



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Particle data visualization and ParaView

International School of Computational Astrophysics

Jean M. Favre, CSCS

May 25-27, 2016

Outline

- Intro to ParaView
- Python numpy/VTK integration
- H5Part - Gadget interfaces
- Pause
- Python Calculator/filter
- Rendering and SPH interpolators

Objectives

Familiarize ourselves with introductory notions of:

- HDF5 file formatting, h5py syntax
- Numpy array tricks
- ParaView
 - selection and views
 - Python Calculator
 - Programmable Python sources and filters
 - General scripting and batch mode use
 - vtkSPHInterpolator

Other applications for particle-based data

- SPLASH
- IFRIT
- Yt

SPLASH: Interactive Visualization Tool for SPH Simulations

- Paper: by Daniel Price.
- D. Price was most likely the first person to state “*given that interpolation lies at the heart of SPH, consistency suggests use of the same interpolation algorithms as part of the visualization procedure*” (2007)
- Taking advantage also of the fact that fluid particles in SPH preserve their identity, some dedicated visualization procedures can be put in place, which otherwise, would not be possible with standard visualization tools.

IFRIT

- IFrIT is a powerful tool that can be used to visualize 3-dimensional data sets. IFrIT is written in C++ and is based on the state-of-the-art Visualization ToolKit (VTK) and, optionally, uses a GUI toolkit Qt.
- IFrIT has its origins (and hence name) in a specialized utility designed to visualize ionization fronts in cosmological numerical simulations. But IFrIT has outgrown its origins and now can visualize general data sets as well.
- Traditional VTK-based visualization use a pipeline paradigm for the visualization process. It enables great flexibility. IFrIT, instead, limits your flexibility somewhat by giving you only a fixed set of widget controls.
- Another important feature of IFrIT that distinguishes it from most other visualization tools is that their authors contributed dedicated development support for displaying particles.
- Yet, very recent additions to VTK (April 2016) probably still need to be integrated.

Yt

- Yt is a python package for analyzing and visualizing volumetric, multi-resolution data from astrophysical simulations, radio telescopes, and a burgeoning interdisciplinary community...



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

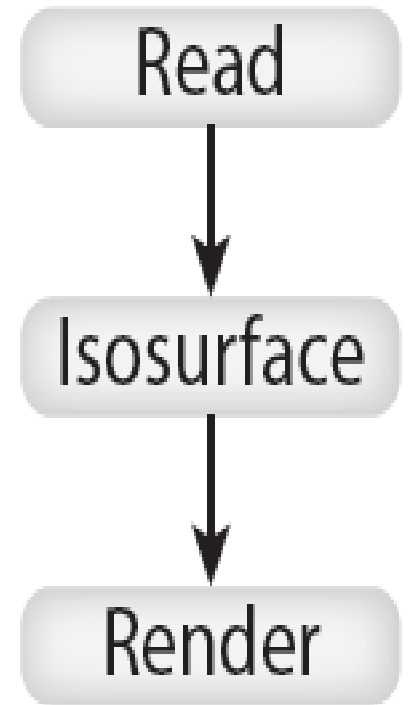
ETH zürich

The ParaView Visualization Application

ParaView: an interactive environment of Visualization modules

A visualization pipeline embodies a *dataflow network* in which computation is described as a collection of executable *modules*. There are three types of module: *sources*, *filters*, and *sinks*

ParaView, is a graphics-based application, enabling the interactive setup of Visualization pipelines. The pipelines are made of VTK objects. They are assembled via a python language interface.

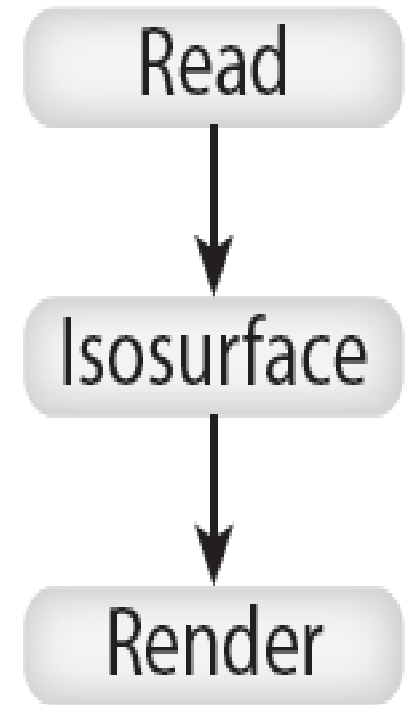


ParaView: an interactive environment of Visualization modules

ParaView can be used in multiple ways:

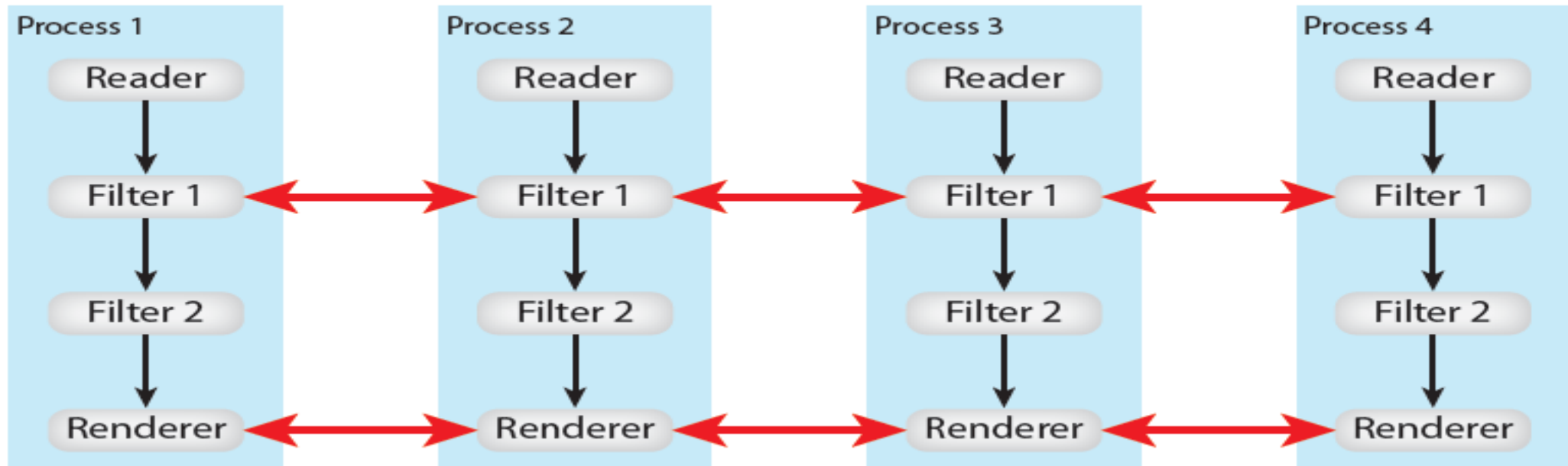
- Standalone interactive application
- Client-server (interactive)
 - Serial or parallel server
- Server only [parallel]

