

# Web Visualization

Sebastien Jourdain @ Kitware

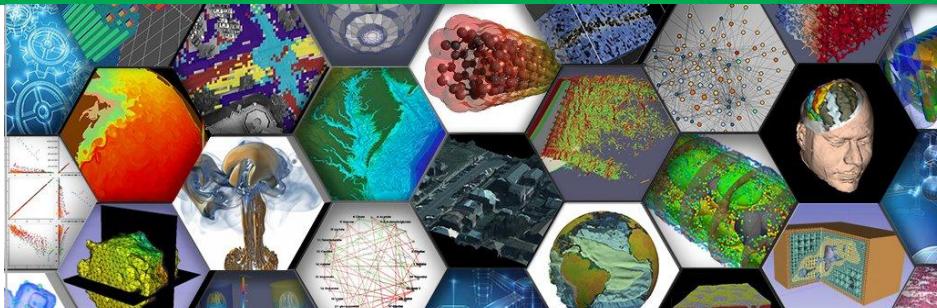
# For tutorial

Make sure you have Python 3.6+  
Python 3.9 is mandatory for ParaView

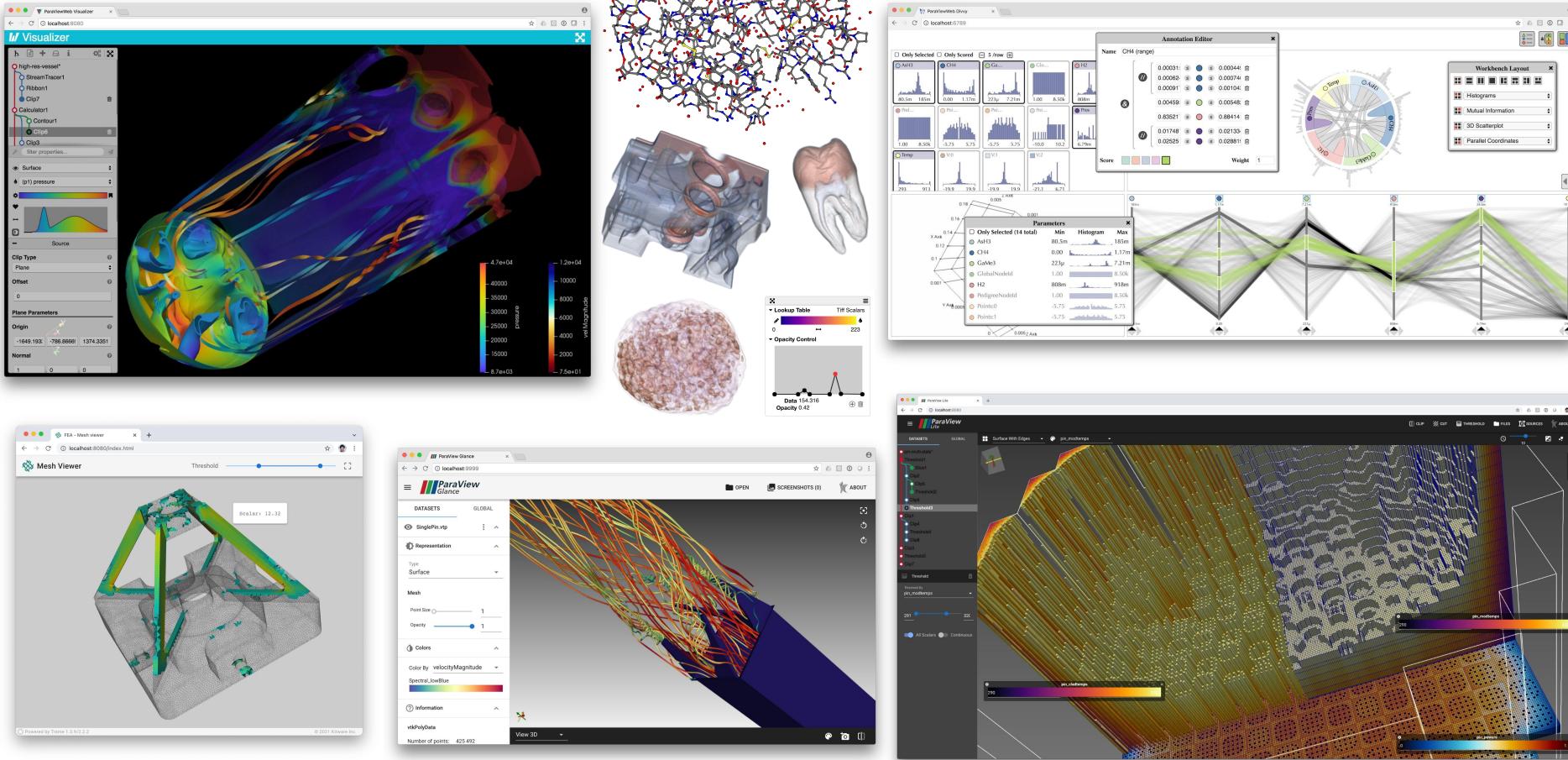
# Kitware - Open Source softwares

Five core areas of expertise

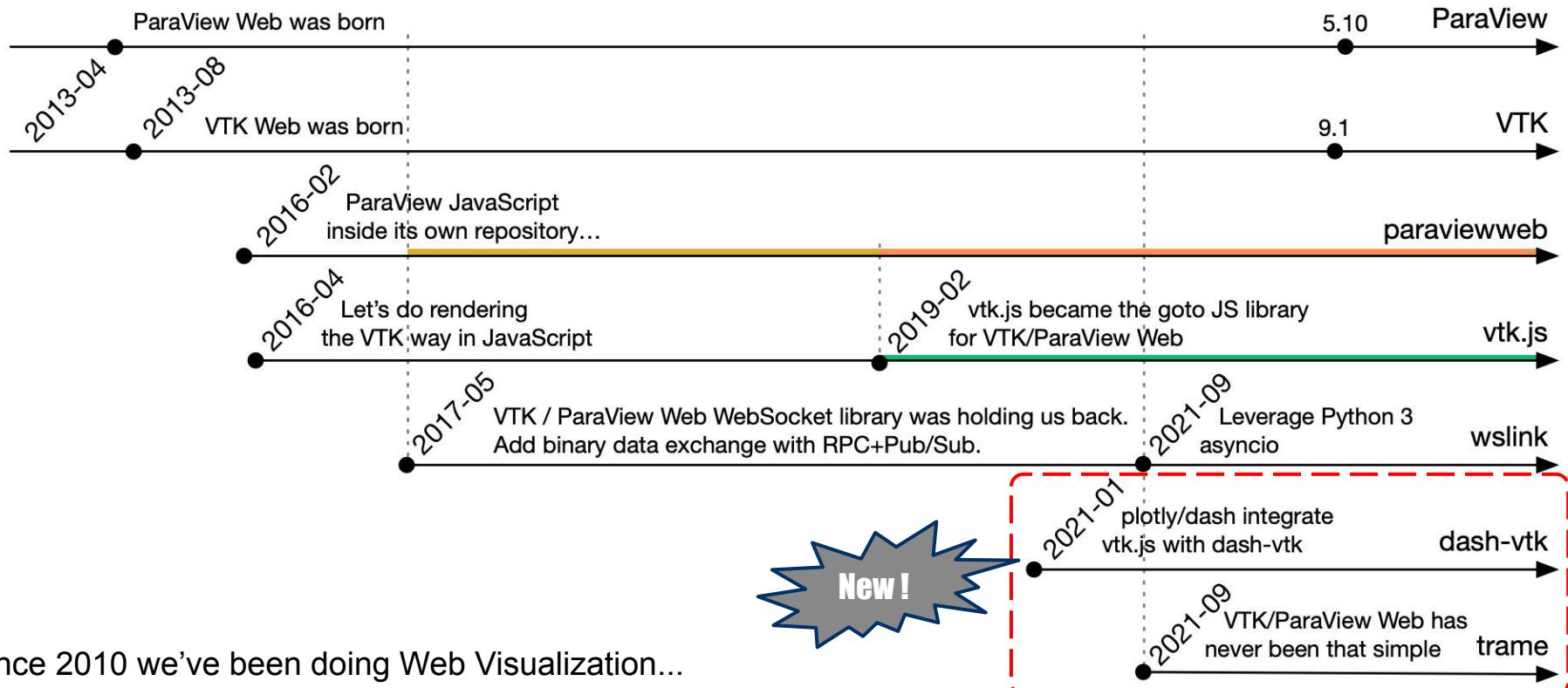
- Computer Vision
- Data and Analytics
- HPC and Visualization
- Medical Computing
- Software Process



