linux/Mac/Windows) + general-purpose
  - visualize scalar and vector fields
  - data on top of structured (e.g. Cartesian) and unstructured meshes in 2D and 3D, particles, polygonal meshes, irregular topologies, AMR / multi-resolution datasets
  - ability to handle very large datasets (GBs to TBs), up to $10^{\sim 12}$ resolution elements
  - ability to scale to large ($10^3 - 10^5$ cores) computing facilities + parallel I/O
  - GUI interactive manipulation + Python scripting
  - client-server for remote interactive visualization
  - support for most common data formats and rendering views

Multi-dimensional visualization

- For the past 20+ years choice between ParaView and VisIt, with a ∼85/15 breakdown
- We taught full-day workshops on both over the years





- `https://www.paraview.org`
- Started in 2000 as a collaboration between Los Alamos and Kitware Inc., later joined by Sandia and other partners; first release in 2002
- Developed to visualize extremely large datasets on distributed memory machines
- ParaView is based on VTK (Visualization Toolkit): same authors, PV = GUI on top of the C++ VTK classes
- Uses MPI for distributed-memory parallelism on HPC clusters
- Big ecosystem of tools: web, in-situ, Cinema
- Latest 6.0.1 - Basic ParaView EN on Feb-06, FR on Feb-13

- `https://visit-dav.github.io/visit-website`
- Developed to visualize results of terascale simulations, first release in fall 2002
- Over 80 visualization features (contour, mesh, slice, volume, molecule, ...)
- Reads over 110 different file formats
- Full integration with VTK library
- Uses MPI for distributed-memory parallelism on HPC clusters
- Smaller ecosystem
- Latest 3.4.2

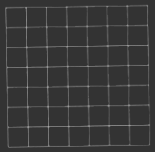
VTK = Visualization Toolkit

- Open-source software system for 3D computer graphics, image processing and visualization
- Bindings to C++, Tcl, Java, Python, and partial port to JavaScript (started in 2016)
- ParaView is based on VTK ⇒ supports all standard file formats supported by VTK
  `https://docs.vtk.org/en/latest/supported_data_formats.html`
- VTK's own file formats
  1. legacy serial formats (*.vtk): ASCII header lines + ASCII/binary data
  2. XML formats (extension depends on VTK data type): XML tags + ASCII/binary/compressed data
     - newer, much preferred to legacy VTK
     - supports parallel file I/O, compression, portable binary encoding (big/little endian byte order), random access, etc.
  3. VTKHDF formats (*.vtkhdf): in development since 2022
     - all the bells and whistles of HDF5
     - the file looks like an HDF5 file with specific formatting ⇒ can write these files using HDF5 tools, e.g. with h5py and compression with hdf5plugin
     - store the following VTK data types: vtkPolyData, vtkUnstructuredGrid (not all cell types supported), vtkImageData, vtkOverlappingAMR, vtkMultiBlockDataSet, vtkPartitionedDataSetCollection, vtkPartitionedDataSet
     - seems that vtk.vtkHDFWriter() only supports vtkPolyData and vtkUnstructuredGrid at the moment
     - on presenter's laptop:

       ```
       paraview ~/training/paraviewWorkshop/vtkhdf/warpingSpheres.vtkhdf
       ```

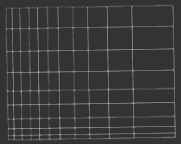
- More on VTK evolution in the web visualization section

VTK 3D data: 6 major dataset (discretization) types



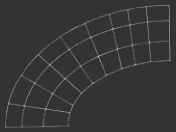
(a) Image Data

Image Data/Structured Points: `*.vti`, points on a regular rectangular lattice, scalars or vectors at each point



(b) Rectilinear Grid

Rectilinear Grid: `*.vtr`, same as Image Data, but spacing between points may vary, need to provide steps along the coordinate axes, not coordinates of each point



(c) Structured Grid

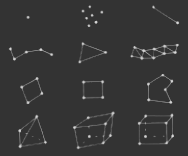
Structured Grid: `*.vts`, regular topology and irregular geometry, need to indicate coordinates of each point

(d) Unstructured Points

Particles/Unstructured Points: `*.particles`

(e) Polygonal Data

Polygonal Data: `*.vtp`, unstructured topology and geometry, point coordinates, 2D cells only (i.e. no polyhedra), suited for maps



(f) Unstructured Grid

Unstructured Grid: `*.vtu`, irregular in both topology and geometry, point coordinates, 2D/3D cells, suited for finite element analysis, structural design

Standard Python scripting in ParaView

EN on Feb-24, FR on Feb-27

- Why use scripting?
  - automate mundane or repetitive tasks, e.g., making frames for a movie
  - document and store your workflow
  - use ParaView on clusters from the command line and/or via batch jobs