In all cases, it is driven by a python interface



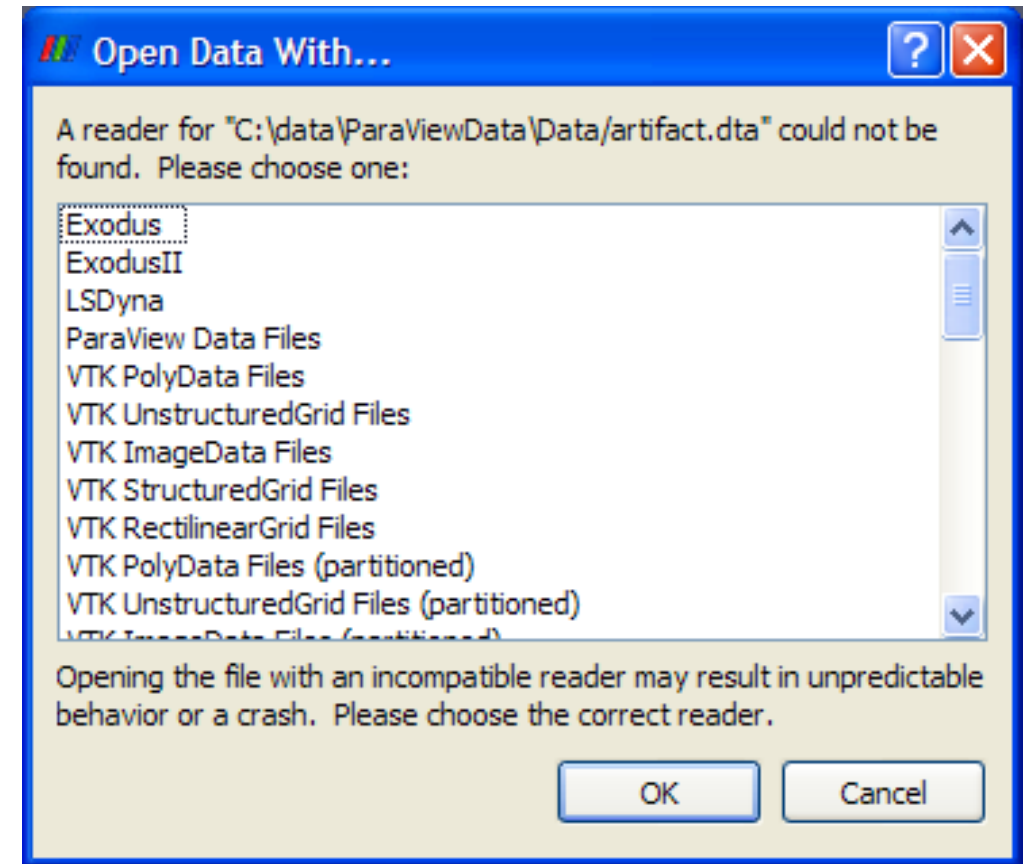
Visualization Pipeline: Data Parallelism

- Data parallelism partitions the input data into a set number of pieces, and replicates the pipeline for each piece.
- Some filters will have to exchange information (e.g. streamlines)






























ParaView/VTK data formats

- ParaView supports gridded and mesh-less data and has support for time-dependent results.
- VTK File formats
- What file formats does ParaView support?



ParaView “Data Object Generator”, and Statistics Inspector

Statistics View

Name	Data Type	No. of Cells	No. of Points	Spatial Bounds	Temporal Bounds
ImageData	 Image (Uniform Rectilinear Grid)	 1	 8	[0, 1] , [0, ...	[ALL]
UniformGrid	 Image (Uniform Rectilinear Grid) with blanking	 8	 27	[0, 1] , [0, ...	[ALL]
RectilinearGrid	 Rectilinear Grid	 1	 8	[0, 1] , [0, ...	[ALL]
StructuredGrid	 Structured (Curvilinear) Grid	 1	 8	[0, 1] , [0, ...	[ALL]
PolyData	 Polygonal Mesh	 1	 3	[0, 1] , [0, ...	[ALL]
UnstructuredGrid	 Unstructured Grid	 1	 3	[0, 1] , [0, ...	[ALL]
Octree	 AMR Dataset	 24	 81	[0, 1] , [0, ...	[ALL]
MultiBlock	 Multi-block Dataset	 2	 11	[0, 1] , [1, ...	[ALL]
TimeSource1	 Image (Uniform Rectilinear Grid)	 1	 8	[0, 1] , [0, ...	[0, 1]

Internally

- VTK data types and data formats offer all standard grid shapes and data attributes.
- Custom formats readers can be added but a thorough knowledge of internal data structures is required.
- Node-base (point-based) data is stored in a Python object called **PointData**. Likewise for **CellData**, or **FieldData**.
- A reader interface can be prototyped in python using the Python Programmable Source

HDF5

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format

We will look at two examples (conventions of usage) to familiarize ourselves with basic concepts of HDF5

- H5Part
- GADGET

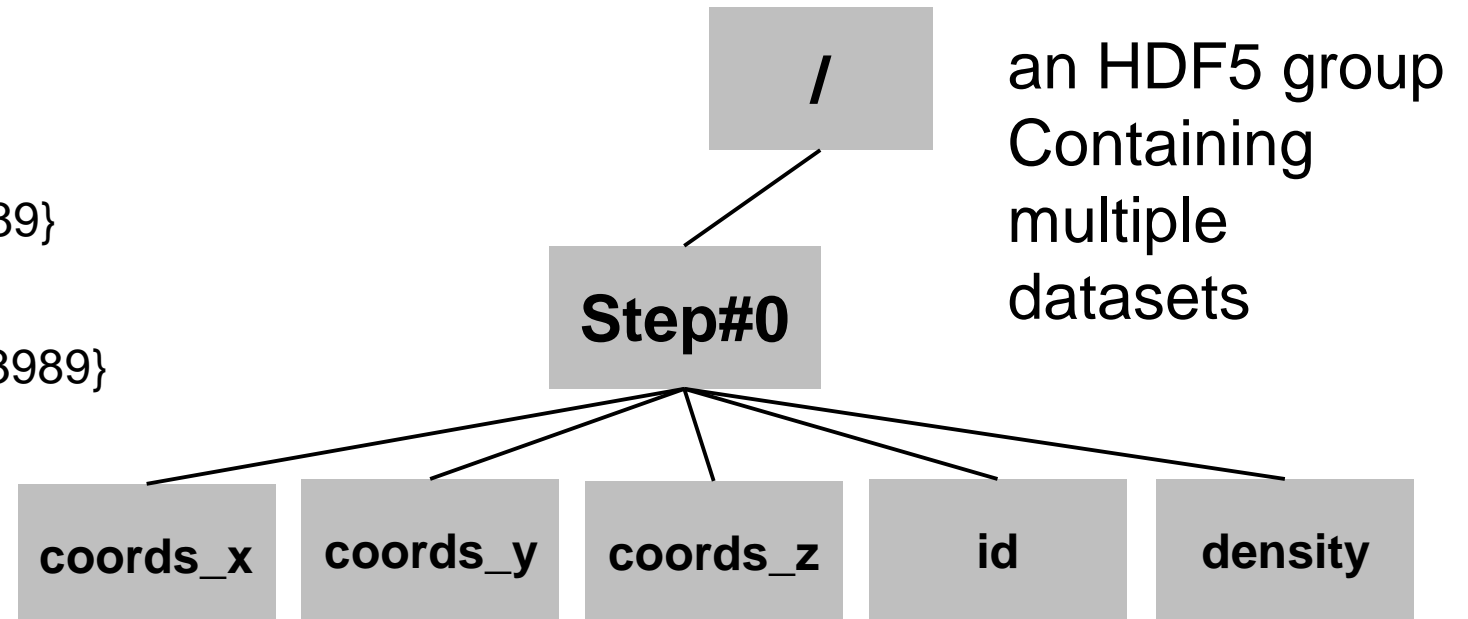
H5Part: an HDF5-based SPH file format

```
h5ls -r particles00509.h5part
```

/	Group
/Step#0	Group
/Step#0/x	Dataset {23989}
/Step#0/y	Dataset {23989}
/Step#0/z	Dataset {23989}
/Step#0/density	Dataset {23989}
/Step#0/electron\ fraction	Dataset {23989}
/Step#0/gravitational\ potential	Dataset {23989}
/Step#0/id	Dataset {23989}
/Step#0/specific\ internal\ energy	Dataset {23989}
/Step#0/vel_x	Dataset {23989}
/Step#0/vel_y	Dataset {23989}
/Step#0/vel_z	Dataset {23989}

Some tools:

h5ls
hdfview
h5dump



H5Part file format

- H5Part is a usage convention to store particles datasets in the HDF5 format
- Using h5py, we have a straightforward interface to create simple files.

A minimal set of python source code to create an H5Part file

given, x, y, z, id as numpy arrays of shape (N,)

```
import h5py
file = h5py.File("particles.h5part","w")
g = file.create_group("Step#0")
d1 = g.create_dataset("x",data=x[:], dtype='f')
d2 = g.create_dataset("y",data=y[:], dtype='f')
d3 = g.create_dataset("z",data=z[:], dtype='f')
d4 = g.create_dataset("id",data=id[:], dtype='f')
file.close()
```

Reading back the data

```
import h5py
```

```
file = h5py.File("particles00509.h5part")
```

```
grp = file['Step#0']
```

```
gp = grp["gravitational potential"][:]
```

```
ef = grp["electron fraction"][:]
```

<http://docs.h5py.org/en/latest/>

“file” is a python dictionary

file.keys() return ['Step#0']

grp.keys() returns

['x', 'y', 'z', 'density', 'electron fraction',
'gravitational potential', 'id', 'specific
internal energy', 'vel_x', 'vel_y', 'vel_z']

GADGET: an HDF5-based SPH file format

The image displays four overlapping windows of the HDFView 2.9 application, illustrating the hierarchical structure of the `snap_009.14.hdf5` file. The windows show different levels of the file's hierarchy, from the root to specific data groups.