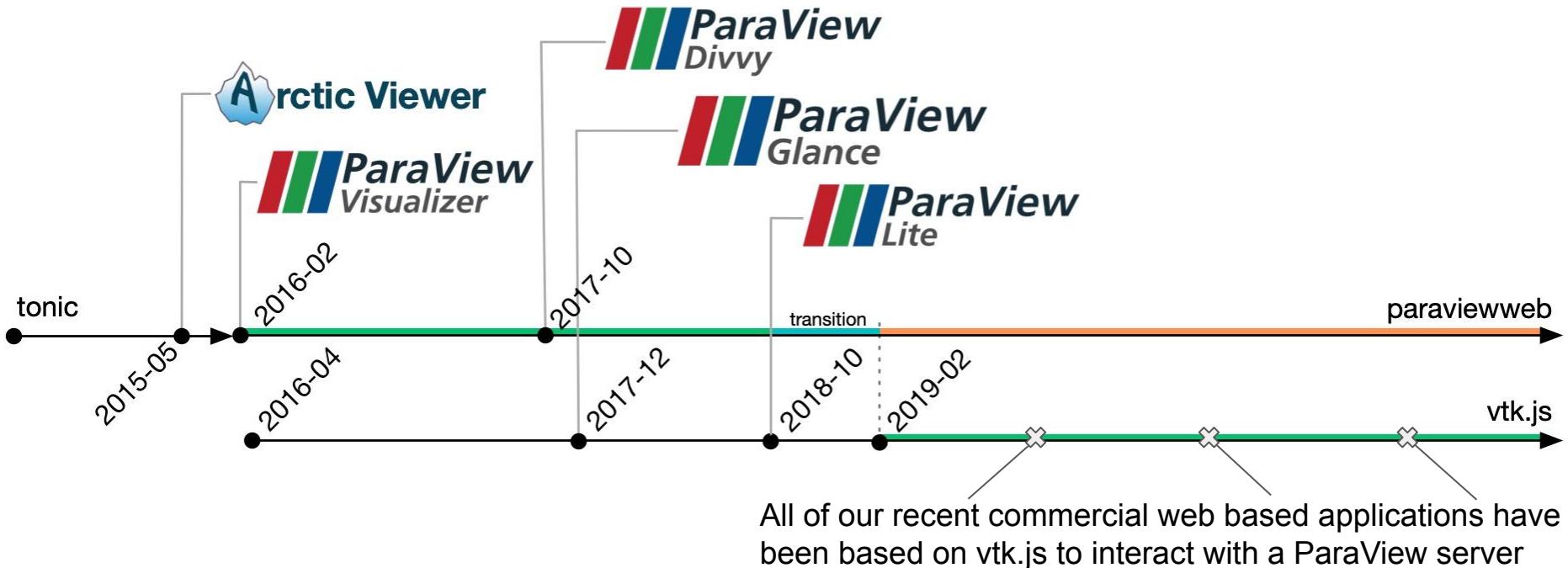
# Visualization in the Web



# History and background



# Web Visualization Applications/Demonstrators

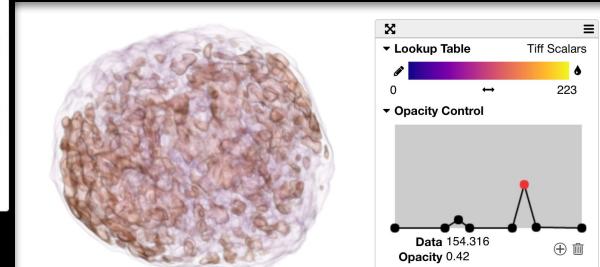
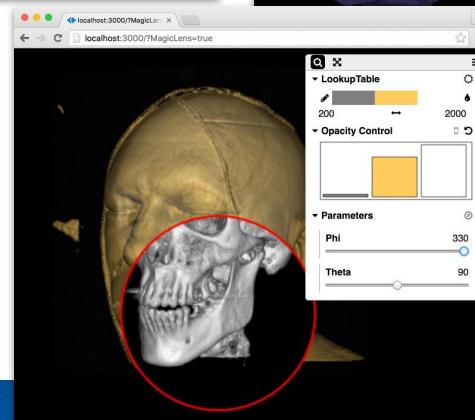
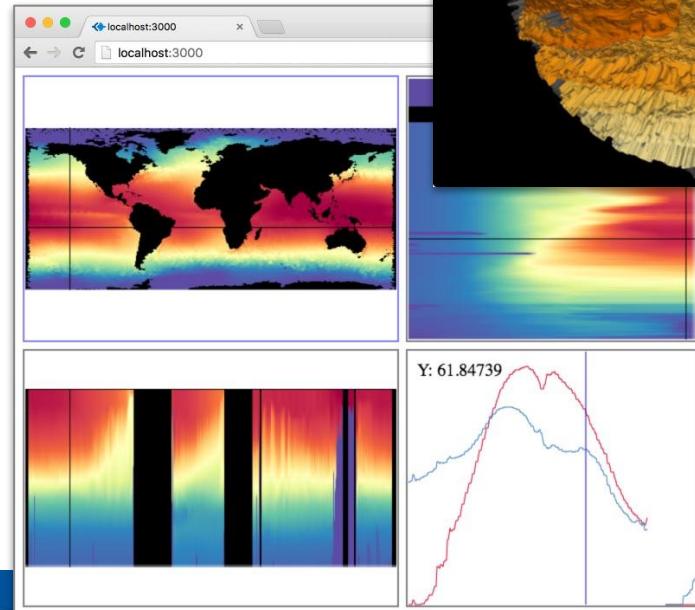
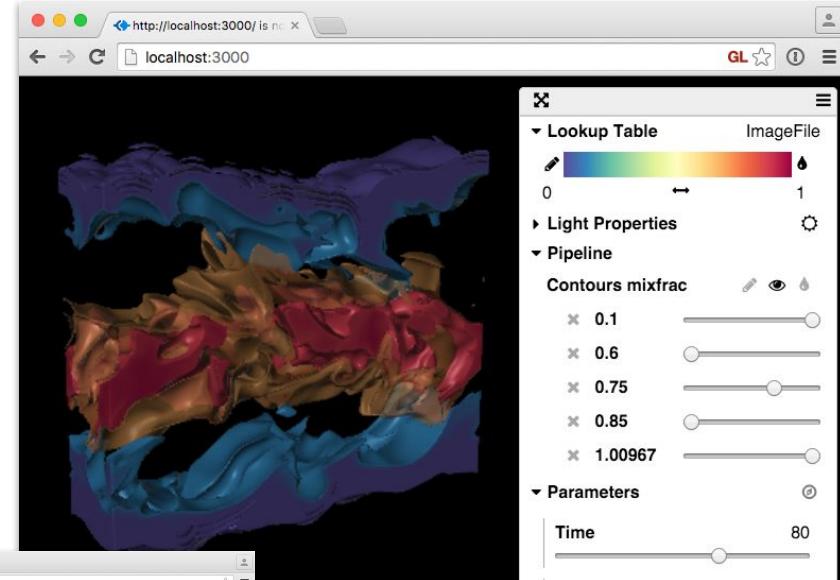
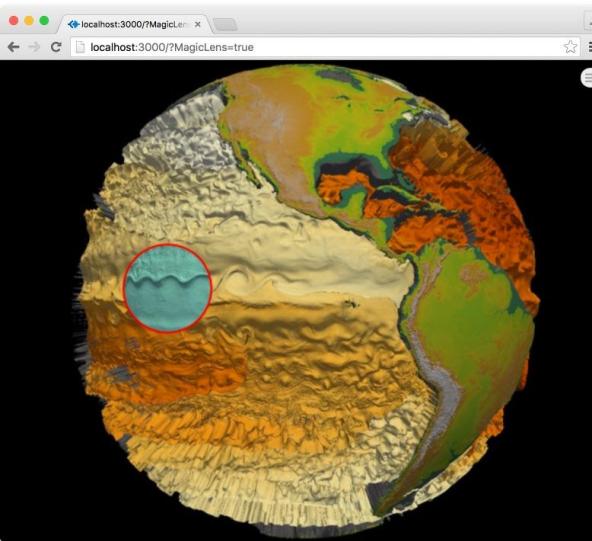
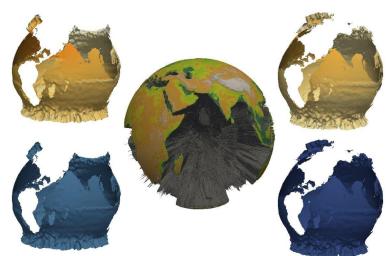


