

# ParaView Advanced Short



An application for display and analysis of massive scientific datasets.

# Kitware / Leader in scientific open source solutions

## Software development

Based on open source tools



## Constant Growth

Since creation of the company



## 200 employees Worldwide

USA, Europe



## 80% staff with PhD or Master

High-Level customer interaction



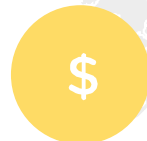
## 20 years of expertise

Kitware Inc USA, 1998  
Kitware SAS Europe, 2010



## Revenue 2019

2,1M€ Europe  
25M\$ USA



# What is ParaView?

An **application** and **architecture** for **display** and **analysis** of **massive** scientific datasets.

- **Application** - you don't have to write any **code** to **analyze** your data
- **Architecture** - designed to be **extensible** if you want to code
  - Notably custom apps (ParaViewWeb), plugins and python scripting
- **Display** - excels at traditional **sci vis** qualitative **3D rendering**
- **Analysis** - data drill down through charts, stats, all the way to values
- **Massive** - **scales** from netbooks to worlds largest supercomputers

**Strength** is its **flexibility** you create arbitrary **VTK** pipelines with it.

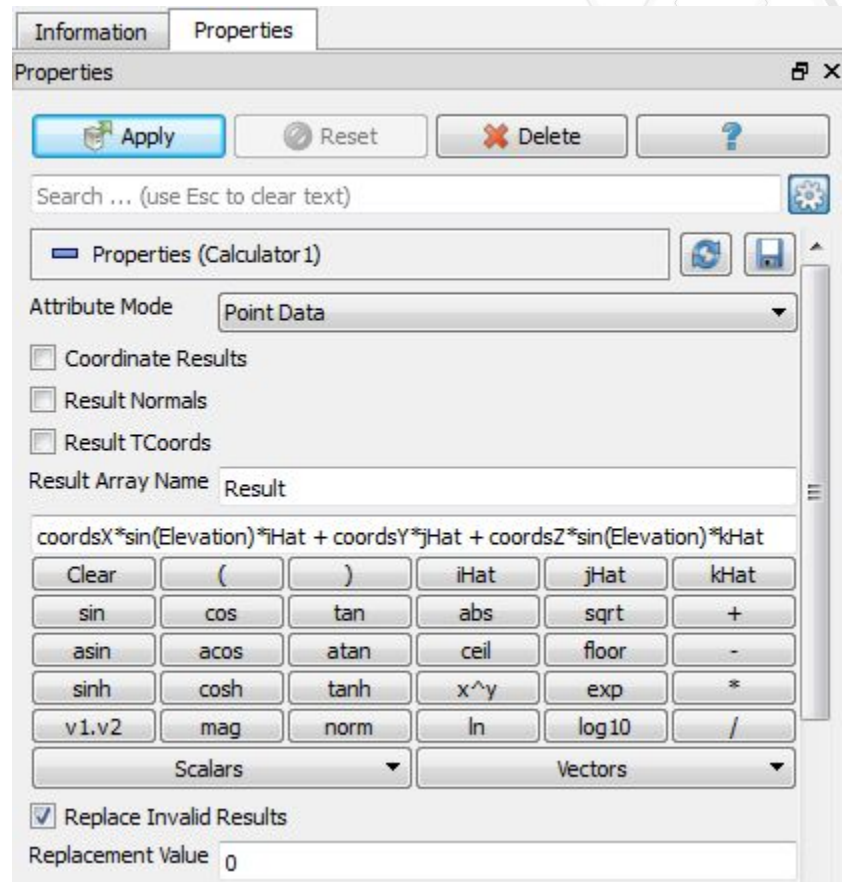
# What is ParaView?

An **application** and architecture for display and analysis of massive scientific datasets.

- **Advanced User Visualization Tool**
  - Access to advanced **filters** from VTK
  - Annotation tools
  - Advanced **Selection** tools
  - Animation panel

# Calculator

- Write expression to derive new data from input
- Expression takes in:
  - point centered scalars/vectors
  - point coordinates
  - OR cell centered scalars/vectors
- Runs over each point or cell and evaluates expression
- Expression produces either:
  - A new point centered array
  - A new cell centered array
  - New point coordinates



# Calculator – Array Manipulation Example (optional)

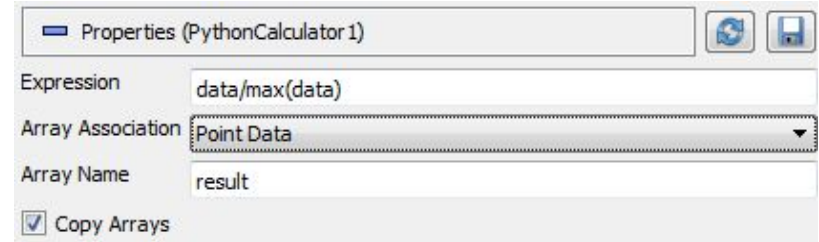
- ◆ **Open motorbike.foam**
- ◆ **Given the 3D vector,  $U$ , extract a 2D vector in the XZ plane**
- ◆ **Hint:**
  - Use  $i\hat{H}$ ,  $k\hat{H}$  unit vectors to build up output vector

# Calculator – Example

- ◆ **Attribute Mode = Point Data**
- ◆ **Coordinate Results = OFF**
- ◆ **Result Array Name = “Uxz”**
- ◆ **Expression =**
  - $U\_X * i\text{Hat} + U\_Z * k\text{Hat}$

# Python Calculator

- ◆ Add the power of Python to the Calculator
- ◆ Between the Calculator and the Programmable Filter
- ◆ Accepts multiple inputs
- ◆ Process one or more input arrays based on an expression provided by the user to produce a new output array
- ◆ Based on Python and NumPy





# Python Calculator – Accessing Input Arrays

- ◆ `inputs[0]` refers to the first input (ie. dataset) of the filter
- ◆ Accessing point data: `inputs[0].PointData['arrayname']`
- ◆ Accessing cell data: `inputs[0].CellData['arrayname']`
- ◆ Normals is equivalent to `inputs[0].PointData['Normals']`
- ◆ To access point coordinates: `inputs[0].Points[]`
  - `inputs[0].Points[:,0]` to extract the X coordinates