Window 1 (Leftmost): Shows the root of the file. The left pane lists the hierarchy: `Constants`, `Header`, `Parameters`, `ChemicalElements`, `NumericalParameters`, `RadiativeTransferParameters`, `StellarEvolutionParameters`, and `WindParameters`. The right pane displays the `Header` group details:

```
Header (800, 2)
Group size = 0
Number of attributes = 13
BoxSize = 4.4375
ExpansionFactor = 0.0
Flag_Cooling = 0
Flag_Feedback = 0
Flag_Metals = 0
Flag_Sfr = 0
Flag_StellarAge = 0
HubbleParam = 0.7
MassTable = 0.0, 9.4
NumFilesPerSnapshot = 1
NumPart_InFile = 56
NumPart_Total = 56
NumPart_Total_HighRes = 56
NumberOfTasks = 1
Omega_m = 0.365
```

Window 2: Shows the `PartType0` group. The left pane lists the hierarchy: `PartType0`, `PartType1`, `PartType4`, `Coordinates`, `Density`, `ElementAbundances`, `InitialMass`, `Mass`, `Metallicity`, `ParticleIDs`, `SmoothedElementAbundances`, `SmoothingLength`, `StellarFormationRate`, and `Velocity`. The right pane displays the `Units` group details:

```
Units (2536, 2)
Group size = 0
Number of attributes = 13
UnitDensity_in_cgs = 1.0
UnitEnergy_in_cgs = 1.0
UnitLength_in_cm = 1.0
UnitMass_in_g = 1.0
UnitPressure_in_cgs = 1.0
UnitTime_in_s = 3.0
UnitVelocity_in_cm = 1.0
```

Window 3: Shows the `ChemicalElements` group. The left pane lists the hierarchy: `ChemicalElements`, `NumericalParameters`, `RadiativeTransferParameters`, `StellarEvolutionParameters`, and `WindParameters`. The right pane displays the `ChemicalElements` group details:

```
ChemicalElements (12760, 2)
Group size = 0
Number of attributes = 13
BG_ELEMENTS = 9
CalciumOverSilicon = 0.0941736
ElementNames = Hydrogen, Helium, Carbon, Nitrogen, Oxygen, Neon, Magnesium, Silicon, Iron
InitAbundance_Carbon = 0.0
InitAbundance_Helium = 0.25
InitAbundance_Hydrogen = 0.75
InitAbundance_Iron = 0.0
InitAbundance_Magnesium = 0.0
InitAbundance_Neon = 0.0
InitAbundance_Nitrogen = 0.0
InitAbundance_Oxygen = 0.0
InitAbundance_Silicon = 0.0
SulphurOverSilicon = 0.605416
```

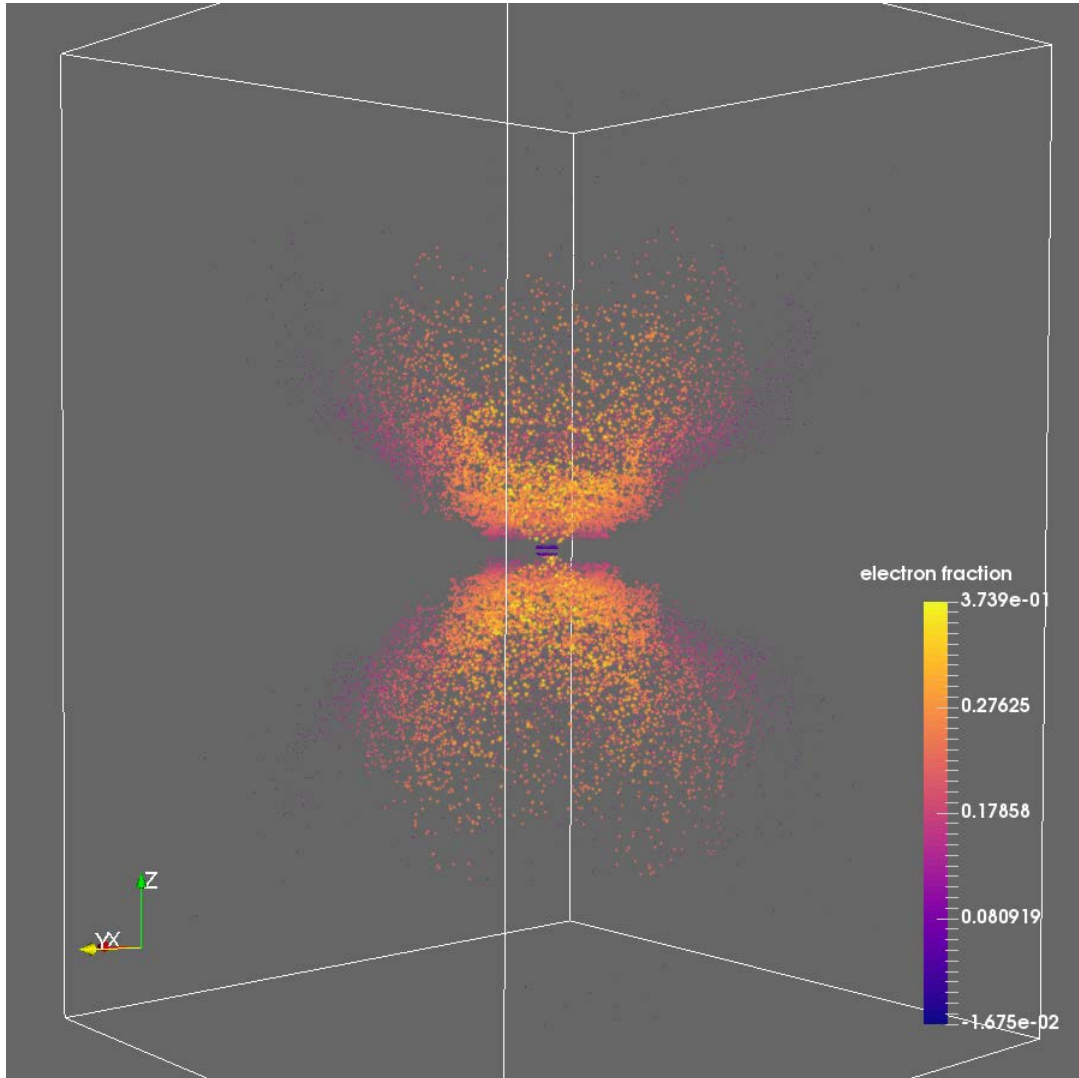
Window 4 (Rightmost): Shows the `Units` group. The left pane lists the hierarchy: `Units`. The right pane displays the `Units` group details:

```
Units (2536, 2)
Group size = 0
Number of attributes = 13
UnitDensity_in_cgs = 1.0
UnitEnergy_in_cgs = 1.0
UnitLength_in_cm = 1.0
UnitMass_in_g = 1.0
UnitPressure_in_cgs = 1.0
UnitTime_in_s = 3.0
UnitVelocity_in_cm = 1.0
```

How to read H5Part or Gadget data in ParaView

- H5Part is a C++ compiled plugin
- A prototype Gadget interface is available, written in Python. I will be very glad to share my source code with you. Send me email: jfavre@cscs.ch
- The Python Gadget interface can be used as an example of other I/O interfaces written in Python with h5py.