# Applications / Demonstrators

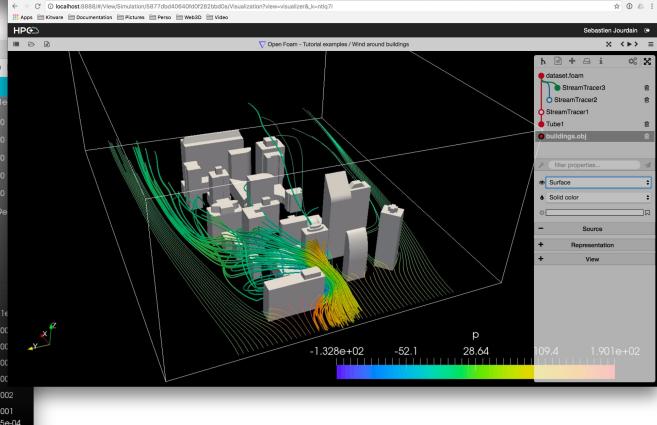
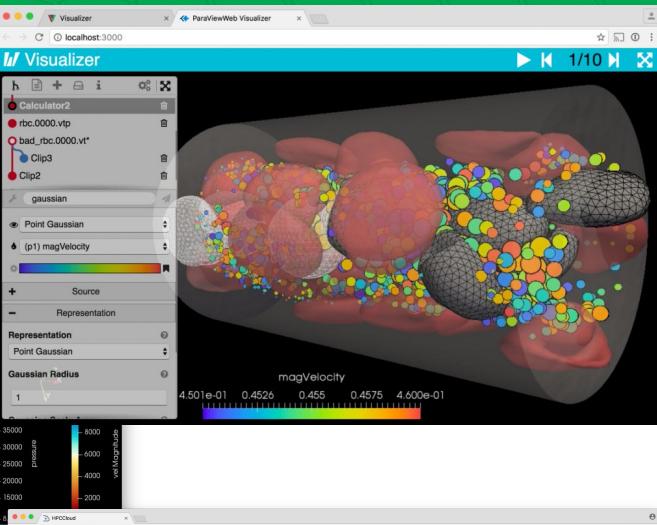
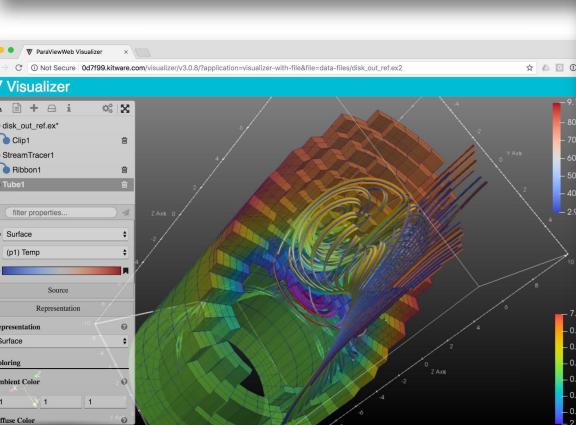
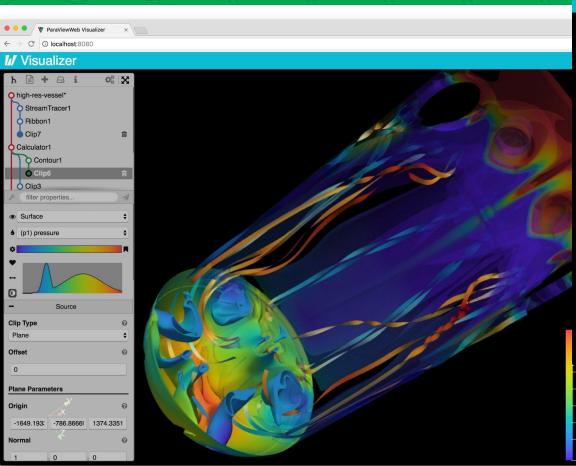
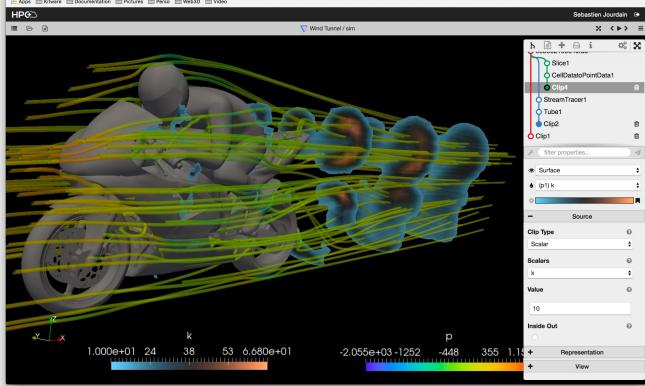
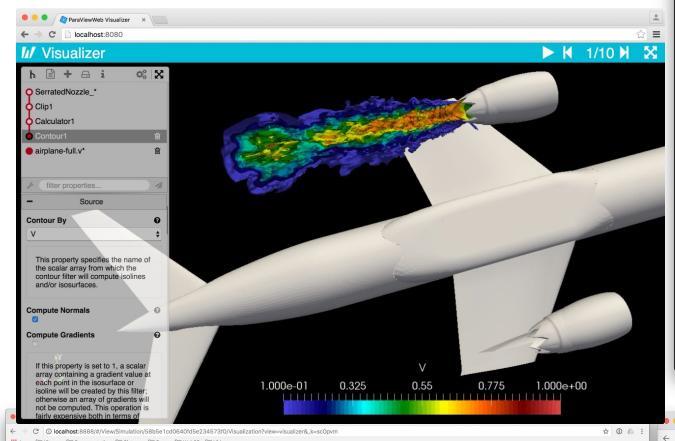
- Arctic Viewer (2015)
  - Fast and interactive viewer for ParaView based pre-processed assets
- ParaView Visualizer (2016)
  - The ParaView like application in the Web
- ParaView Divvy (2017)
  - Interactive data analytic with ParaView that does not look like ParaView
- Glance (2018)
  - The ultimate vtk.js viewer in a serverless fashion
- ParaView Lite (2018)
  - A fresh look at what ParaView in the Web could look like



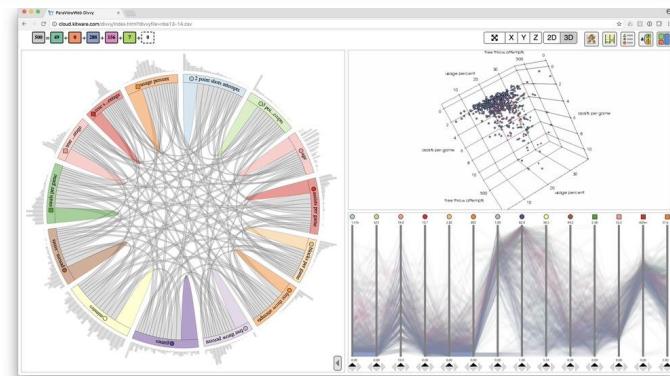
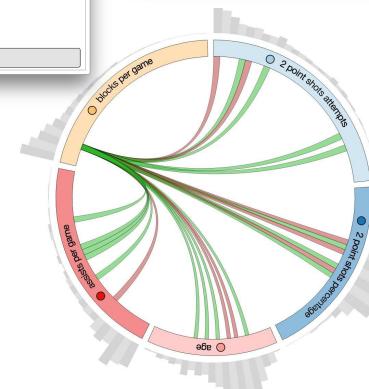
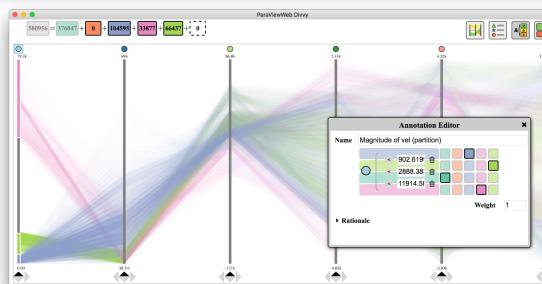
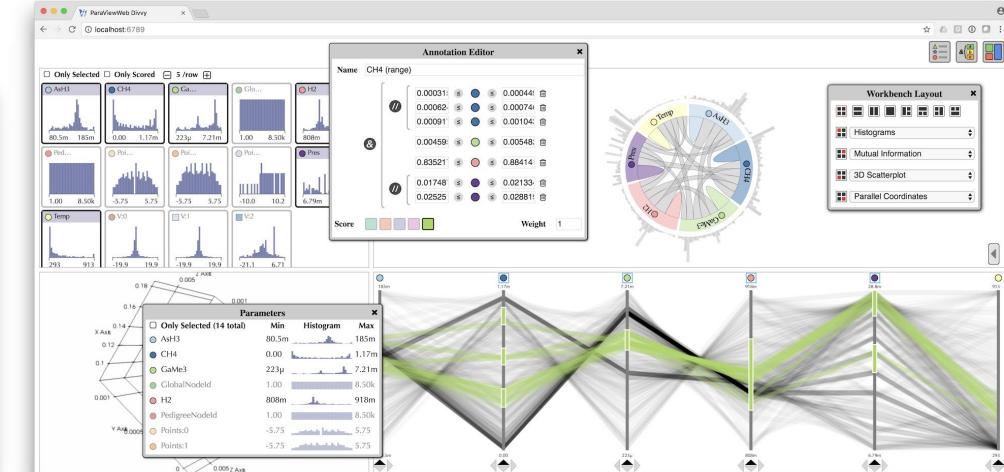
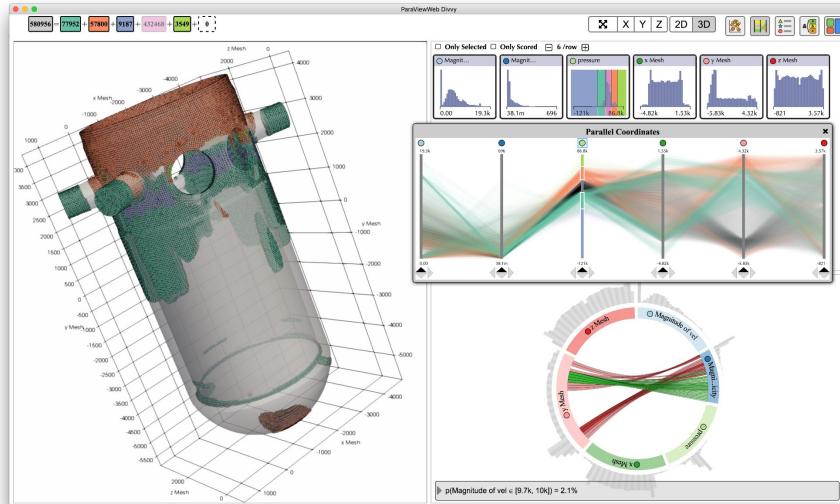
# Fast and interactive viewer for ParaView/Catalyst based pre-processed assets