# Python Calculator – Basic Operations

- Supports all of the basic arithmetic operations using the `+`, `-`, `*`, `^` and `/` operators
- Always applied element-by-element to point and cell data including scalars, vectors and tensors - also work with single values
  - `Normals + 5`
    - adds 5 to all components of all Normals
  - `Normals + [1, 2, 3]`
    - adds 1 to the first component, 2 to the second component and 3 to the third component
  - `(Normals - min(Normals)) / (max(Normals) - min(Normals))`
    - normalizes the Normals array

# Python Calculator – Comparing Datasets

- ◆ `inputs[1].PointData['Iterations'] - inputs[0].PointData['Iterations']`
- ◆ **The filter always copies the mesh from the first input to its output**
- ◆ **All operations are applied point by point. It requires that the inputs have the same number of points and cells**
- ◆ **In parallel execution mode, the inputs have to be distributed exactly the same way across processes**

# Optional Exercise : Using Python Calculator

**Compute an operation between two arrays in an input dataset**

- ◆ **Open motorbike, Apply**
- ◆ **Add a Calculator and compute local Reynolds number:**
  - $Re = (U * 1) / \nu$
- ◆ **Add a Python Calculator on the first dataset and compute Re too**
- ◆ **Add a python calculator on the two previous calculator and subtract on result to another to see if there is difference in the computation.**

# Optional Exercise : Using Python Calculator : Solution

## ◆ First Calculator expression:

- $U/nut$

## ◆ First Python Calculator expression (multiple solutions):

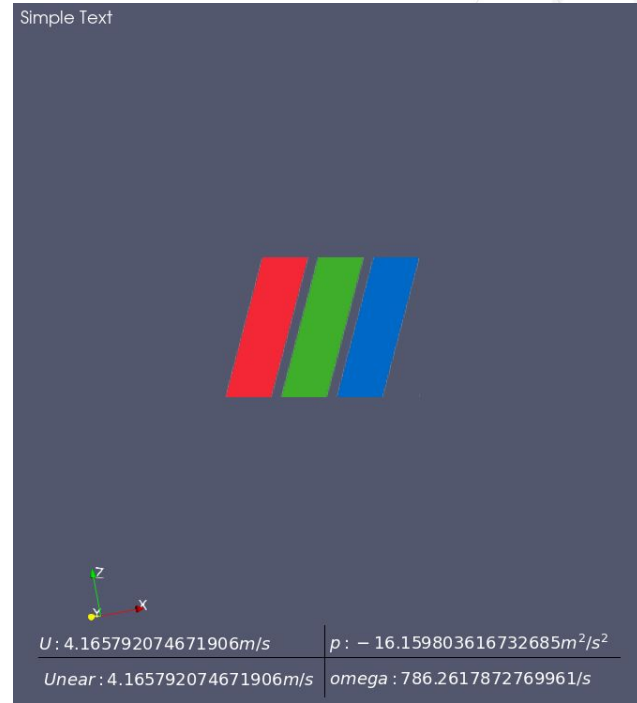
- $U/nut$
- `inputs[0].PointData["U"] / inputs[0].PointData["nut"]`
- `inputs[0].PointData.GetArray("U") / inputs[0].PointData.GetArray("nut")`

## ◆ Second Python Calculator expression (multiple solutions):

- `inputs[0].PointData["Re"]-inputs[1].PointData["Re"]`
- `inputs[0].PointData.GetArray("Re")-inputs[1].PointData.GetArray("Re")`

# Annotations filters and sources

- ◆ **Text**
  - Write your own **text** and position it
- ◆ **Annotate \* Filters**
  - Access to different **fields and values** as displayable text
- ◆ **Python Annotation**
  - Access to all inputs data to generate text using **python**
- ◆ **Text representation**
  - All above source and filters rely on this **representation**
  - Place it anywhere
  - Format it using **latex!**
- ◆ **Logo**
  - Display a **PNG logo** anywhere



# What is ParaView?

An application and **architecture** for display and analysis of massive scientific datasets.

- Open source (BSD license) and **Cross Platform**
- **Fully controllable via Python Scripting**
  - Use trace to produce a python script
  - Run it in the python shell or with pvpython/pvbatch
- **Extensible:**
  - By writing python based algorithms
  - By loading python plugins
  - By loading C++ plugins
  - By creating ParaView based applications
- Pick and choose what parts you need
- Extend and customize as you see fit
- Outside the scope of this course

# Client Side Programming

- ◆ **ParaView's scalable processing infrastructure is reusable**
- ◆ **The GUI is just one program that exercises the core features**
  - Can write others in C++
  - Much faster development time if done so in Python
- ◆ **Do in python what you did with buttons !**



# Why?

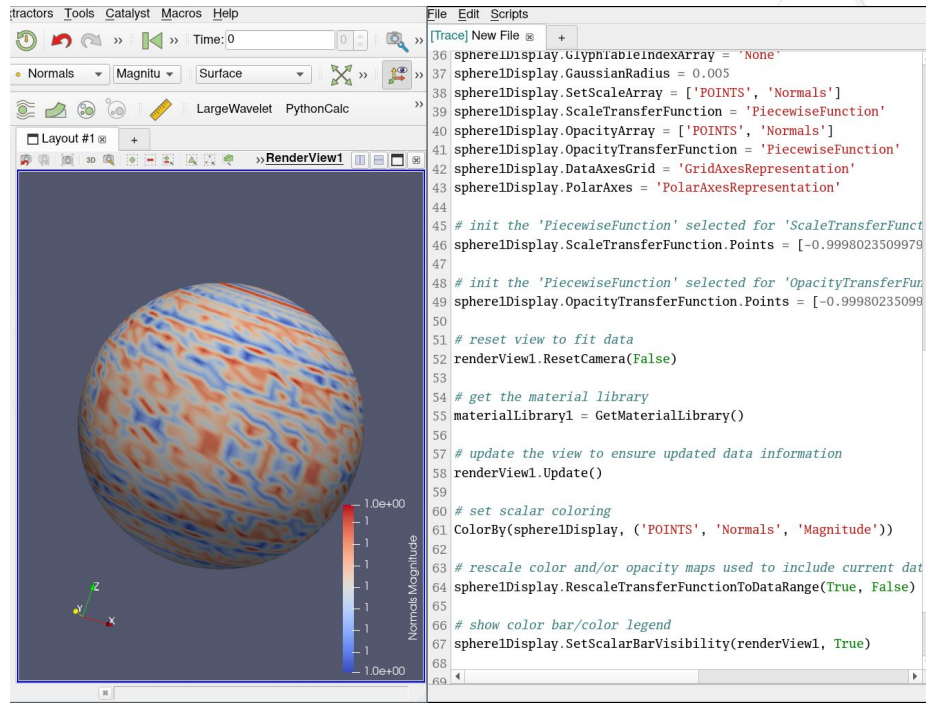
- ◆ **Run in Batch mode**
  - Set up task on small representative dataset locally
  - Repeat with real data on supercomputer
- ◆ **Script arbitrary parallel processing tasks**
  - Not just visualization
  - A parallel interpreted programming environment
  - Examine, change and act upon individual data values in huge data sets
- ◆ **To interface ParaView with other tools**
- ◆ **Scripted additions to the GUI**