H5Part Input dataset used in the next slides



- 23989 particles with Id, 3 components of velocity, position, «electron fraction», ...
- Can be viewed as a cloud of Gaussian Points, with a ParaView-specific plugin for GPU-rendering

3D points and beyond

We'll start with simple examples of usage of matplotlib

Then, we'll go to ParaView and present

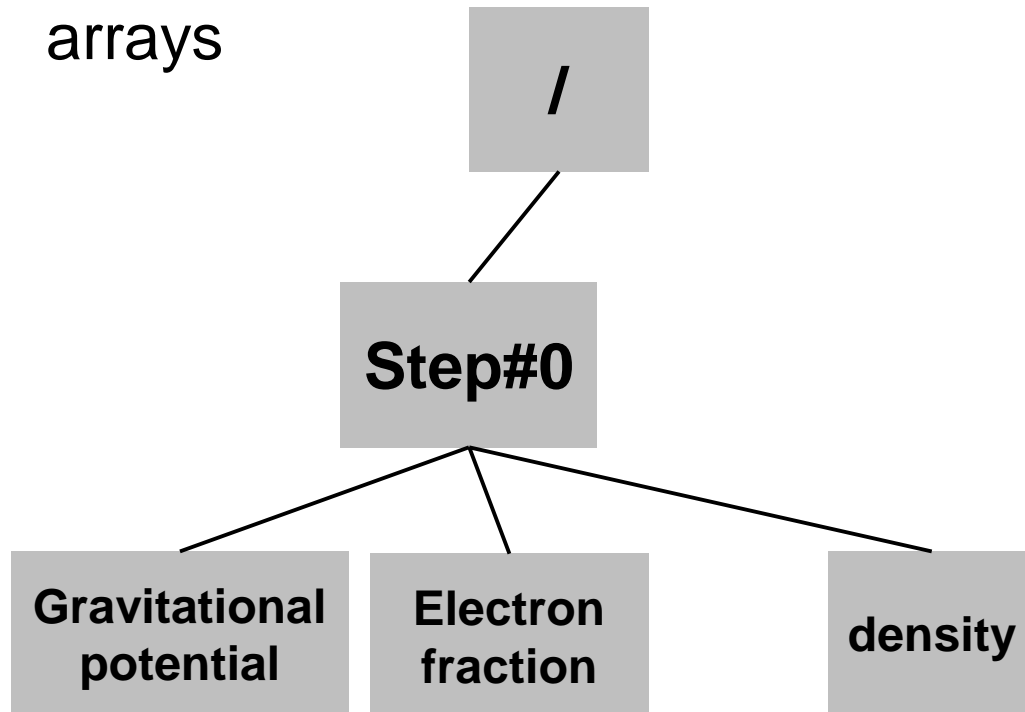
- Visual selections/filtering
- Numerical selections/filtering

Matplotlib introductory tutorial at scipy-lectures.github.io

- Matplotlib is the de-facto standard for 2D plotting in the Python world

Matplotlib minimal example

- With a very simple python console, one can replicate the standard scatter plot of SPLASH or IFRIT
- Read the data with h5py into numpy arrays



```
import matplotlib.pyplot as plt
import h5py
```

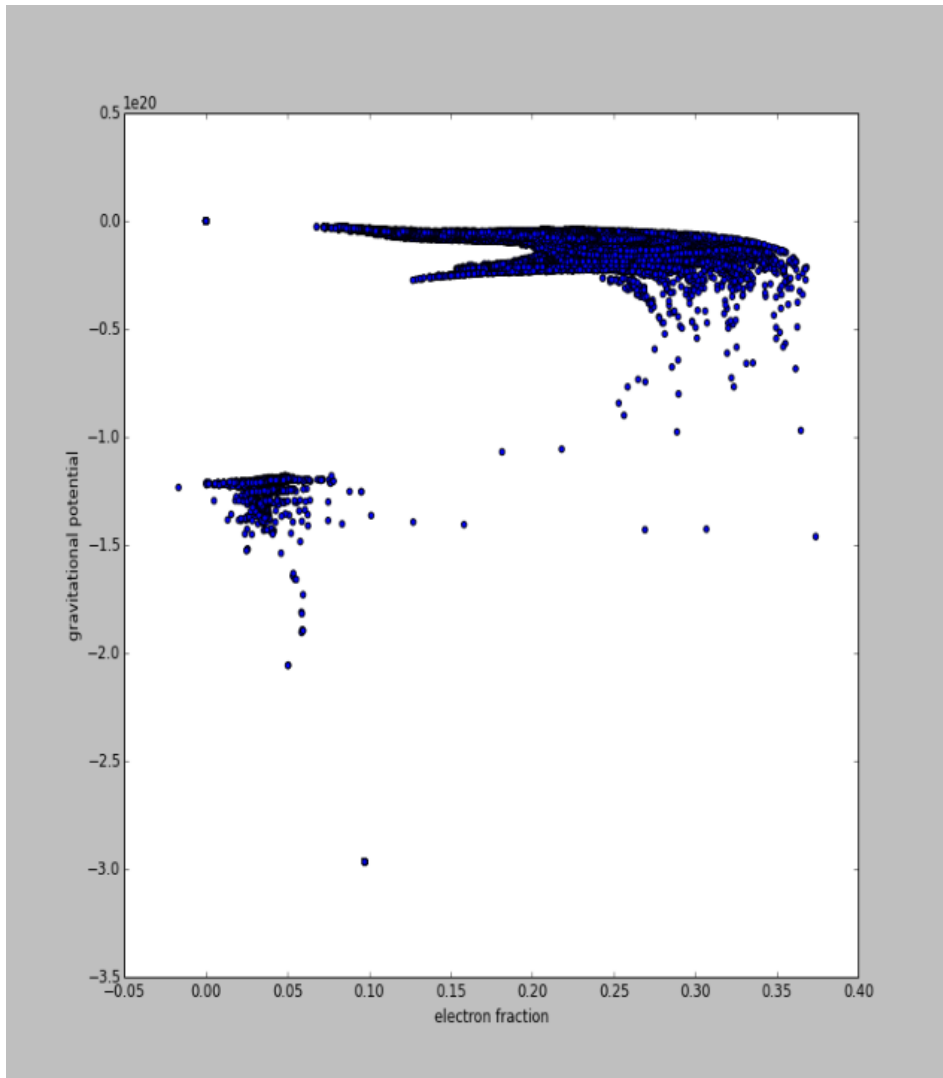
```
file = h5py.File("particles00509.h5part")
```

```
gp = file['Step#0']["gravitational  
potential"][:]
```

```
ef = file['Step#0']["electron fraction"][:]
```

```
plt.subplots_adjust(hspace=0.5)
```


Matplotlib: scatter plot



```
plt.subplot(122)
```

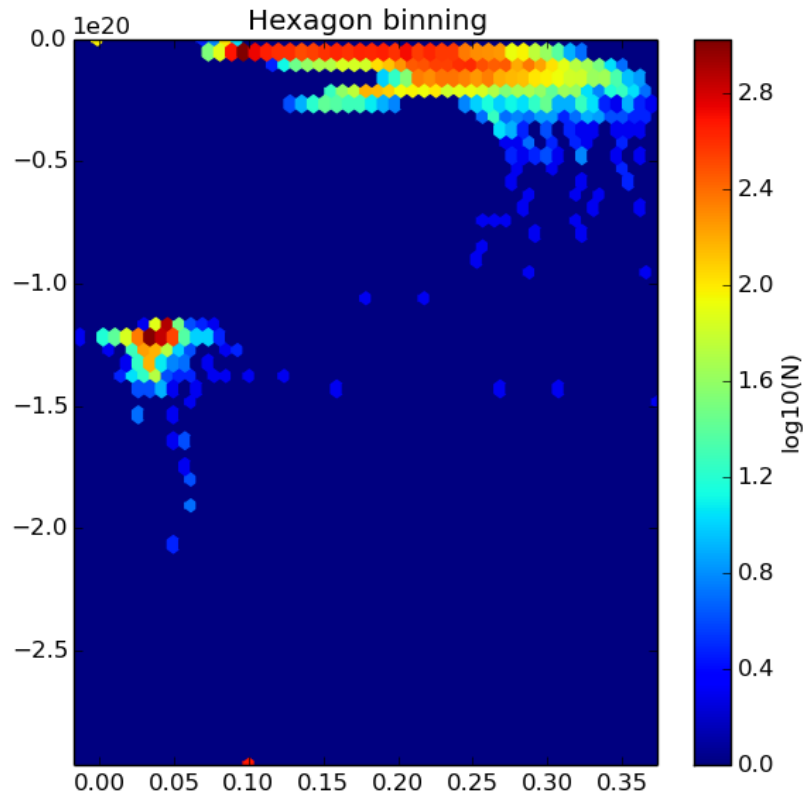
```
plt.set_xlabel("electron fraction")
```

```
plt.set_ylabel("gravitational potential")
```

```
plt.scatter(ef, gp)
```

```
plt.show()
```