# The ParaView like application in the Web

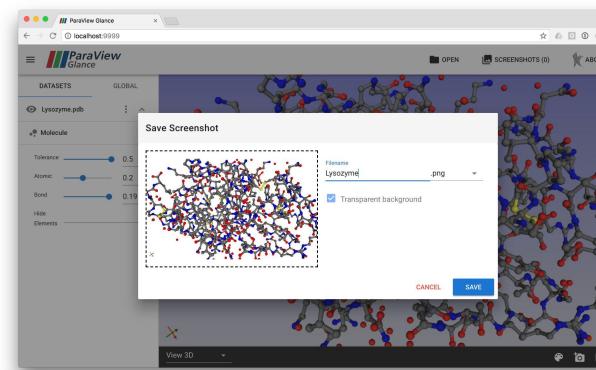
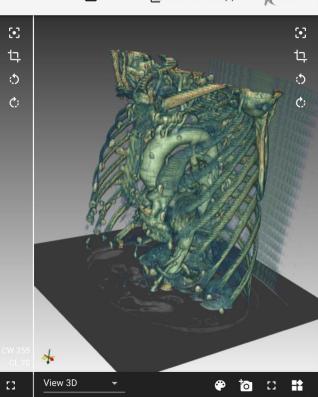
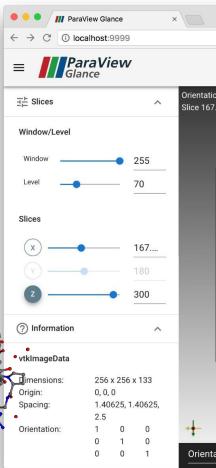
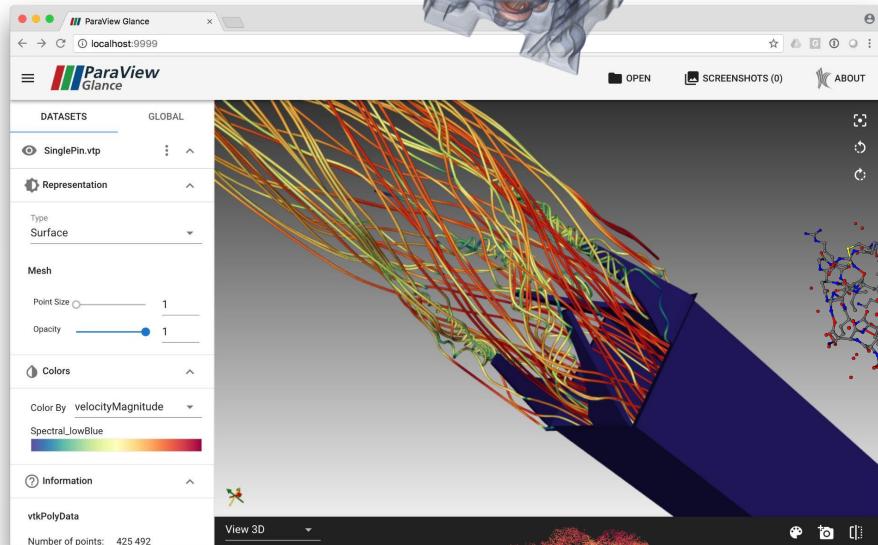


# Interactive data analytic with ParaView that does not look like ParaView

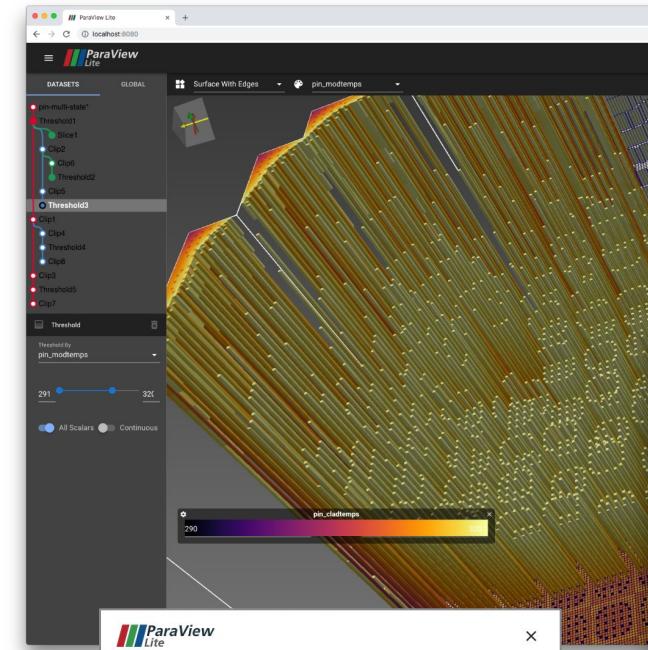


# The ultimate vtk.js viewer in a serverless fashion

 **ParaView  
Glance**



# A fresh look at what ParaView in the Web could look like



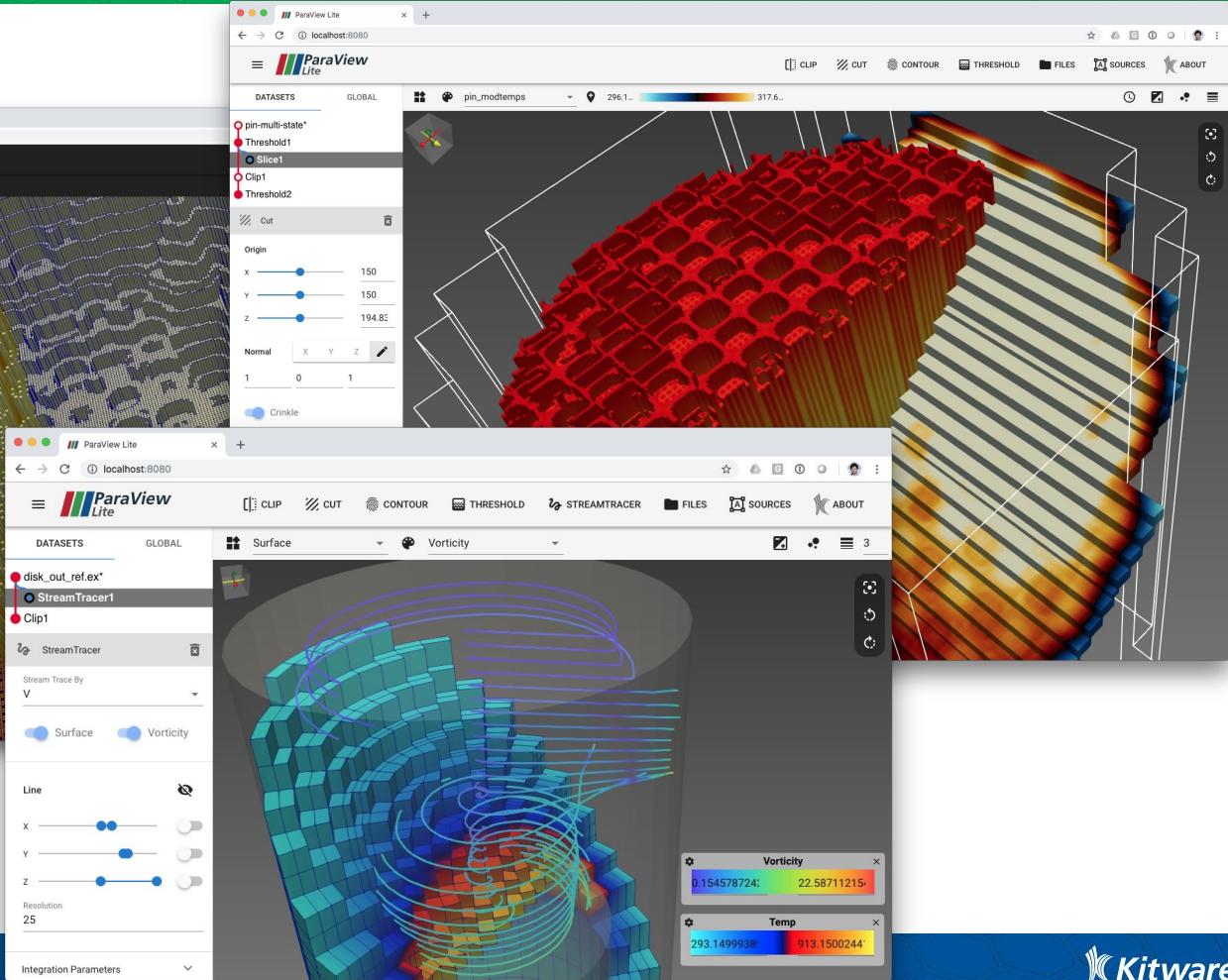
**ParaView Lite**

ParaView Lite is an open-source web application developed at [Kitware](#) for remote visualization. It is part of the [ParaView](#) platform and can serve as a foundation for building custom web-based visualization applications involving [ParaViewWeb](#).