# Built In Interpreters

- **Shell within GUI**
  - View > Python Shell
  - Fixed to same server that GUI is connected to
  - Trace, visual feedback, and tab completion make this easiest place to learn
- **pvpython**
  - python interpreter that comes with ParaView
  - Paths are set automatically
- **pvbatch**
  - MPI pvpython
  - Made to run on supercomputer
  - Can not interact with it, must give it filename of a script to run
  - Can not change server (no TCP) it actually runs inside the server

# Trace – Record Python Script from GUI

- **Tools->Start Trace**
  - Starts recording GUI actions
- **Tools->Stop Trace**
  - Finishes recording
  - Brings up script editor
- **File->Save**
  - Writes script to file
- **File->Save as Macro...**
  - Saves script and adds button to menu/toolbar to call it
- **Macros**
  - Execute and manage macros you have created



# Creating a Simple Visualization

- **OpenDataFile()**
  - Select the right reader and open the file
- **Show()**
  - turn visibility on
- **Render()**
  - update the view, ask for a new pipeline run if needed
- **Interact()**
  - as Render() but the view is interactive. Disable shell until interaction is done. Available in pvpython and not in ParaView python shell
  
- **Open a dataset, Show it, Render it**

```
from paraview.simple import *  
reader = OpenDataFile("path/to/file.ext")  
Show(reader)  
Render()
```



On windows, it may be needed to use a "raw string": `r"C:\path\to\b1ow1.vtk"`

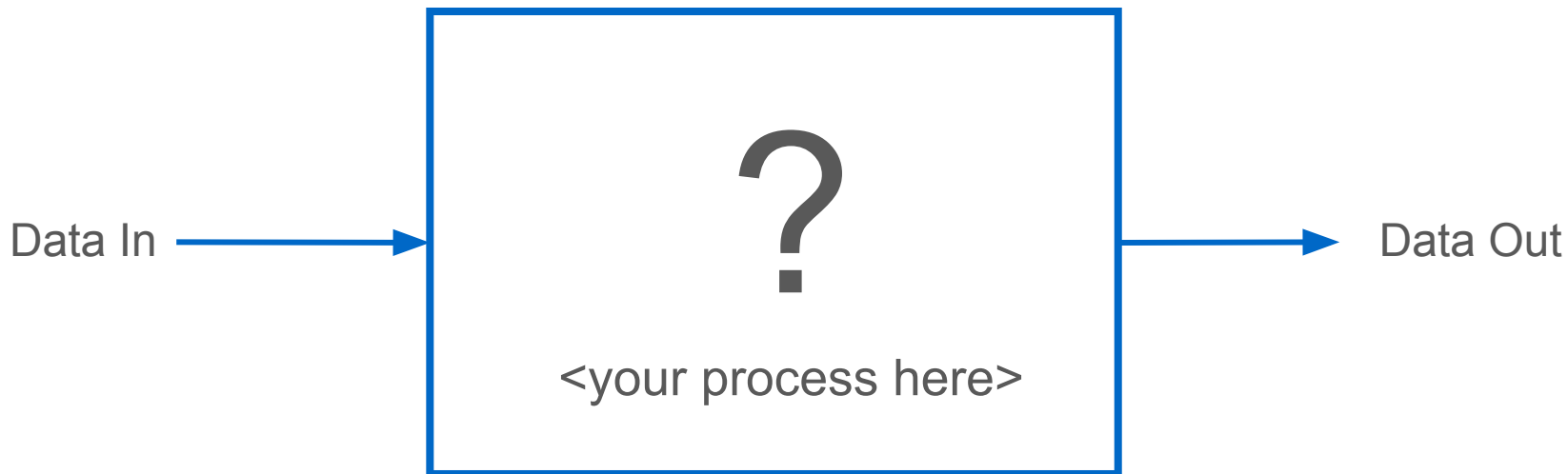
# Python Scripting

- **Python is the main scripting language for ParaView**
- **Python can be used to write pure client side code**
  - Client Side is like using UI, task is to create a pipeline
- **We are actively improving the scripting API to make it simpler and more python friendly**
- **For more information**
  - ParaView Users Doc:  
<https://docs.paraview.org/en/latest/UsersGuide/introduction.html?pvpython-scripting-interface>
  - Check out the py-doc as well :  
<https://kitware.github.io/paraview-docs/latest/python/>
  - And the pvpython Quick Start :  
<https://kitware.github.io/paraview-docs/latest/python/quick-start.html>

# Python Programmable filter

{...}

## ◆ A “White box filter”



# Python Programmable filter



- ◆ **I want a filter that does this and that ...**
  - Write one at runtime, test it as you go!
  - Same syntax as if writing pure VTK python
- ◆ **Must build using `PARAVIEW_USE_PYTHON = ON`**
  - Binary distributions are built this way
- ◆ **Available at**
  - Sources->Programmable Source
  - Filters->Programmable Filter
  - Filters->Programmable Annotation

# Effective Python Filters

- ◆ **Python filter purpose is to do arbitrary manipulation, but ParaView provides a lot of functionality**
- ◆ **If you have a task that ParaView lacks a filter for:**
  - **Set up a pipeline** to do most of the work
  - Design a filter to **bridge the feature gap**