Matplotlib: hexagonal binning



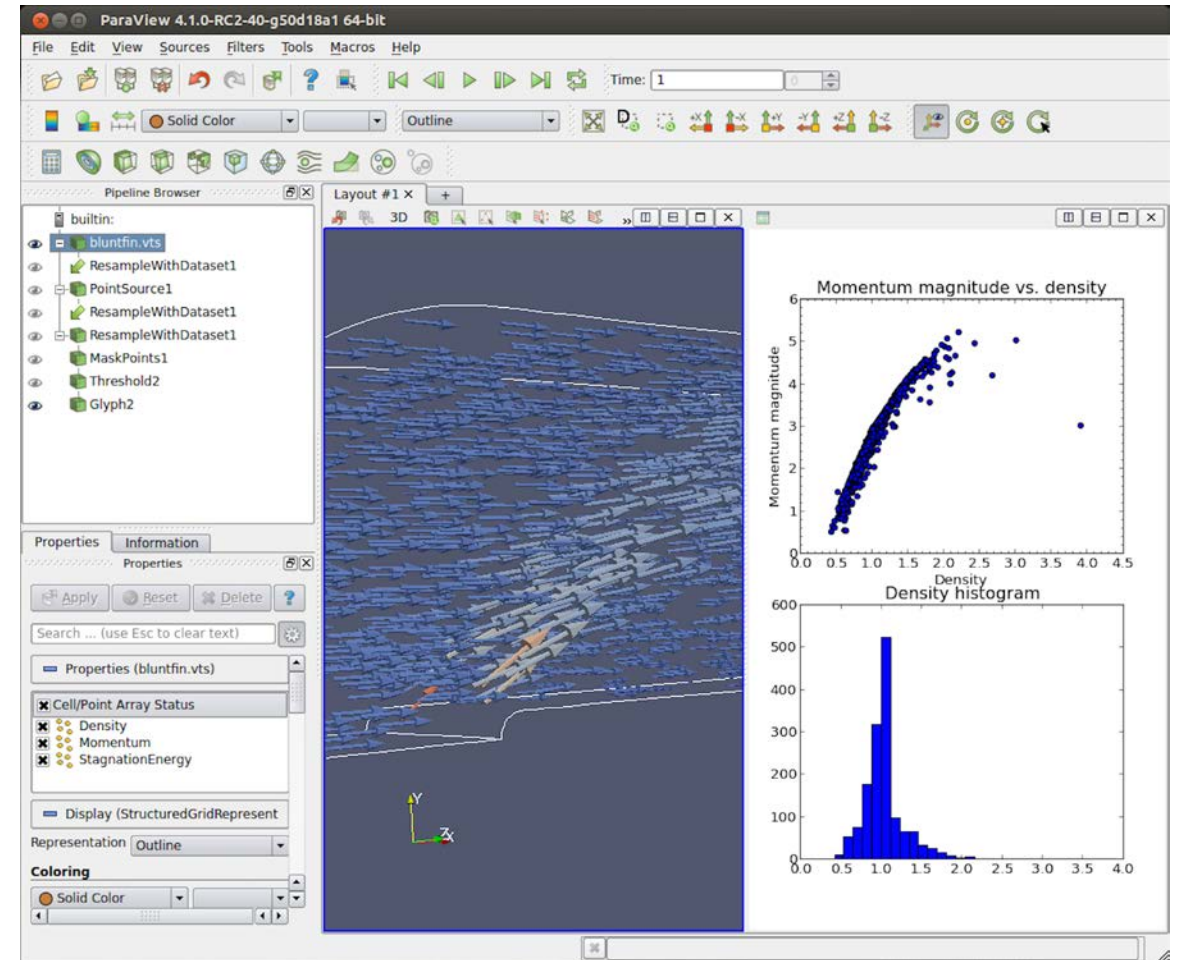
hexbin is an axes method or pyplot function that is essentially a pcolor of a 2-D histogram with hexagonal cells. It can be much more informative than a scatter plot

```
plt.hexbin(ef, gp, gridsize=50, bins='log')
```

```
plt.show()
```

ParaView Python View

- Matplotlib commands can actually be used directly within a dedicated View Window of ParaView.
- Mix 3D graphics and 2D plotting.
- See page 73 of the User manual



ParaView's Python View

def setup_data(view):

```
numVisibleObjects =
view.GetNumberOfVisibleDataObjects()

for i in xrange(numVisibleObjects):

    dataObject =
view.GetVisibleDataObjectForSetup(i)

    if dataObject:

        pd = dataObject.GetPointData()

        desiredArrays = ["electron fraction",
"gravitational potential"]

        for arrayName in desiredArrays:

            view.SetAttributeArrayStatus(i,
vtkDataObject.POINT, arrayName, 1)
```

def render(view, width, height):

```
figure = Figure()

ax1 = figure.add_subplot(1,1,1)

numVisibleObjects =
view.GetNumberOfVisibleDataObjects()

for i in xrange(numVisibleObjects):

    dataObject =
view.GetVisibleDataObjectForRendering(i)

    if dataObject:

        pd = dataObject.GetPointData()

        ef = pd.GetArray("electron fraction")

        gp = pd.GetArray("gravitational potential")

        np ef = numpy_support.vtk_to_numpy(ef)

        np gp = numpy_support.vtk_to_numpy(gp)

        ax1.scatter(np ef, np gp)
```

Selections / filtering

- Visual (point and click)
- Numerical
 - We will need introductory notions of numpy



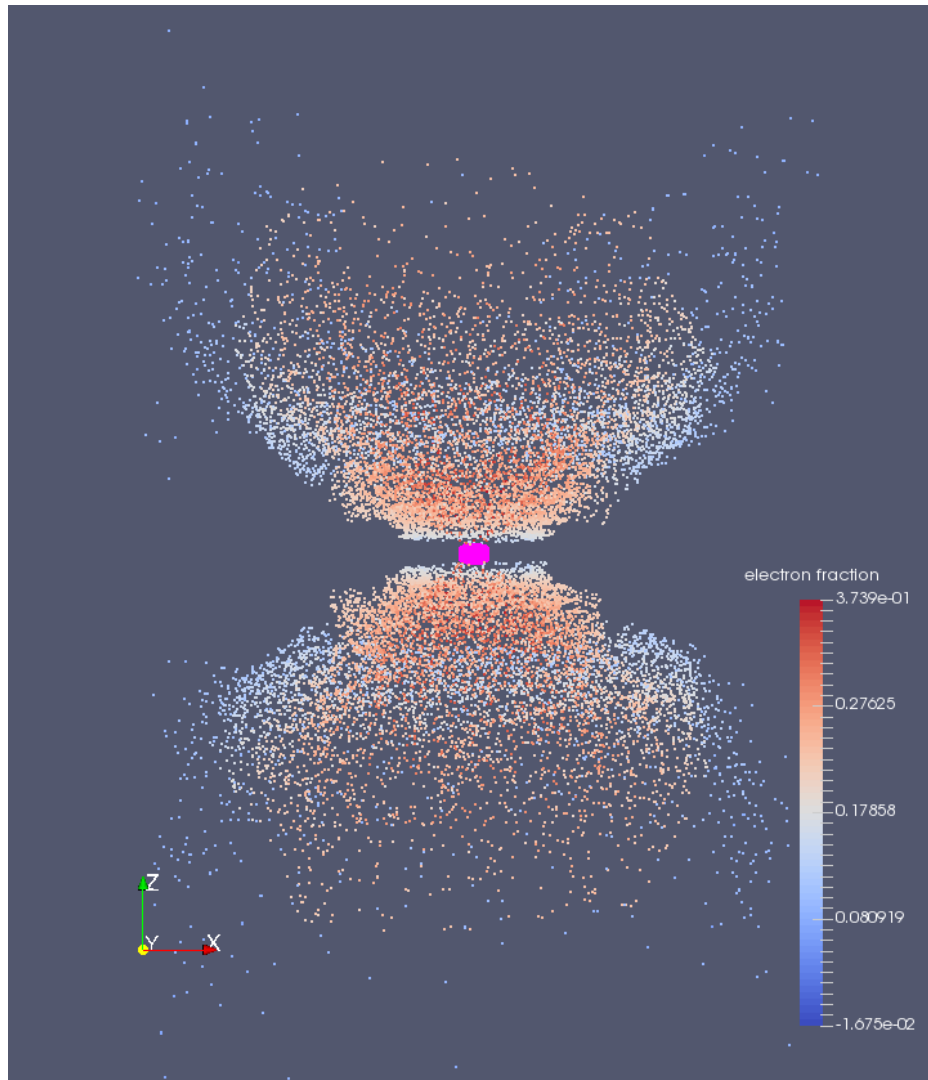
CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