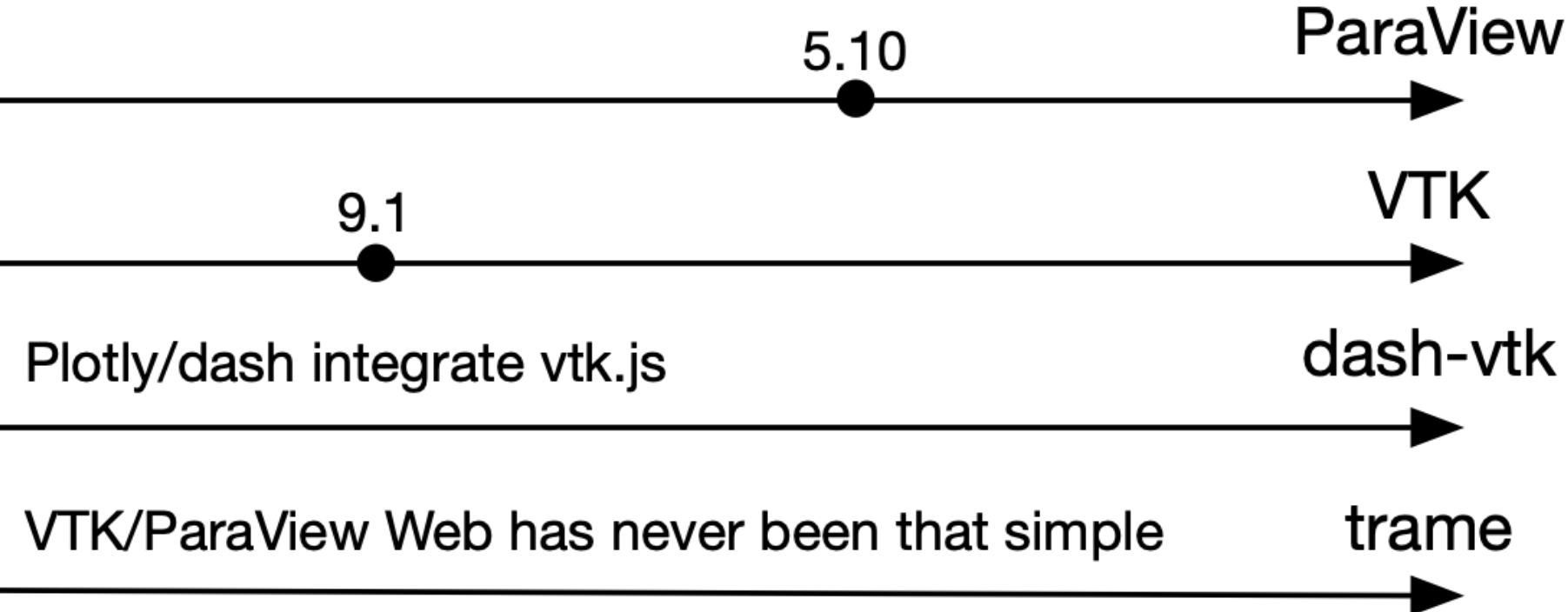
- Here is our [documentation](#).
- We are open source! [Code is on Github](#).
- Current version: Development

**Kitware** **ParaView**

© 2018 – Kitware, Inc.



## What's new?



## ParaView (5.10) and VTK (9.1) releases

They both include the latest version of ***wslink***

- Python 3.6+ with ***asyncio*** and ***aiohttp***

They both include improvements and fixes for their Web protocols

- Mouse wheel handling
- Support for vtkRenderWindow synchronization with vtk.js for
  - Volume Rendering, Cube Axes, Scalar Bar Actor

# Plotly/Dash add support for 3D Viz with vtk.js

The screenshot shows the 'Dash Python > Overview' page. On the left, there's a sidebar with navigation links like 'What's Dash?', 'Dash Tutorial', etc. The main content area has tabs for 'Python', 'FORUM', 'GALLERY', and 'DASH CLOUD'. Below the tabs, there's a heading 'Dash VTK' with a Python logo. The text explains what VTK is and how Dash VTK integrates it. A 'User Guide' section follows, with a red arrow pointing from the 'Advanced Demos' link in the sidebar to the 'Advanced Demos' section in the main content. This section is highlighted with a red border and contains links for 'Intro to 3D Visualization', 'Structure of Datasets', 'Representation Components', 'Other Dash VTK Components', and 'Advanced Demos'.

The screenshot shows the 'Advanced Demos | Dash for Py' page. It features a 'Seeds' panel on the left with sliders for 'Seed line' and 'Line resolution', and dropdowns for 'Color By' (Field name: p) and 'Color Preset' (erdc\_rainbow\_bright). To the right is a 3D visualization of a skull with a complex mesh and colored lines representing data fields. Below the visualization, there's a heading 'Medical examples' and a 'Real medical image' section showing a 3D rendering of a skull.

The screenshot shows a browser window titled 'Other Dash VTK Components' with the URL 'dash.plotly.com/vtk/other'. The page displays a large block of Python code for a Dash VTK component. The code uses the dash\_vtk module to create a 3D visualization. It includes imports for os, dash, dash\_html\_components, dash\_vtk, and specific components like GeometryRepresentation and ConcentricCylinderSource. The code defines a content block with various parameters for the visualization, such as color mapping, scalar mode, and interpolation settings. At the bottom, it shows the standard Dash setup with app.run\_server(debug=True).

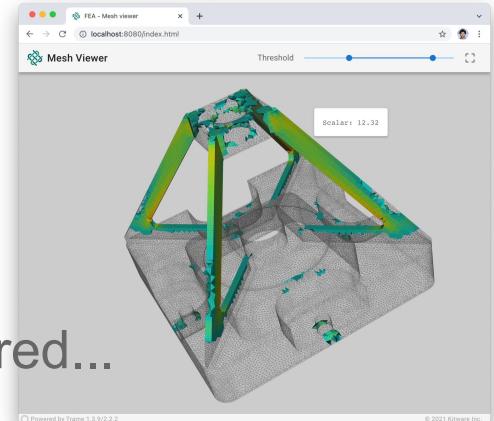
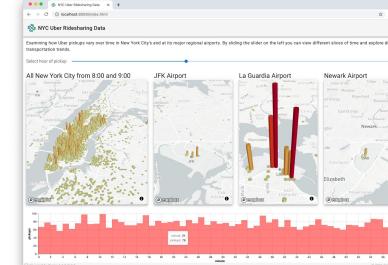
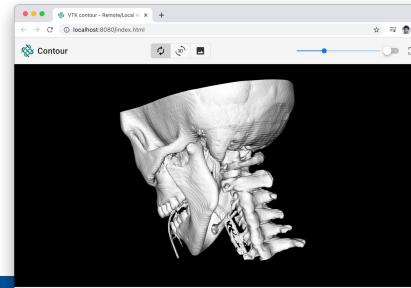
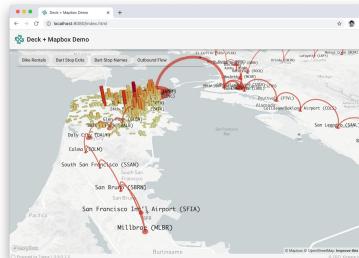
# trame - VTK/ParaView Web has never been that simple!

trame provides the frame for building your next Web application

- Python is all you need to know

Make your first Web app today - (tutorial coming next)

- Rich set of UI components
- Play well with VTK 9.1 and ParaView 5.10
- Charts, 3D maps or 3D views, trame has you covered...



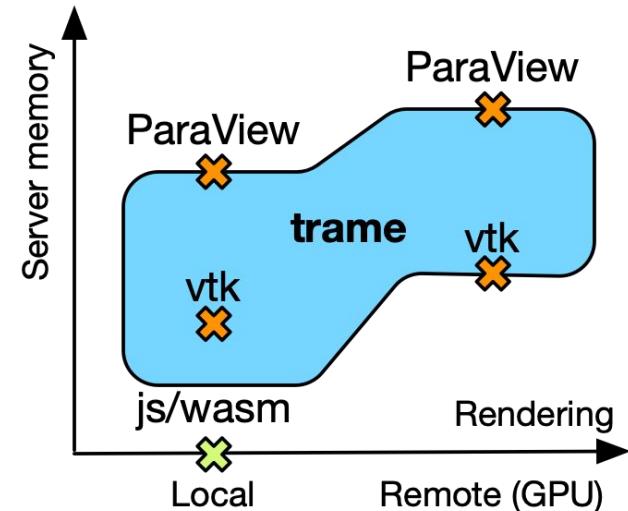
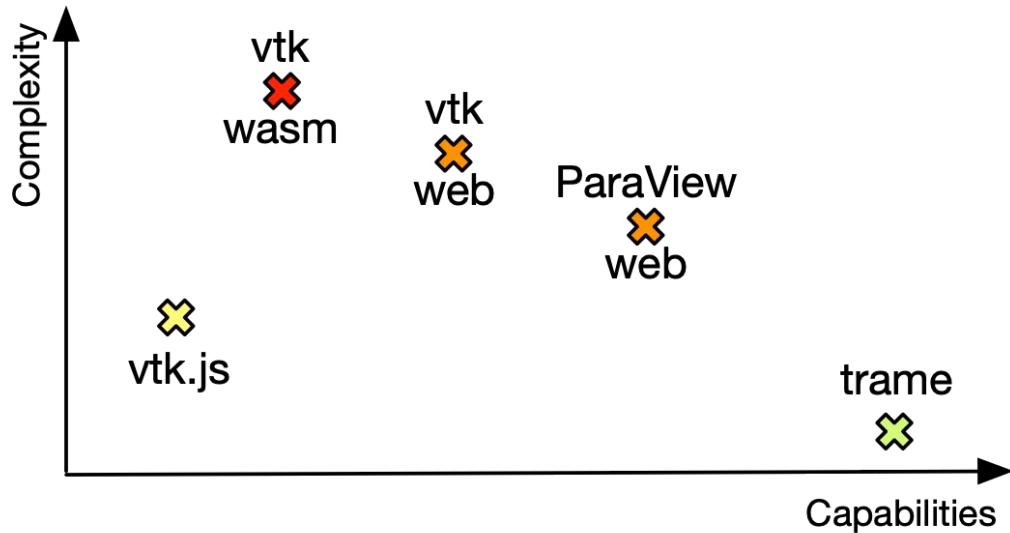
```
pip install "vtk>=9.1.0"  
pip install "trame"
```

# But wait a minute...

Why do we have so many options?