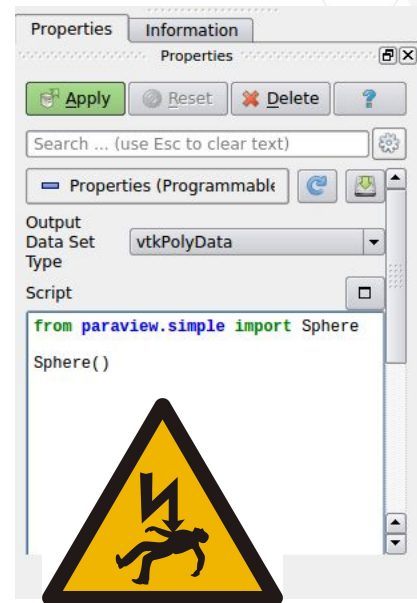


# Python Programmable filter

- ◆ **A filter – it runs on server side !**
- ◆ **Default behavior is produce copy of input geometry and topology, with attributes stripped**
- ◆ **Choose output type via menu, if not the same as input.  
Warning: Choose carefully, it can not be changed after first “Apply”**

# Python Programmable filter


- Same as using pure VTK python
  - Public C++ classes and methods in VTK
  - Usually a matter of using those to
    - Examine input data objects
    - Perform some computation
    - Fill in output data objects
  - It is like writing a VTK filter in pure Python
- **!/ \ ParaView scripting has nothing to do with it, we are in the VTK world here!!!**



# Programmable filters ecosystem

- ◆ **Programmable \* filters setup the necessary environment**
  - implicitly import `numpy` and `dataset_adapter`
  - declare `inputs` : array of all inputs
  - declare `output` : the output to populate, already initialized using `inputs[0]`
- ◆ **Those are NOT VTK object, but DataSetAdapter ones !**
  - Underlying VTK object available as ``VTKObject`` attribute.

# Exercise : Cumulative Sum Along a Line (Optional)

- How do I plot the cumulative sum of a value across space?
- Plot over line will plot values across space
- Bridge the feature gap
- Make a summation filter
  
- Steps:
  - Open the motorbike, Apply
  - Create a Plot Over Line, Apply
  - Create a Programmable Filter, Apply
  - Press the “External Edit” button: 
    - Copy the text from cumulative\_sum.py into the python editor
    - **Complete the exercise**
    - **Press apply to test your script**
  - Add a PlotData filter to visualize in a chart
  - Improve Chart

# Documentation

- **Calculator documentation:**

<https://docs.paraview.org/en/latest/UsersGuide/filteringData.html?highlight=Calculator#calculator>

- **Python Calculator documentation:**

<https://docs.paraview.org/en/latest/UsersGuide/filteringData.html?highlight=Python%20Calculator#python-calculator>

- **Python Annotation documentation:**

<https://docs.paraview.org/en/latest/ReferenceManual/annotations.html?highlight=python%20annotation#python-annotation-filter>

- **Programmable Filter/Source/Annotation documentation:**

<https://docs.paraview.org/en/latest/ReferenceManual/pythonProgrammableFilter.html>

# What about Python Plugins ?

- ◆ **Programmable filters and sources can be embedded in “Plugins”**
- ◆ **Better user interface**
- ◆ **Easy to distribute and share**
- ◆ **Editable by anyone**
- ◆ **Reload at runtime**

# What is ParaView?

An application and architecture for **display** and analysis of massive scientific datasets.

- **Advanced rendering techniques**
  - Shader based representations
  - Volume Rendering
  - Physically Based Rendering lighting
  - Raytracing

# Shader Based Representations ?

- Traditional representation in ParaView are very **similar to filters**
  - Input of the representation is the filter output that we want to show.
  - output is a simpler geometry given to the GPU for rendering.
  - Eg: *SurfaceRepresentation*.
- **More efficient** approach: Use the **GPU** to process the data
  - Great for specific algorithms : Glyphs, Volume Rendering, Isosurfaces
  - GPU memory must be able to handle the whole dataset
  - Shader based representations are implemented using GLSL



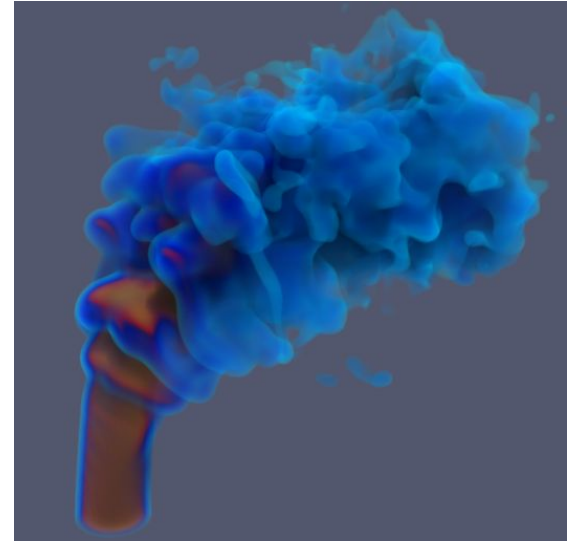
# Streamlines Representation Plugin

- Available directly in ParaView
- Load the “StreamLines Representation” plugin
- Use shader to show dynamic streamlines computed during rendering
- Instantaneous compared to the Stream Tracer filter
- **Exercise (optional) :**
  - Load StreamlinesRepresentation plugin
  - Open the motorbike surface, Apply
  - Open the motorbike volume, Apply
  - Add multiple clip to reduce the domain size
  - Create a new render view
  - Show moto in new view with opacity
  - Show clip in new view
  - Switch to streamlines representation
  - Add coloring, reduce alpha and max time to live