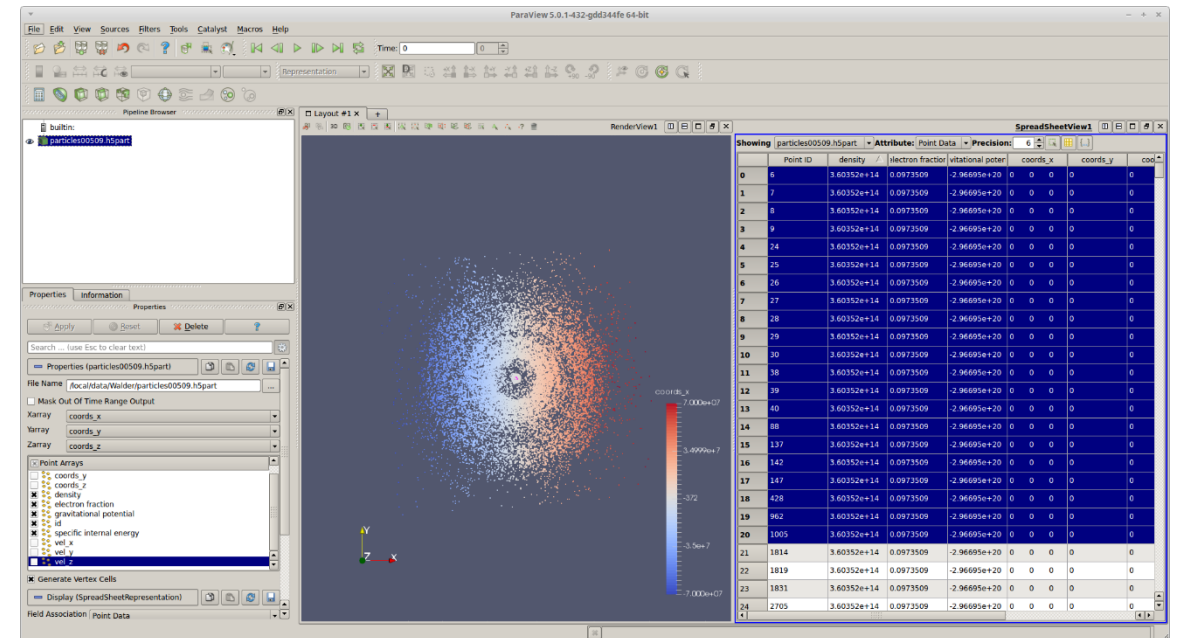
Particle selection via interactive viewing and mouse selection

Spreadsheet View

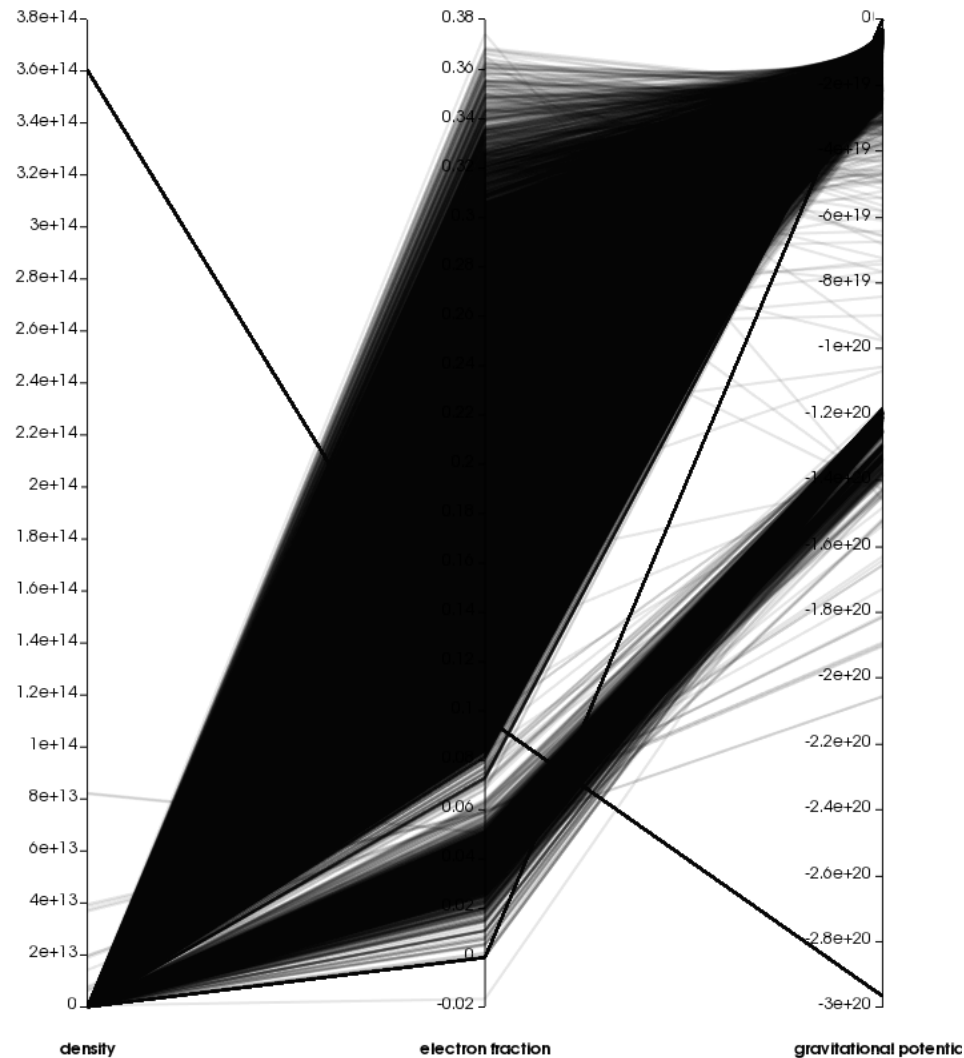


The Spreadsheet view enables sorting, and selection.

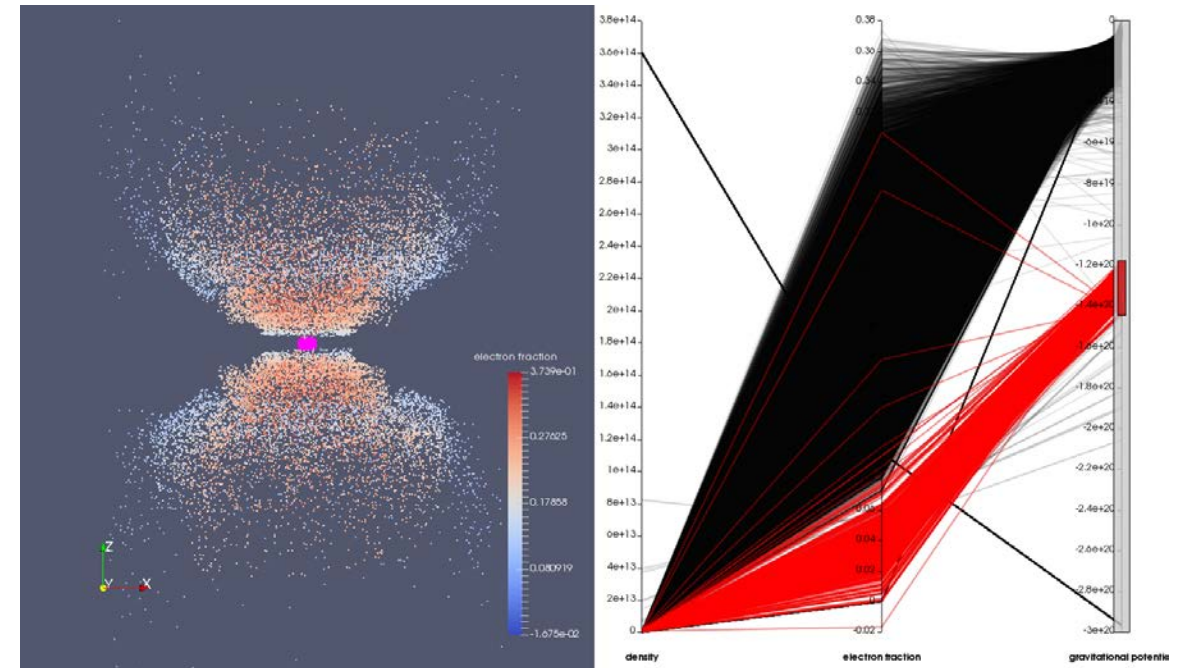
All point-and-click selections in ParaView are synchronized in all views