# Blend of needs and constraints

- Client vs client+server
- Resources (knowledge, hardware, operating cost...)



# Tutorial - trame

<https://kitware.github.io/trame/docs/tutorial.html>



# simple, powerful, innovative

**trame** - a web framework that weaves together source components into customized visual analysis.

Getting Started

- Introduction
- How to start
- API

**Tutorial**

- Overview
- Download
- Setup for VTK
- VTK
- Layouts
- HTML
- Application
- ParaView

Html Modules

- vuetify
- vega
- markdown
- widgets
- vtk
- paraview
- deckgl

How do I...

- Display static content
- Display dynamic content
- Display interactive UI
- Display a dataframe

Documentation

**Tutorial of trame**

We walk you through the basics of **trame** and end up creating a simple but full-featured VTK example application.

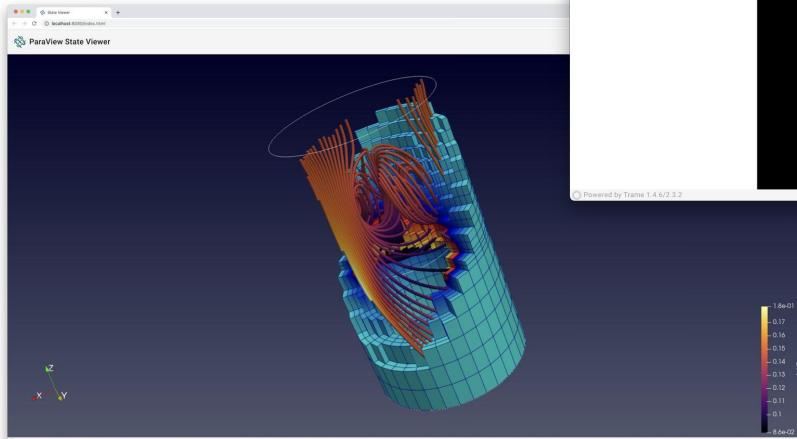
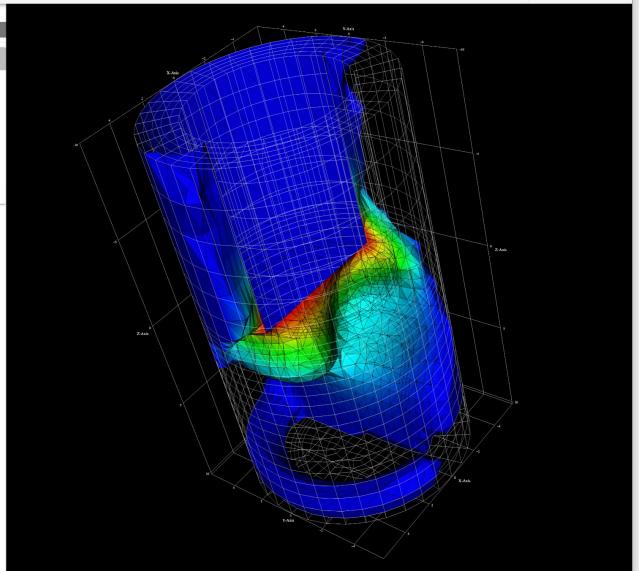
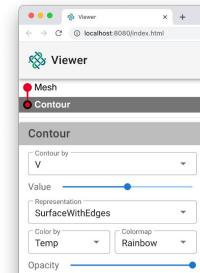
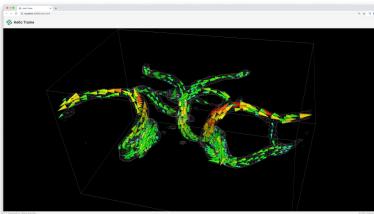
It is easiest to work on the tutorial from within the **trame-tutorial** repository. The tutorial are broken down into the following ordered sections:

1. Download
2. Setup
3. VTK
4. Layouts
5. HTML
6. Application
7. ParaView

◀ PREV      Last updated: 2021-11-16      NEXT ▶

# Tutorial

- Download code with exercises
- Setup your Python environment
- VTK for doing 3D visualization
- Trame UI
  - Layouts
  - HTML
- Application
- ParaView



# Download exercises

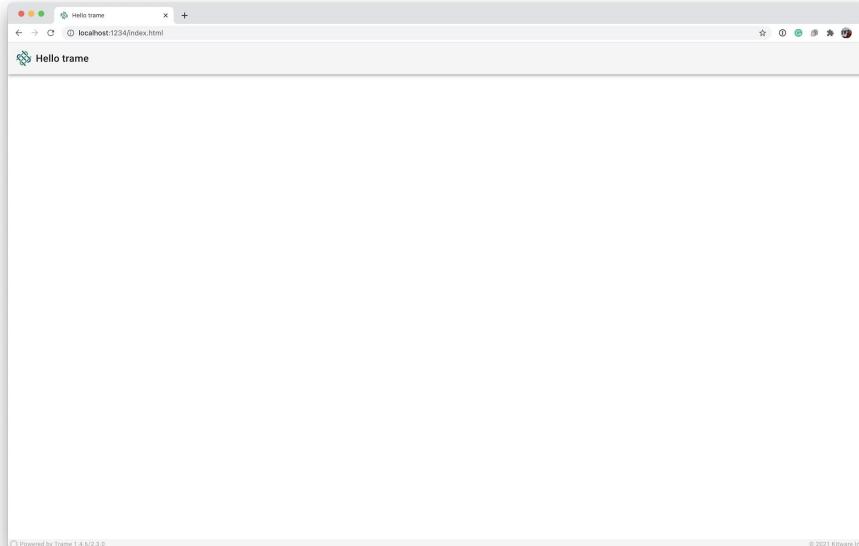
```
git clone https://github.com/Kitware/trame-tutorial.git
```

## Setup a Python virtual environment

```
cd trame-tutorial
python3.9 -m venv .venv
source ./venv/bin/activate
python -m pip install --upgrade pip
pip install "trame"
pip install "vtk>=9.1.0"
```

# Run your first trame application

```
$ python ./00_setup/app.py --port 1234
```

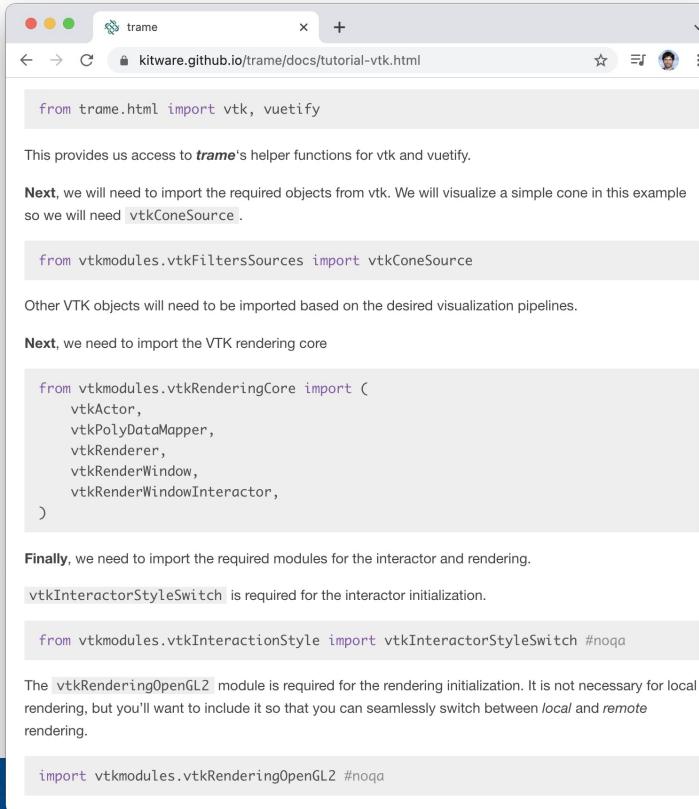
A screenshot of a code editor window titled "app.py". The code is as follows:

```
from trame.layouts import SinglePage
# -----
# GUI
# -----
layout = SinglePage("Hello trame")
layout.title.set_text("Hello trame")
# -----
# Main
# -----
if __name__ == "__main__":
    layout.start()
```

The code is written in Python, utilizing the trame library for creating a web-based application. The code defines a single-page layout with the title "Hello trame" and starts it when run.