# StreamLines Representation in ParaView

# Volume Rendering ?

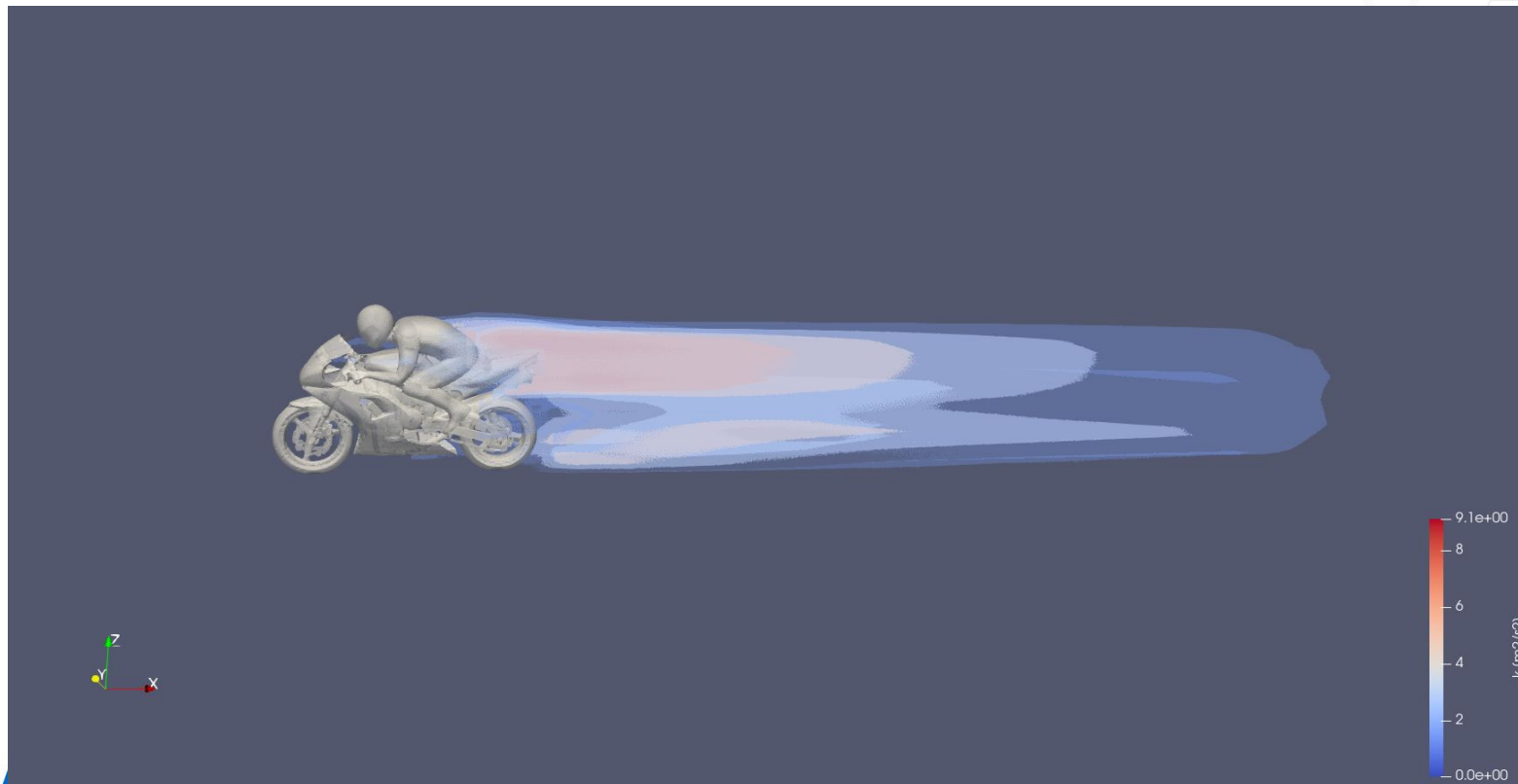
- Standard rendering method in scientific visualization postprocessing
- Complex to implement and very demanding CPU and GPU-wise
- Multiple implementations available in ParaView



# GPU Slice / Isosurface

- ◆ Isosurface available in ParaView 5.6.2
- ◆ Slice available in ParaView 5.8
- ◆ Shader based representation for **faster rendering** or isosurface and slices
- ◆ Only works with vtkImageData

# GPU rendered isosurface in ParaView



# Physically Based Rendering

- ◆ Available since ParaView **5.8.0**
- ◆ **Standard** lighting technique in the rendering industry
  - Approximated simulation of light travelling
- ◆ Attributes also mapped to the powerful **OSPRay Principled material**

# Exercise (optional)

- **Open motorbike surface, transform it in X, extract rider and the rest separately**
- Switch Lighting interpolation to **“PBR”** on both (Display / advanced mode)
- Set **Metallic** to 1 on moto, set **Roughness** to 0.5 on rider
- Set Background to **“Skybox”** and load road.hdr (View)
- Enable **“Use environnement lighting”**
- Disable default light kit (View / Light inspector)

# Simple Demo of PBR in ParaView



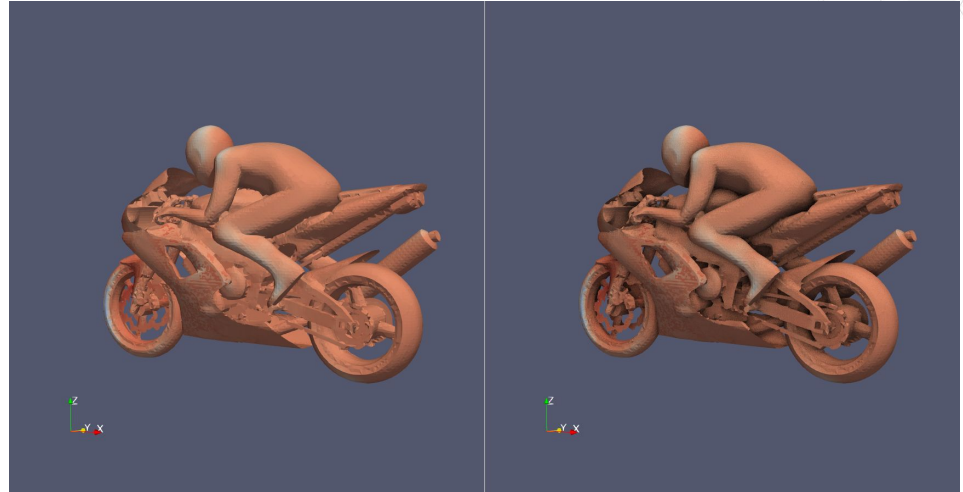


# glTF + PBR example in ParaView



# Bonus: Screen Space Ambient Occlusion (SSAO)

- ◆ Since ParaView 5.9
- ◆ Simulate short-range light occlusion
  - OpenGL backend only
- ◆ Adds great details for complex geometry with very little cost
- ◆ Enabled in *View -> Use Ambient Occlusion*
- ◆ Can be tweaked in the settings of ParaView