1. In the GUI: View | Python Shell opens a Python interpreter
   - write or paste your script there
   - use the button to run an external script from a file
2. */usr/bin/* | */Applications/Paraview*.app/Contents/bin/* | *C:\Program Files\ParaView*\bin* pvpython will give you a Python shell connected to a ParaView server (local or remote) without the GUI
3. */usr/local/bin/* | */Applications/Paraview*.app/Contents/bin/* | *C:\Program Files\ParaView*\bin* pvbatch --force-offscreen-rendering script.py is a serial (on some machines parallel) application using a local ParaView server ✐ make sure to save your visualization
4. */usr/local/bin/* | */Applications/Paraview*.app/Contents/bin/* | *C:\Program Files\ParaView*\bin* paraview --script=codes/displayWireframe.py to start ParaView GUI and auto-run the script

Less obvious ways to use Python scripting in ParaView

5. Python Calculator Filter (vs. regular Calculator Filter) to create new data arrays
   - treats data as NumPy arrays
   - can use syntax like `inputs[0].PointData['U'] - inputs[1].PointData['U']`
   - personally, I don't use it, instead using either regular Calculator or Programmable Filter
6. Programmable Source / Filter to create new geometry
   - separate slide and workshop
7. Python state files: more robust than `*.pvsm` files for complex workflows

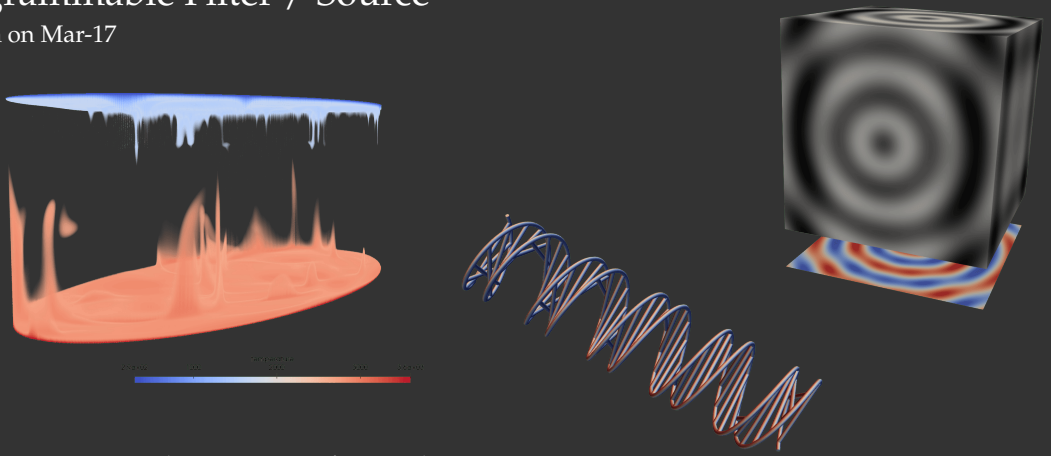
Animation in ParaView

EN on Mar-03, FR on Mar-06

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create eye-catching animations, somewhat limited in what you can do
   - in Animation View: select an object, select its property, create a new track with "+", double-click the track to edit it, press "➤"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "➤")
   - also works for multi-frame NetCDF and VTKHDF files
3. Script your animation in Python
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage: generate one frame per input file, write a Python loop

Programmable Filter / Source

Session on Mar-17



- To create new dicretizations / VTK objects
- Good for creating custom objects from scratch, e.g. projection of a volume on to a plane
- An alternative way to build your own custom file reader (if format not supported by default)

In-situ

Session on Mar-24

- Modern parallel simulations can produce huge amounts of data $\rightarrow$ I/O bottleneck
  - cannot afford to write full data to disk every time step
- In-situ visualization idea: instead of writing full 3D variables to disk, only write either:
  1. some derived data needed for visualization (e.g. 2D polygonal isosurfaces), or
  2. images directly from the simulation code

  $\Rightarrow$ much smaller disk output, no need to write full 3D datasets every time step
  - sometimes referred to as "co-processing"

- Wait ... writing images or smaller datasets to disk directly from simulation codes is nothing new: we've been doing it for the last 50 years

In-situ (cont.)

Session on Mar-24

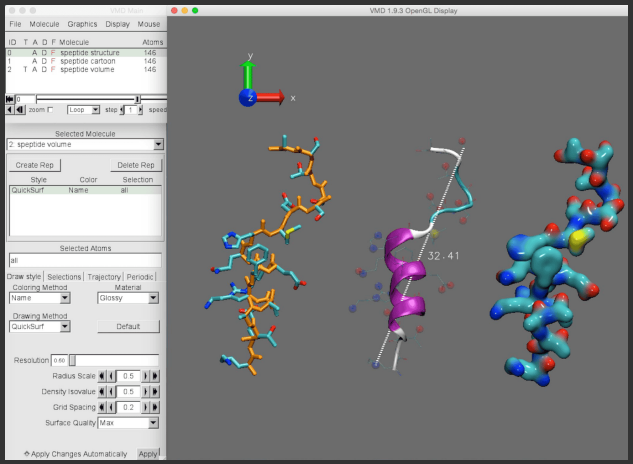
- Let's rephrase: in a ParaView Catalyst / VisIt LibSim context, in-situ refers to exposing your in-memory simulation data arrays to Catalyst (at runtime, without writing any data to disk) and then using familiar interactive Python pipelines to process and visualize data
  1. instrument your simulation code once
     - Catalyst2 is extremely diligent about avoiding data duplication in memory, passing only pointers and data descriptions through its API
  2. modify and apply your analysis/visualization pipelines without recompiling your code