# VTK inside trame

Start editing the file in `01_vtk/app_cone.py` which has the same content as `00_setup/app.py`.



The screenshot shows a web browser window titled "trame" displaying the URL `kitware.github.io/trame/docs/tutorial-vtk.html`. The page content is as follows:

```
from trame.html import vtk, vuetify
```

This provides us access to `trame`'s helper functions for vtk and vuetify.

**Next**, we will need to import the required objects from vtk. We will visualize a simple cone in this example so we will need `vtkConeSource`.

```
from vtkmodules.vtkFiltersSources import vtkConeSource
```

Other VTK objects will need to be imported based on the desired visualization pipelines.

**Next**, we need to import the VTK rendering core

```
from vtkmodules.vtkRenderingCore import (
    vtkActor,
    vtkPolyDataMapper,
    vtkRenderer,
    vtkRenderWindow,
    vtkRenderWindowInteractor,
)
```

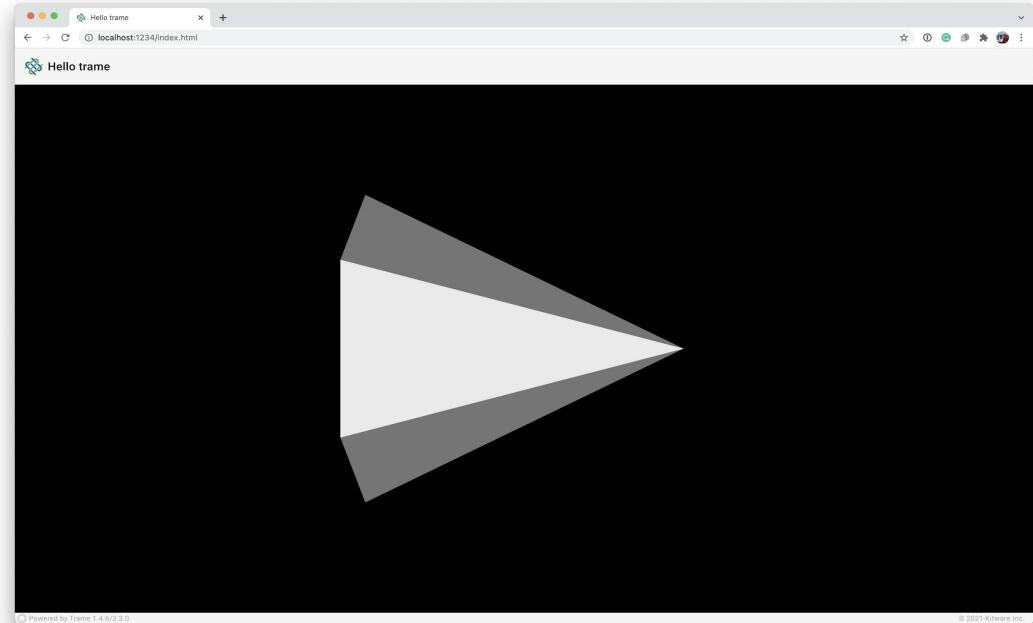
**Finally**, we need to import the required modules for the interactor and rendering.

`vtkInteractorStyleSwitch` is required for the interactor initialization.

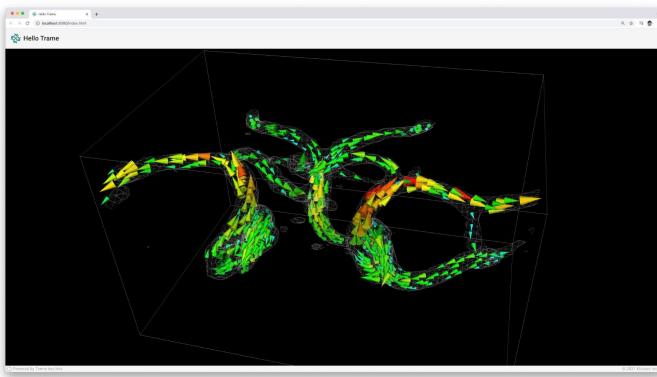
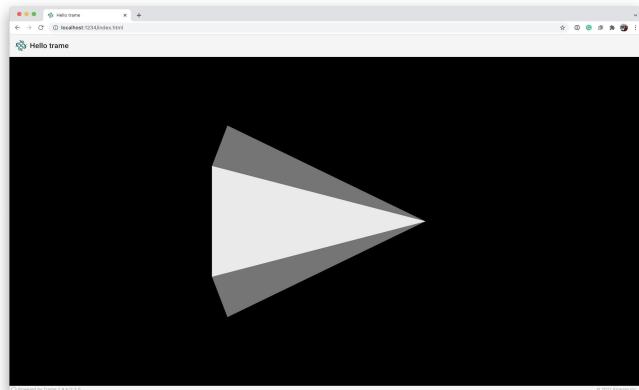
```
from vtkmodules.vtkInteractionStyle import vtkInteractorStyleSwitch #noqa
```

The `vtkRenderingOpenGL2` module is required for the rendering initialization. It is not necessary for local rendering, but you'll want to include it so that you can seamlessly switch between `local` and `remote` rendering.

```
import vtkmodules.vtkRenderingOpenGL2 #noqa
```



# Coding time...



```
solution_cone.py
```

```
4  from vtkmodules.vtkFiltersSources import vtkConeSource
5  from vtkmodules.vtkRenderingCore import (
6      vtkActor,
7      vtkPolyDataMapper,
8      vtkRenderer,
9      vtkRenderWindow,
10     vtkRenderWindowInteractor,
11   )
12
13 # Required for interactor initialization
14 from vtkmodules.vtkInteractionStyle import vtkInteractorStyleSwitch # noqa
15
16 # Required for rendering initialization, not necessary for
17 # local rendering, but doesn't hurt to include it
18 import vtkmodules.vtkRenderingOpenGL2 # noqa
19
20
21 # -----
22 # VTK pipeline
23 #
24
25 renderer = vtkRenderer()
26 renderWindow = vtkRenderWindow()
27 renderWindow.AddRenderer(renderer)
28
29 renderWindowInteractor = vtkRenderWindowInteractor()
30 renderWindowInteractor.SetRenderWindow(renderWindow)
31 renderWindowInteractor.GetInteractorStyle().SetCurrentStyleToTrackballCamera()
32
33 cone_source = vtkConeSource()
34 mapper = vtkPolyDataMapper()
35 mapper.SetInputConnection(cone_source.GetOutputPort())
36 actor = vtkActor()
37 actor.SetMapper(mapper)
38
39 renderer.AddActor(actor)
40 renderer.ResetCamera()
41
```

Line 58, Column 1      master      Spaces: 4      Python

# Local and Remote rendering

## Local Rendering

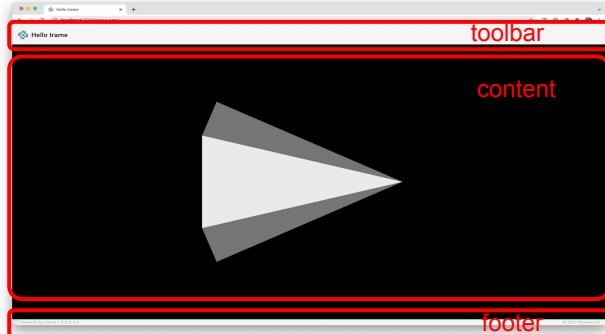
- Advantages
  - The server doesn't need a GPU
  - Interacting with the scene is as fast as it can get
- Disadvantages
  - The geometry needs to be send to the client. This can be big and therefore slow.
  - The client rendering capabilities is not known

## Remote Rendering

- Advantages
  - No data movement except the rendered image
  - Rendering can use parallel and distributed processing
  - Application can serve a more diverse set of client (phone)
- Disadvantages
  - The framerate will be capped by the speed and latency of the network (~20-30 fps).
  - The server needs a GPU for faster rendering

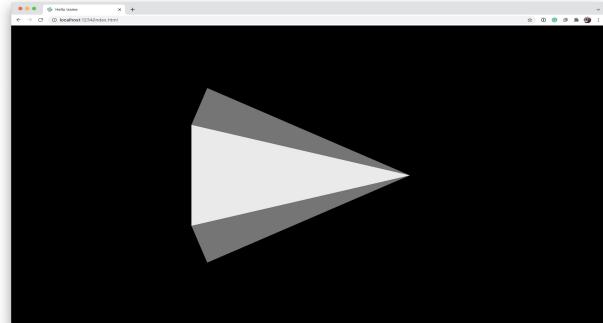
# Layouts

SinglePage



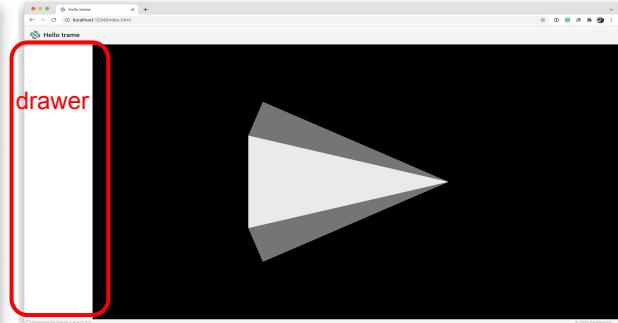
- toolbar
  - logo
  - title
  - content
  - footer
- Inside toolbar