Without (left) and with (right) SSAO

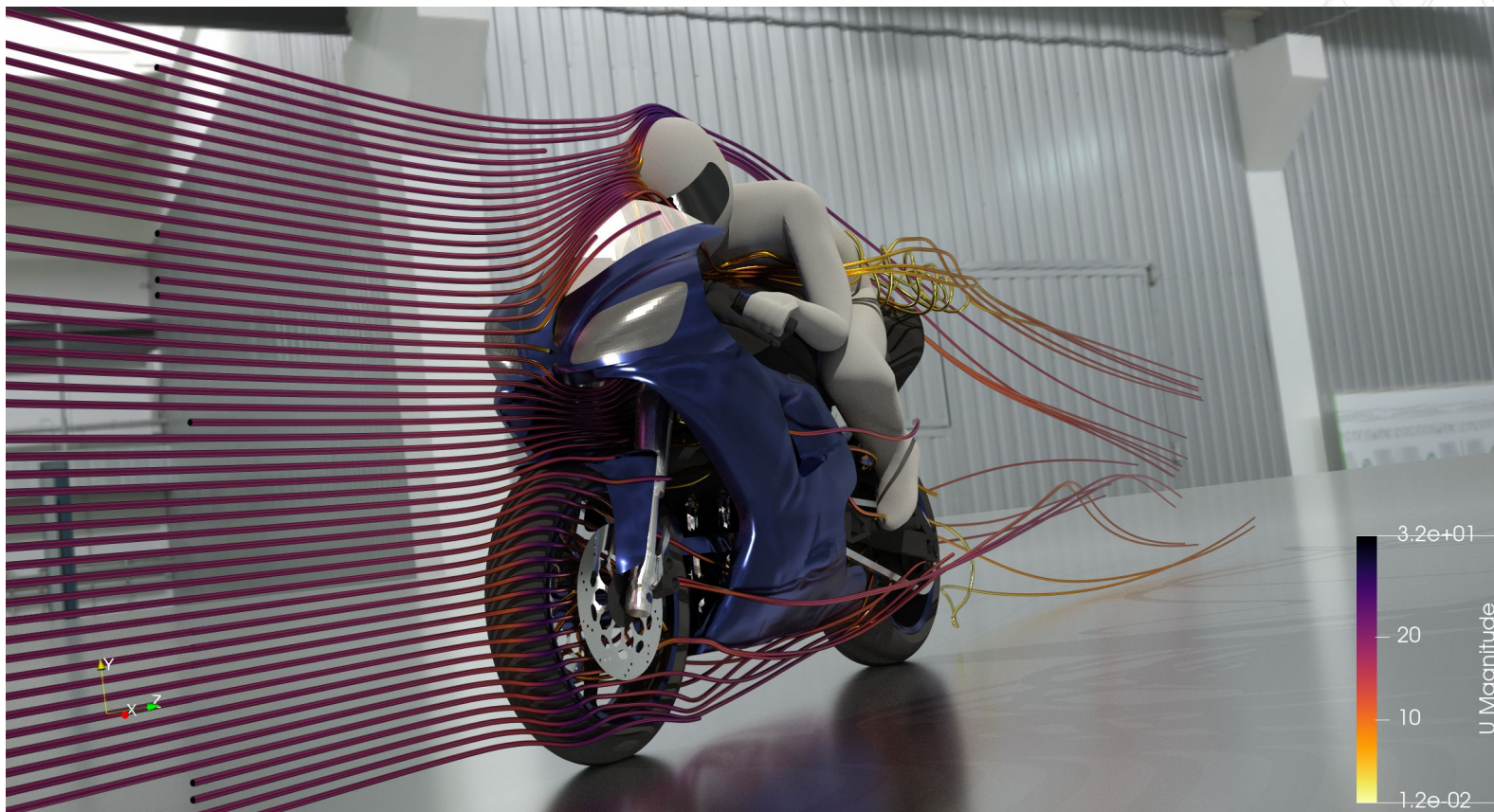
# Raytracing

- Simulate the behavior of physical light rays
- Can be very costly, especially with many rays and multiples reflections
- Two Backends in ParaView: NVIDIA Optix and Intel OSPRay

# The OpenFoam Motorbike, rendered with the NVIDIA OptiX path tracer in ParaView.







The OpenFoam Motorbike, rendered with the Intel OSPRay pathtracer in ParaView.

# What is ParaView?

An application and architecture for display and analysis of **massive** scientific datasets.

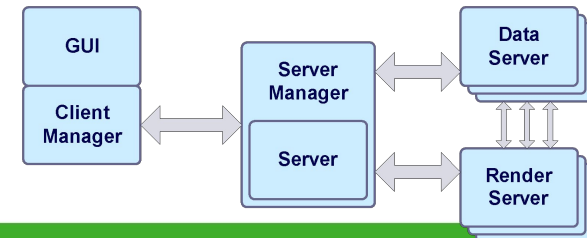
- **Client/Server architecture lets it run on a variety of platforms**
  - from netbooks
  - to the largest machines in the world
- **Support for tile display and parallel rendering**
- **Level of detail techniques keep it interactive on huge data**
- **Can perform In-Situ analysis with Catalyst**
- **Out of scope of this course**

# Distributed processing with ParaView

- ◆ **What about large data visualization?**
- ◆ **Distributed != Faster**
  - If data is **small enough** to process on **one machine**, running it on more machines probably won't make it faster
  - If data is **too large** the only way to run it is with **multiple machines**
  - With enough machines parallel interactive processing is feasible

# Visualizing Massive Data

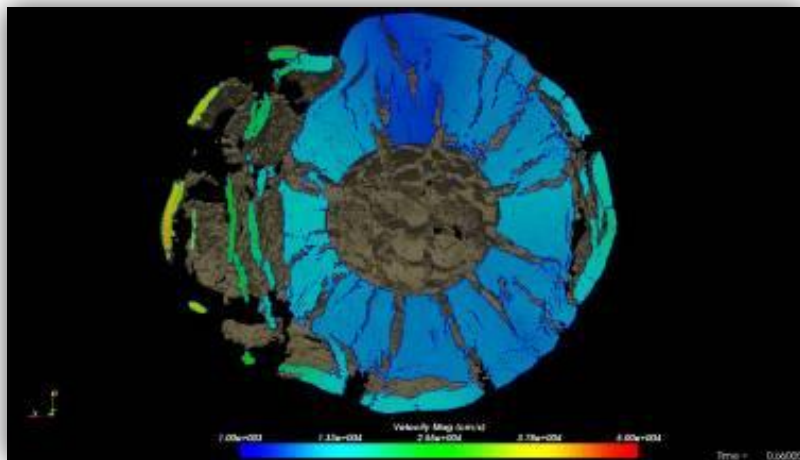
- ◆ Run data processing portion of ParaView (server) as a message passing parallel program (MPI) on a large cluster
- ◆ Distribution = Data parallelism
  - Server divides data, each of N processors gets 1/N'th (ideally)
  - Each processor runs identical processing pipeline
  - Result mesh is sent back to the client for local rendering OR image results are depth composited for local or remote display
- ◆ Run the front end (“GUI” or “Client”) as normal but connect to remote server