Parallel Coordinates View

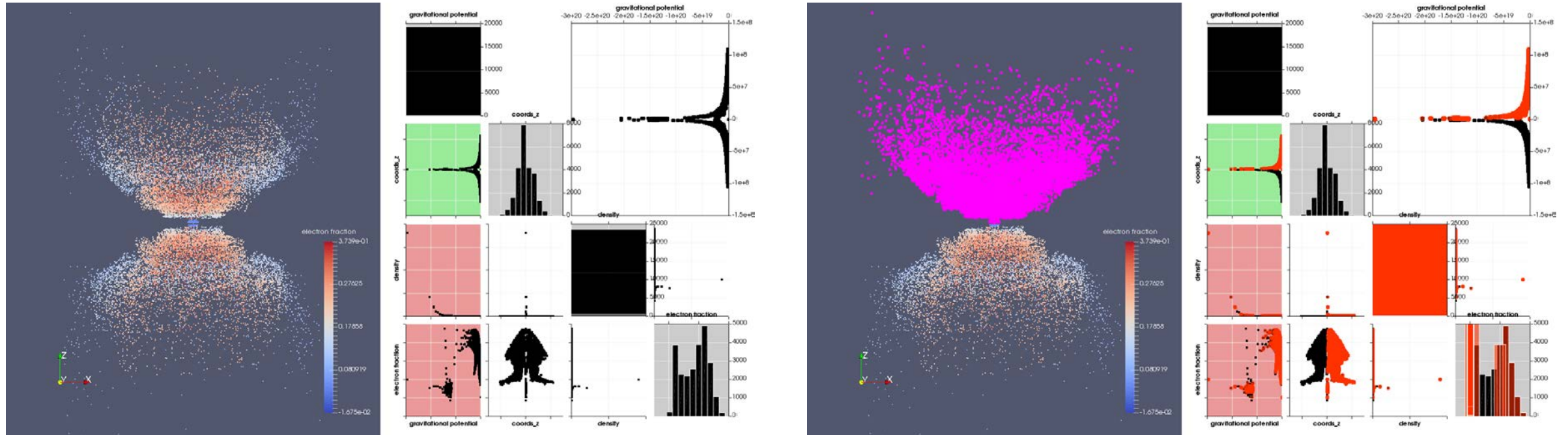


Parallel coordinates offer a column-wise display of the frequency distribution of data, and enables a multi-mask selection



Plot Matrix View

A diagonal matrix of plots showing correlation between all individual scalar fields





CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Particle selection via numpy array handling

Numpy data arrays

Numpy provides an N-dimensional array type, the *ndarray*, which describes a collection of “items” of the same type. The items can be *indexed* using for example 3 integers (i.e. `a[start:end:step]`)

<http://scipy-lectures.github.io/intro/numpy/index.html>

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> a[2:9:3]
array([2, 5, 8])
```

Numpy data arrays

```
>>> momentum = np.array([[0,1,2], [3,4,5], [6,7,8], [9,10,11]])
```

```
>>> momentum.shape
```

```
(4, 3)
```

```
>>> momentum[:, 0]           # reads the 0-th component
```

```
array([0, 3, 6, 9])
```

```
>>> momentum[:, 1]           # reads the 1-st component
```

```
array([ 1,  4,  7, 10])
```

```
>>> momentum[:, 2]           # reads the 2-nd component
```

```
array([ 2,  5,  8, 11])
```

Numpy array selection

```
xyz = np.array([
    [ 0.,  0.,  0.],
    [ 1.,  0.,  0.],
    [ 2.,  0.,  0.],
    [ 0.,  1.,  0.],
    [ 1.,  1.,  0.],
    [ 2.,  1.,  0.],
    [ 0.,  0.,  2.],
    [ 1.,  0.,  2.],
    [ 2.,  0.,  2.],
    [ 0.,  2.,  2.]
])
```

```
coord_z = xyz[:,2]
```

```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  2.,  2.,  2.,  2.])
```

```
coord_z < 2
```

```
array([ True,  True,  True,  True,  True,  True,
       False, False, False, False], dtype=bool)
```

Numpy array selection

```
xyz[coord_z < 2]
```

```
array([[ 0.,  0.,  0.],  
       [ 1.,  0.,  0.],  
       [ 2.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 1.,  1.,  0.],  
       [ 2.,  1.,  0.]])
```

```
np.where(coord_z < 2)  
array([0, 1, 2, 3, 4, 5])
```

```
xyz[ coord_z >= 2 ]
```

```
array([[ 0.,  0.,  2.],  
       [ 1.,  0.,  2.],  
       [ 2.,  0.,  2.],  
       [ 0.,  2.,  2.]])
```

```
np.where(coord_z >= 2)
```

```
array([6, 7, 8, 9])
```

Numpy array selection

```
import h5py
import numpy as np
file = h5py.File("particles00509.h5part", "r")