FullScreenPage



- children

SinglePageWithDrawer



- toolbar
- logo
- title
- content
- footer
- drawer

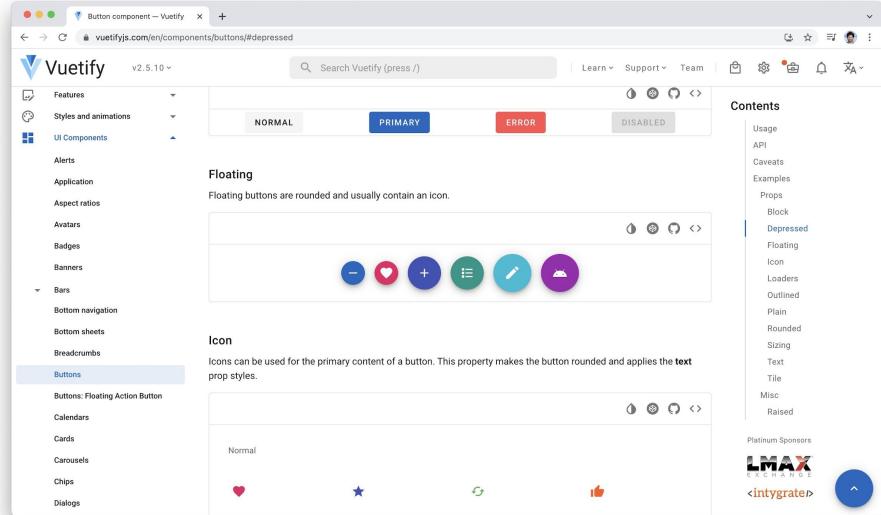
# HTML

- Vuetify (-:@) => (\_)
- Shared state / actions



A screenshot of a Vue.js application interface. On the left, there is a horizontal slider with a blue track, a red handle, and a small circular icon next to it. To the right of the slider is a blue button with a white icon. A red box highlights the entire slider component. Below the slider, the corresponding Vue.js code is shown:

```
vuetify.VSlider(  
  v_model="resolution", DEFAULT_RESOLUTION),  
  min=3,  
  max=60,  
  step=1,  
  hide_details=True,  
  dense=True,  
  style="max-width: 300px",  
)  
  
@change("resolution")  
def update_resolution(resolution, **kwargs):  
  cone_source.SetResolution(resolution)  
  html_view.update()
```

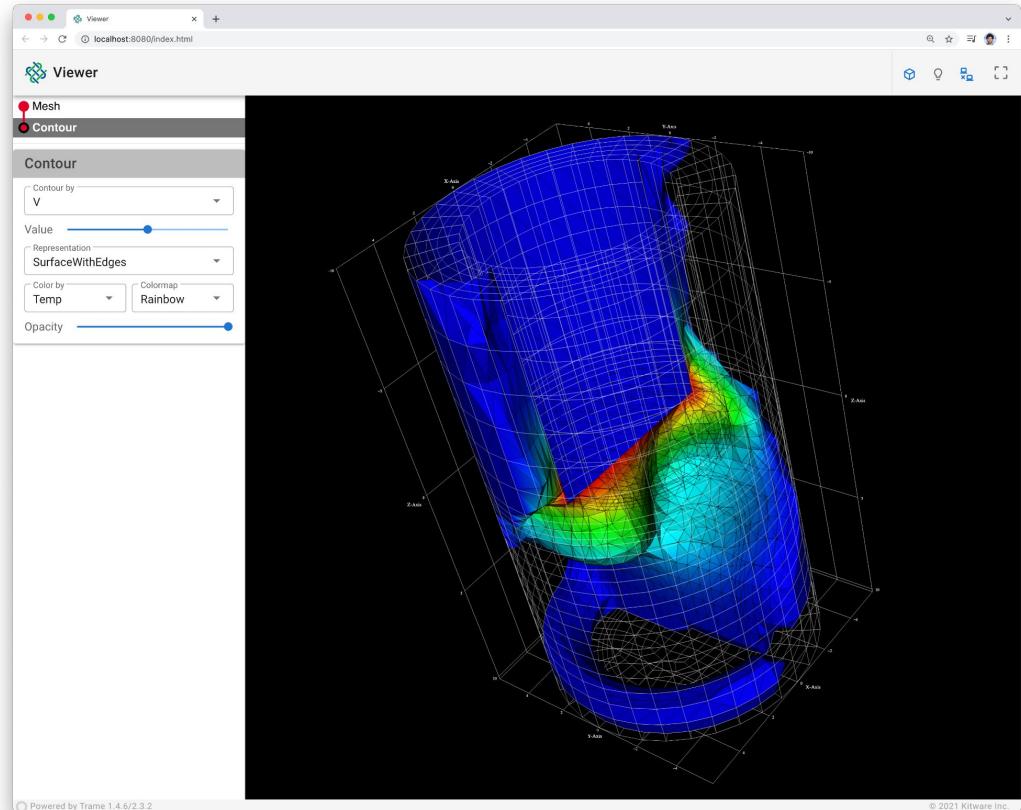


```
def reset_resolution():  
  update_state("resolution", DEFAULT_RESOLUTION)
```

# Coding time...

# Application

- VTK pipeline
  - CubeAxes
  - Mesh (unstructured grid)
  - Contour (filter)
- UI to control
  - Visibility
  - Representation
  - Color By
  - Color maps
  - Opacity
  - Contour By (field+value)
  - Local / Remote rendering

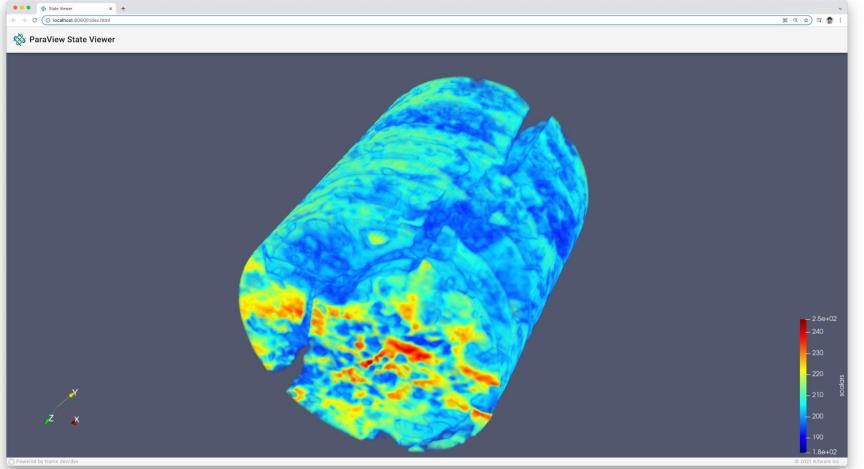
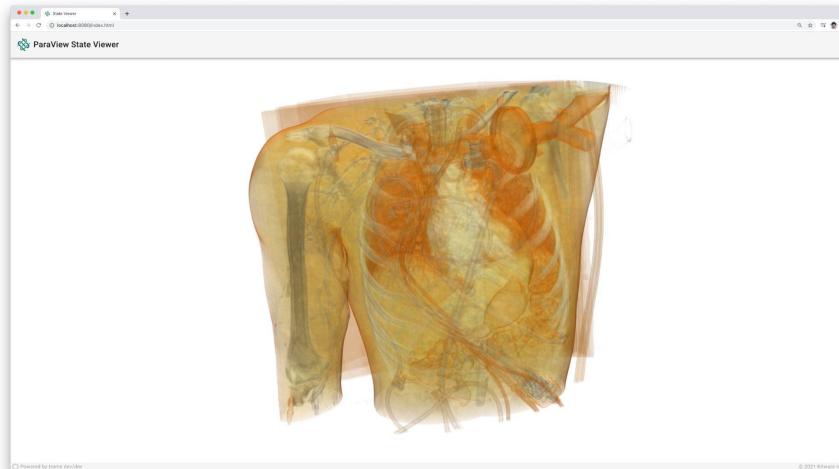
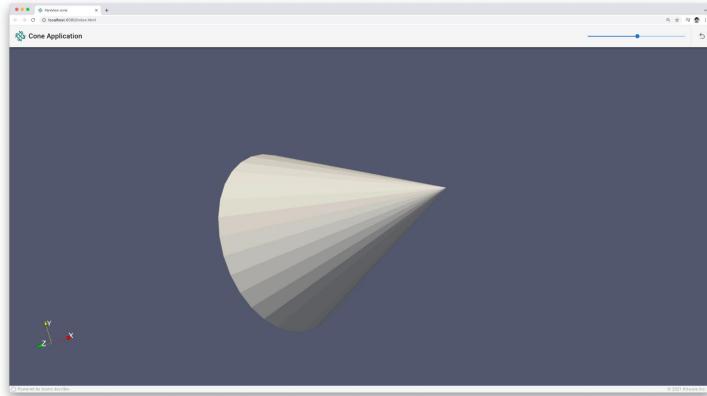


© 2021 Kitware Inc.

# Coding time...

# ParaView

- --venv path/to/venv
- Cone example
- StateLoader



# Demo time...

# Thank you!

[sebastien.jourdain@kitware.com](mailto:sebastien.jourdain@kitware.com)