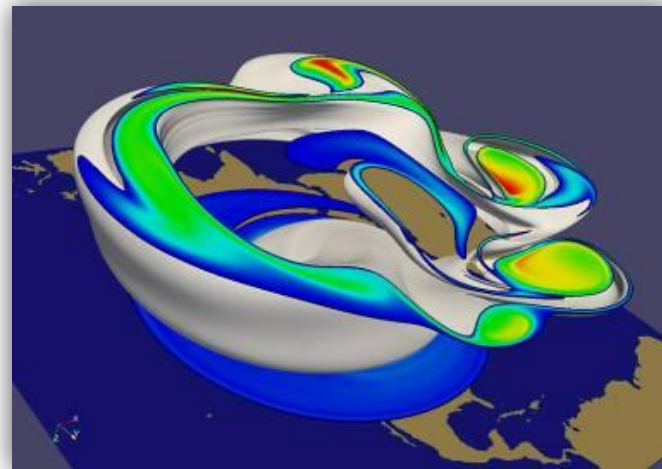


# Extremely Large Data

1 billion cell asteroid detonation simulation



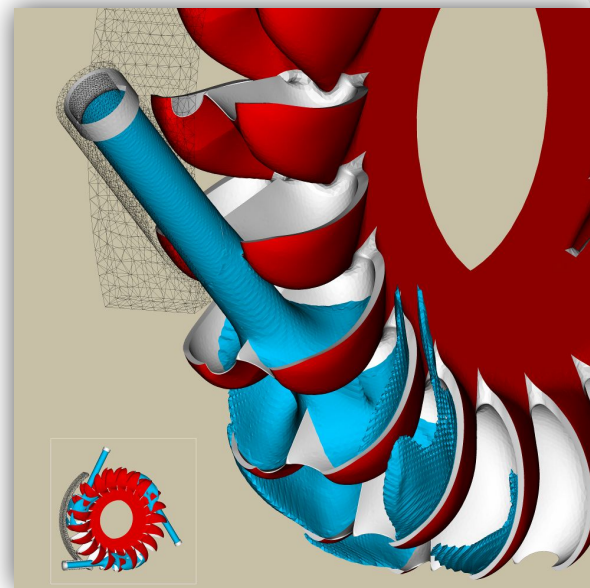
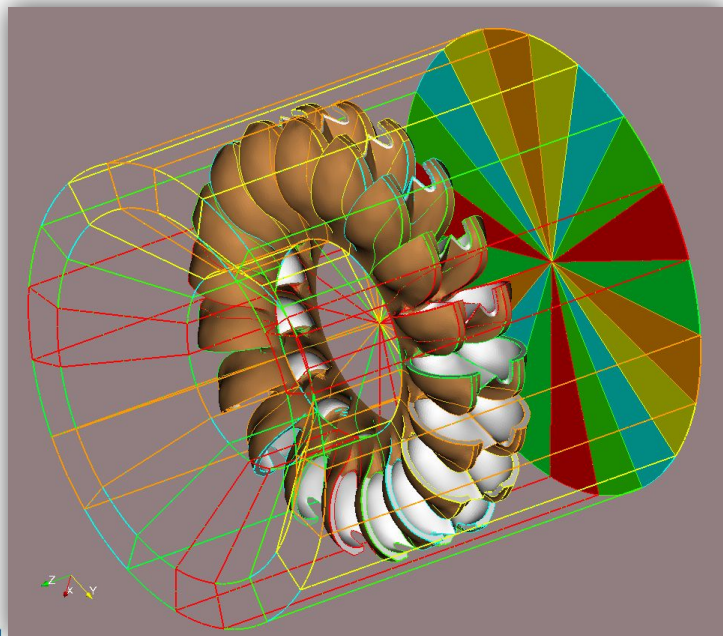
1/2 billion cell weather simulation



