ParaView User's Guide[1]: The Summary

Introduction to ParaView

1.1.1 Introduction

ParaView started in *2000*, as a collective effort between Kitware Inc. and LANL (Los Alamos National Laboratories). First *Public Release* was *version 0.6* in *October 2002*.

ParaView is intended for extremely large datasets, and is general-purpose, providing a wide range of possibilities working with it:

- personal desktops,
- remote parallel computing,
- web visualization (`ParaView Web`),
- in-situ analysis (`Catalyst`),
- scripting (`Python`)

`ParaView` is *open source* (BSD licenses, commercial software friendly).

1.1.2 Basics of visualization in `ParaView`

Visualization – process of converting raw data to images. `ParaView` uses the `Visualisation Toolkit – VTK` for the backbone in visualization and data processing.

Different types of ***objects*** **(1)** exist:

- data objects,
- process objects,
- source objects (*data readers*)
- filter objects (*intermediate*),
- sink objects (*data writers)*,
- mapper objects

The `VTK` model is based on the data-flow paradigm, where initial data – the *source* – is inserted to a *processing pipeline* and is transformed at each step by *modules* (various algorithms, which are referred to as ***filters*** **(2)** (pipeline modules) in `ParaView`).

The *filters* that can be applied to existing source, depend on the data type of the current source.

Note that original source might change its underlying type multiple times based on previous filters applied. According to ParaView User's Guide: *"When dealing with structured data, it is absolutely important to know what filters will change the data to unstructured. Unstructured data has a much higher memory footprint, per cell, than structured data because the topology must be explicitly written out."*[2]. This is crucial to volume rendering, as unstructured data is orders of magnitude slower). Filters that change the data type are listed in [2].

1.1.3 ParaView executables

- `paraview`

  Main Graphical User Interface (GUI). It is `Qt`-based and cross-platform.

- `pvpython`

  `Python` interpreter that runs `ParaView`'s `Python` *interactive* scripts

- `pvbatch`

  `Python` interpreter that runs `ParaView`'s `Python` *batch processing* scripts, can run in-parallel (when running on machines with `MPI` capabilities).

- `pvserver`

  Remote visualization support. `paraview` can be made to connect to `pvserver` that runs remotely on an HPC resource and used on a local desktop as if it was running locally.

- `pvdataserver` and `pvrenderserver`

  Remote visualization support. `paraview` can be made to connect to `pvdataserver` for remote data processing, and `pvrenderserver` for remote visualization, running on separate nodes with appropriate computing capabilities for each task.

1.1.4 Getting started with `paraview`

`paraview` graphical user interface

Panels, to peek into the application state:

- Pipeline Browser (hierarchical pipeline inspection)
- Memory Inspector (see how `ParaView` uses out of system total)
- Properties (see options for interaction/status of ***objects* (1)**)
- Viewport (area where `ParaView` renders the generated results to)
- *List can be expanded, since there are more panels*

Menus include:

- Files (open/save files)
- Edit
- View (toggling which panels are visible)
- Sources (test sample data of various types
- Filters (list of algorithms for intermediate data processing)
- Extractors (extract common data formats, such as `CSV`, `JPG` etc.)
- Tools (advanced features, plugin management, scripting, favourites)

Example of displaying data (3)

Creating an example source (such as a sample Sphere) can be done from `Sources` → `Sphere`, which: **1)** adds a module to the Pipeline Browser,
2) populates the Properties panel
and
3) highlights the Apply button in the Properties Panel

After clicking that Apply button, then the Viewport shows the Sphere based on properties chosen by the user from the Properties Panel at the time of clicking the Apply button

NOTE: this process applies to user-imported data as well

Changing Properties

Changing underlying properties of the filter will highlight the Apply button again, and not perform an automatic viewport (rendering) update[1]. To update the rendering, the user will have to press the Apply button. However, when *Display* options are updated, the Apply button is not affected and the view updates automatically.