VMD

Session on Mar-13

Visual Molecular Dynamics is a powerful, open-source molecular visualization program to display and analyze the results of MD simulations

- MacOS, Unix, Windows
- Initial release in 1995
- Very fast, written in C++
- Supports multi-core CPUs and GPUs
- Tcl/Tk and Python scripting
- 60+ molecular file formats and data types



Network visualization

Session on Mar-20, out of the Univ. of Ottawa

- Gephi is the most popular open-source software for 2D network exploration
- NetworkX+VTK as a large-scale alternative to Gephi in 3D
  - also interactive, but in other ways

Web-based visualization: evolution

Session on Mar-27

We will cover a little bit of Dash web application framework in Plotly workshop on Jan-30

- VTK (Visualization Toolkit)
  - C++ library, initial release in 1993
  - handles I/O, data processing, rendering
  - interpreted interface layer in Python (started in late 1990s, full API coverage by ~2018)
  - runs server-side only (C++ or Python)
- Vtk.js is a rewrite of (most of) VTK in JavaScript
  - runs client-side only (in the browser), started in 2016
  - rendering limited by the browser
  - somewhat limited processing and I/O
- Previous (pre-2021) state-of-the-art web visualization
  - standalone vtk.js applications, e.g. ParaView Glance or ParaView Divvy
  - vtk.js applications interacting with a remote ParaView server using ParaViewWeb framework, e.g.
    - ParaView Visualizer (recreating full ParaView functionality on the web)
    - ParaView Lite (lighter workflows than ParaView Visualizer)
- In all these cases you create web apps in JavaScript, or use existing apps

Web-based visualization: trame

Session on Mar-27

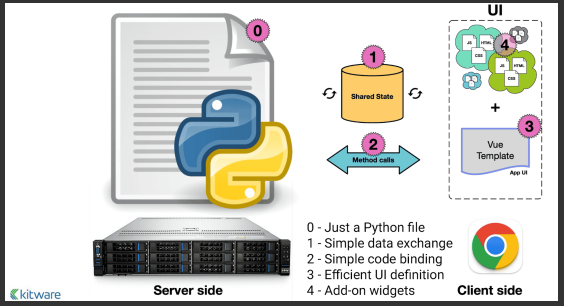


Figure: Trame architecture Source: Kitware, Inc.

Trame has plugins for VTK, Plotly, Matplotlib, Vega, ParaView, etc.

- Single backend Python code on the server (cloud, HPC, Jupyter, desktop app)
- UI (layout/buttons/...), via Vuetify toolkit on top of Vue.js, nicely abstracted
- State variables automatically synced between the backend and the frontend (UI)
- Controller actions called from (1) Python, (2) frontend, or (3) state changes

Not included in this Series

- VisIt
- Cinema, etc. for creating and viewing pre-rendered visualizations: image-based approach to large-scale visualization
- Dedicated Python geomapping libraries:
  - Cartopy
  - Geoplot (on top of Cartopy and GeoPandas)
  - Folium (on top of the Leaflet.js)
  - Rasterio for satellite imagery
- YT for analyzing and visualizing volumetric multi-resolution (AMR and unstructred) data
- Julia plotting libraries: Plots.jl, Makie.jl, AlgebraOfGraphics, etc.
- Chapel Image library
  - impressive scaling up to $32\,000^2$ images
- Photorealistic rendering: assigning materials to surfaces + ray tracing
- Using ParaView for debugging
- Any proprietary tools – covering open-source only

Also not included: visualization with AI

- Using models trained on your datasets, not merely prompting LLMs to generate visualization scripts
- Ideally, would like to load a dataset and have AI generate the best possible visualizations, while also highlighting any problems or discrepancies in the data
- Very active research area, with many identifiable subtopics:
  - visualization surrogates: from generating missing scenes/angles to replacing simulated models
  - novel view synthesis: 3D Gaussian Splatting, Neural Radiance Fields, Generative Diffusion Models
  - speeding up interactive volume rendering / ray tracing
  - super-resolution, e.g. for galaxy formation
  - feature tracking/extraction, data regression, 2D/3D imaging data segmentation
- No automated solutions to recommend at this point ...

Next talk (Matplotlib) at 11:00am Pacific in this room

Questions?