Source: Sandia National Lab

# Fast Large Data Interaction

CFD simulation of 20-30 million cells  
with load balancing



Source: Swiss supercomputing center

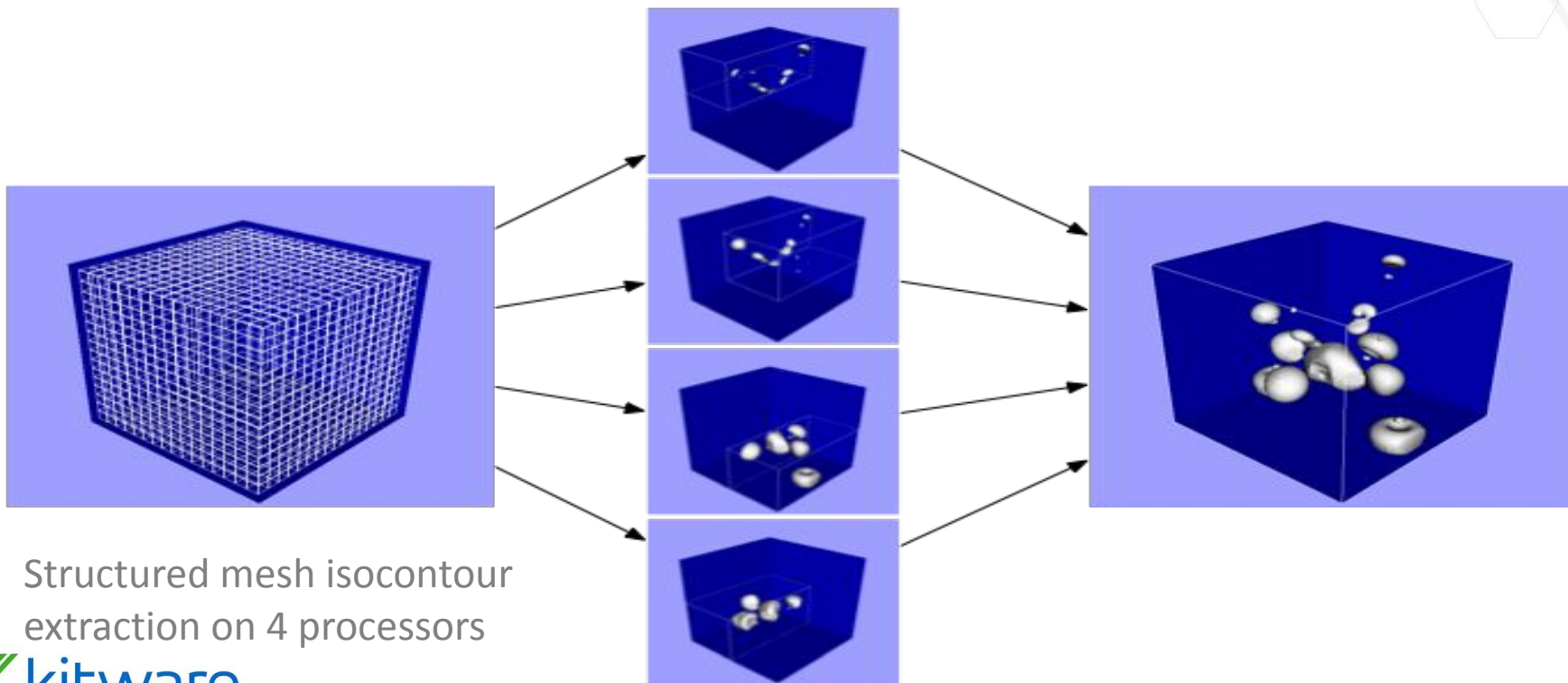
# Single Pipeline Multiple Data

- ◆ **SPMD = Single Pipeline Multiple Data**
  - **Data is split** across all processes
  - **Identical pipeline** on all processes
- ◆ **Filters can use MPI in execute methods**
  - Most filters do not

# Data (re)Distribution

- ◆ **Sources/Readers are responsible for partitioning data**
- ◆ **Partitioning is automatic for structured data, based on data extents**
- ◆ **Repartitioning and load balancing filter is available, especially for unstructured datasets**
  - D3 from Sandia (Legacy)
  - RedistributeDataSet
  - Distribute Point Cloud filter

# Distributed Processing



Structured mesh isocontour  
extraction on 4 processors

# Using MPI

Three cases:

- ◆ Use mpiexec/mpirun provided by the release
- ◆ Windows
  - make sure to install ms-mpi component
- ◆ Use your own mpi when compiling ParaView

# Running a server

## Linux / OSX

```
> cd path/to/paraview/bin  
> ./mpiexec -np 4 ./pvserver
```

## Linux / OSX Compiled

```
> cd path/to/paraview/bin  
> mpirun -np 4 ./pvserver
```

## Windows

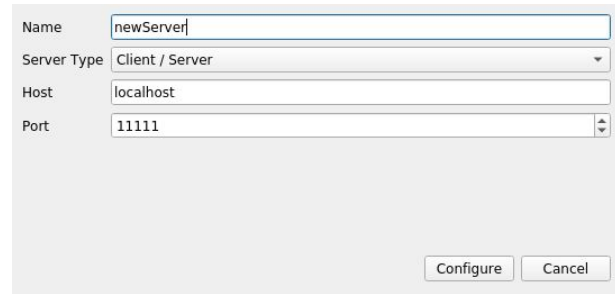
```
> cd path\to\paraview\bin  
> .\mpiexec.exe -np 4 .\pvserver.exe
```

## Windows Compiled

```
> cd path\to\paraview\bin  
> mpiexec.exe -np 4 .\pvserver.exe
```

# Configure server connection

- File/Connect/Add Server
- Name this connection to reuse it later
- Client/Server most common
- Host, Port = IP address of a machine to run pvserver on
- Startup. One of:
  - Command
    - a shell command to start pvserver on that machine
    - Ex. "ssh machine mpirun -np N pvserver"
  - Manual
    - If it is already running or you prefer to start it by hand



The screenshot shows a configuration dialog box with the following fields:

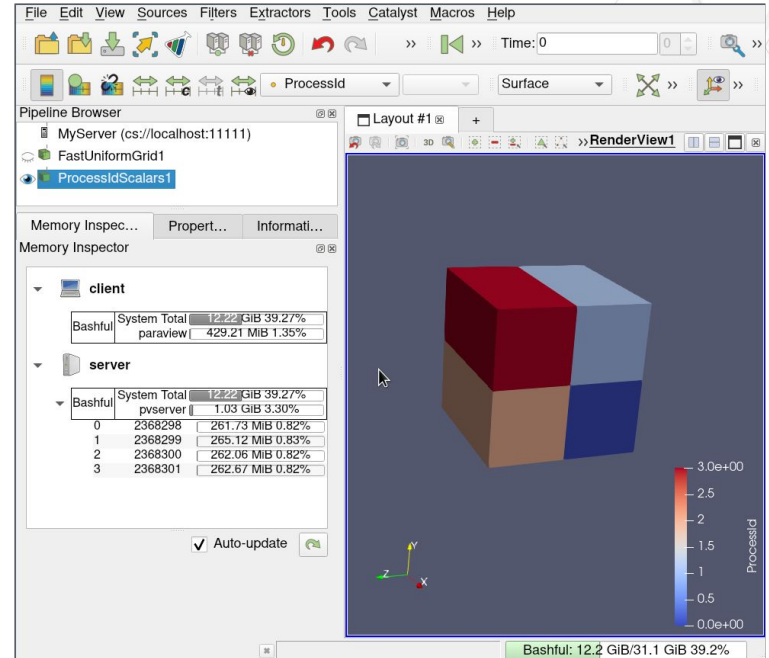
- Name: newServer
- Server Type: Client / Server (dropdown menu)
- Host: localhost
- Port: 11111 (dropdown menu)

At the bottom right, there are two buttons: "Configure" and "Cancel".



# Connecting to a Server

- File / Connect
- Choose the connection you set up above
- When connected try “Process ID Scalars” filter. It shows which processor generated/own what data



# Exercise: Simple Distributed ParaView (optional)

1. Start a four process server (in off-screen rendering mode)
2. Start a client and connect it to the server
3. Open the motorbike in “Decomposed Mode”
4. Show the ProcessId, Reset range
5. Show the memory inspector

# Going Further ...

- ParaView User Doc (Guide) – Official user's manual and reference guide
  - Accessible in the binary version of ParaView
  - Freely available as a website: <https://docs.paraview.org>
  - Printed version on Amazon
- Wiki and Forum
  - Plenty of user and developer resources
  - <https://discourse.paraview.org/>