This is mainly because execution of the filter is usually more computationally expensive than the rendering. Changing the *Display* options essentially just triggers a render with an updated graphics state.

Applying Filters

First, make sure that a module in the Pipeline Browser is selected (otherwise no Filters will be available and print a message *"Requires an input")*

Then, in the *Filters* menu (`Filters` → `Alphabetical` etc.) some filters will be greyed out, and some available, based on the module/data source selected. Filters and their meanings will be discussed in much greater detail later.

[1] Automatic *Apply* after changing properties can be enabled in `Edit` → `Settings` → ☑ `Auto Apply`

If we choose a *Shrink* filter to execute on our existing *Sphere* (by selecting `Filters → Alphabetical → Shrink`), we notice that a new module (called *Shrink1*) appears in the *Pipeline Browser* and that the *Apply* button is again highlighted (with some default parameters in the *Properties* panel). As before, when the *Apply* button is clicked, the filter output is visible (and the result from *Sphere1* is hidden by default to avoid overlapping). To make modules visible or hidden, we can click on the *eyeball* icons in the *Pipeline Browser*.

NOTE: Common User Error is forgetting to hit *Apply* after creating sources and filters

1.1.5 Getting started with `pvpython*`

* also applies to pvbatch, except that `pvpython` has an interactive shell provided, and `pvbatch` excepts the script to be specified on the command line argument

`pvpython` scripting interface

`ParaView` provides an interface to write scripts for performing same tasks that can be done on the GUI. This can be opened from the Menu: `View → Python Shell`

Alternatively, it is a stand-alone Python Shell (on a personal machine, it can be found as a stand-alone application `pvpython` with a *version name* attached.

NOTE: A comprehensive scripting walkthrough is provided in the original User Guide. Rather than summarising here, we refer the reader to follow it instead.

1.1.6 Scripting in `paraview`

This subsection refers to the shell within `ParaView` that we can invoke from `View → Python Shell`

It works the same way, however, is coupled with the GUI. When the commands are run, the modules will appear in the *Pipeline Browser* etc.

Additionally, it has auto-completion (by pressing TAB).

Tracing actions for scripting

`ParaView` supports explicit scripting – that is, following user's actions and generating the script afterwards.

This can be accessed from `Tools → Start Trace`

Then, `ParaView` follows user's actions relevant for the script, until `Tools → Stop Trace` is clicked.

When the trace is stopped, `ParaView` shows the generated script in a `ParaView Python Script Editor` Window.

This script can be saved to use for batch processing (`pvbatch`) – it expects code to be specified as command line argument.

Saving ParaView State File

Alternatively, to capture the visualisation exactly as it looks like in the application (the Python script is not guaranteed to produce a one-to-one accurate output), the user can save the state to a text file. This does not require tracing, and is done with `File → Save State`.

1.2 Filtering Data

1.2.1 Understanding Filters

Filter – pipeline module with inputs and outputs

NOTE: when filter is applied, its output might not match the input type of the original source data

1.2.2 Creating Filters in `paraview`

There is a dedicated Filters menu, where Filters are also categorized (can view them in alphabetical list as well, to see all).

Each Filter accepts distinct input types, therefore, some might be unavailable (greyed out). In this case, the reason is displayed at the User Interface (at the bottom) when the filter is hovered on with a cursor.

Multiple input ports

Some `paraview` filters have multiple input ports. If that is the case, a UI menu will pop up to select the relevant inputs. In `pvpython`, the filter inputs will need to be provided explicitly.

References

[1] Paraview User's Guide: https://docs.paraview.org/en/latest/UsersGuide/index.html

2020, ParaView Developers Revision

[2] Docs » Paraview User's Guide » 5. Filtering Data – 5.12.1. Avoiding data explosion: https://docs.paraview.org/en/v5.10.1/UsersGuide/filteringData.html

2020, ParaView Developers Revision