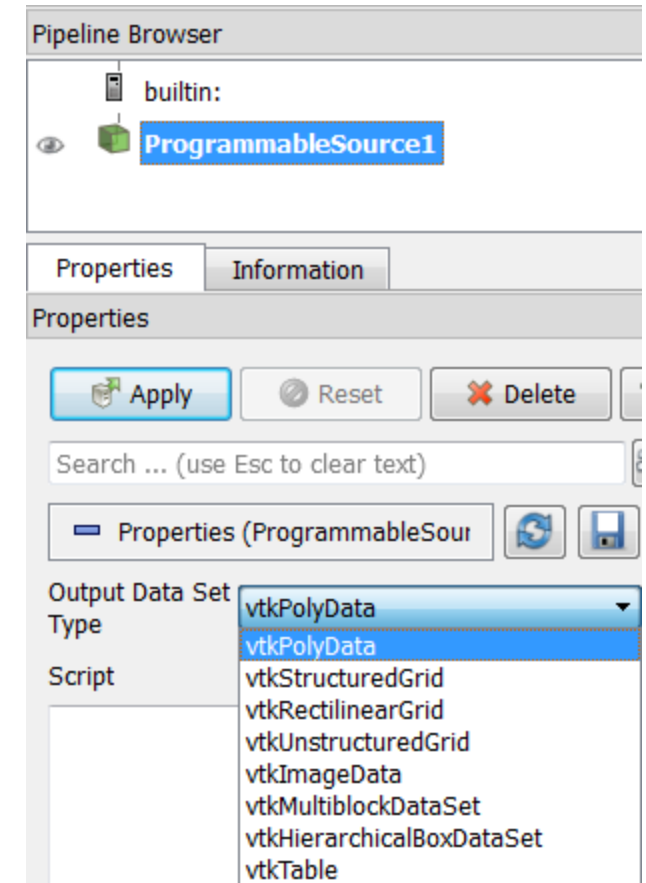
ef = file['Step#0']['electron fraction'][:]
coord_z = file['Step#0']['z'][:]
mask1 = ef < .1
mask2 = coord_z < 0
filtering = np.logical_and(mask1, mask2)
# invert the selection by np.invert(filtering)
indices = np.where(filtering)
```

Create Boolean masks, such as

- “ $ef < .1$ ” are all particles where “electron fraction” is $< .1$
- “ $coord_z < 0$ ” are all particles whose Z coordinate is negative
- Apply as many Boolean combinations of masks using AND. OR,..

Numpy array selection is done in a Python Programmable Source/Filter

1. Define an output dataset type,
2. Write a python initialization script [optional]
3. Write the main python script



VTK Data Objects and numpy

VTK Data objects are scientific datasets such rectilinear grids or finite elements. These datasets are formed of smaller building blocks: mesh (topology and geometry) and attributes.

In general, a mesh consists of vertices (points) and cells (elements, zones). Cells are used to discretize a region and can have various types such a vertex, lines, tetrahedra, hexahedra etc.

Warning! VTK has its own internal storage conventions.

ParaView's Programmable Filter enables the transparent exchange of numpy arrays to VTK

- Python Calculator is simple
 - Pages 100-104 of the manual
- Python ProgrammableSource and Filters are more involved
 - Pages 159-169 of the manual

ParaView Programmable Filter and Python Calculator basics

Can have multiple inputs

“input” and “output” are predefined names

Each of them is accessed with an index, as in

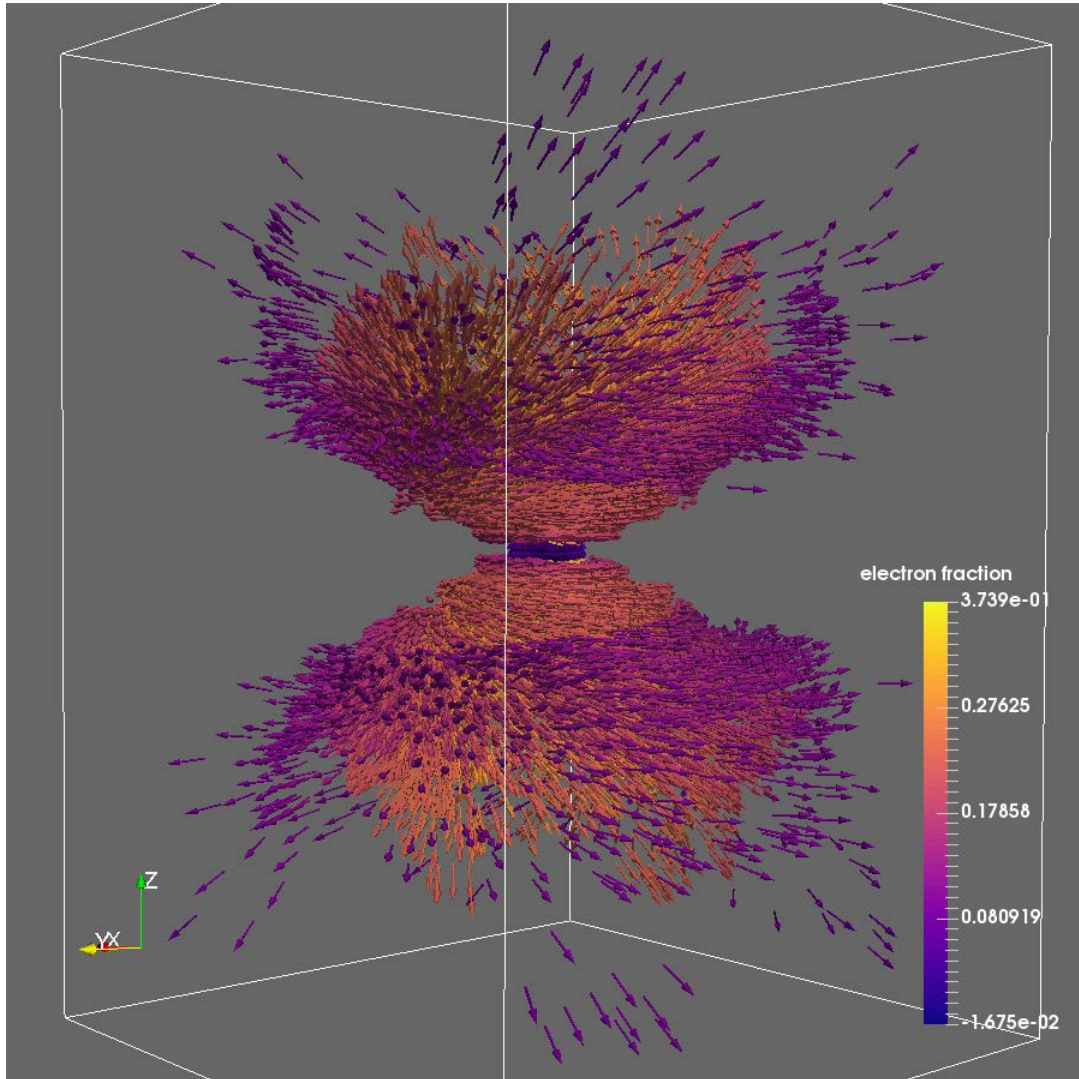
`i=0, inputs[i]`

`inputs[i].Points` # gives access to a numpy array of coordinates (x,y,z)

`inputs[i].PointData` # gives access to a dictionary of numpy arrays of data attributes stored at the nodes

See also `inputs[i].CellData, inputs[i].FieldData`

The velocity vector is assembled with the Python Calculator



Syntax: `make_vector(vx, vy, vz)`

- `vx = inputs[0].PointData['x-velocity']`
- `vy = inputs[0].PointData['y-velocity']`
- `vz = inputs[0].PointData['z-velocity']`
- `make_vector(vx, vy, vz)`

Displayed, using Glyphs

Calculate a simple derived field

Implementation:

The calculator automatically “copies” (via reference counting), the mesh and the existing data arrays.

We simply need to append the new derived field array via a numpy expression

Example:

```
(inputs[0].FieldData["gamma"] - 1.0) * \
inputs[0].PointData["density"] * inputs[0].PointData["thermal_energy"]
```

ParaView Python Programmable Filter

- Used to calculate non trivial derived fields
- Used to do selection
- Runs, in parallel, on the data server side of ParaView

Calculate a more involved derived field

eg: Calculate radial velocity

```
input = inputs[0]
```

```
bv = input.FieldData["bulk_velocity"]
```

```
if bv is dsa.NoneArray:
```

```
    bv = np.zeros(3)
```

```
xv = input.PointData["Velocities"][:,0] - bv[0]
```

```
yv = input.PointData["Velocities"][:,1] - bv[1]
```

```
zv = input.PointData["Velocities"][:,2] - bv[2]
```

```
xyz = input.Points
```

```
bbox = np.array( [
```

```
    np.min(xyz[:,0]), np.max(xyz[:,0]),
```

```
    np.min(xyz[:,1]), np.max(xyz[:,1]),
```

```
    np.min(xyz[:,2]), np.max(xyz[:,2]) ] )
```

```
center = np.array([
```

```
    (bbox[1]-bbox[0])*0.5,
```

```
    (bbox[3]-bbox[2])*0.5,
```

```
    (bbox[5]-bbox[4])*0.5 ])
```

Calculate radial velocity

```
x_hat = xyz[:,0] - center[0]
```

```
y_hat = xyz[:,1] - center[1]
```

```
z_hat = xyz[:,2] - center[2]
```

```
r = np.sqrt(x_hat*x_hat+y_hat*y_hat+z_hat*z_hat)
```

```
x_hat /= r
```

```
y_hat /= r
```

```
z_hat /= r
```

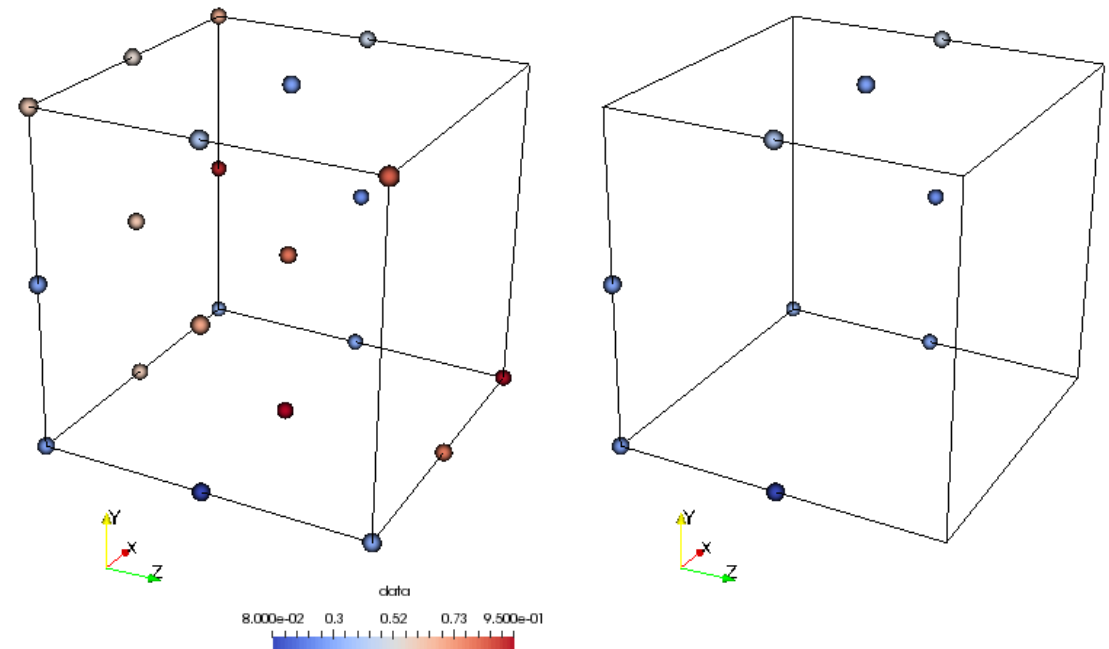
```
output.PointData.append(xv*x_hat + yv*y_hat + zv*z_hat, "radial velocity")
```


ParaView Programmable Filter to do selection

```
coords_z = inputs[0].Points[:,2]
data = inputs[0].PointData["data"]

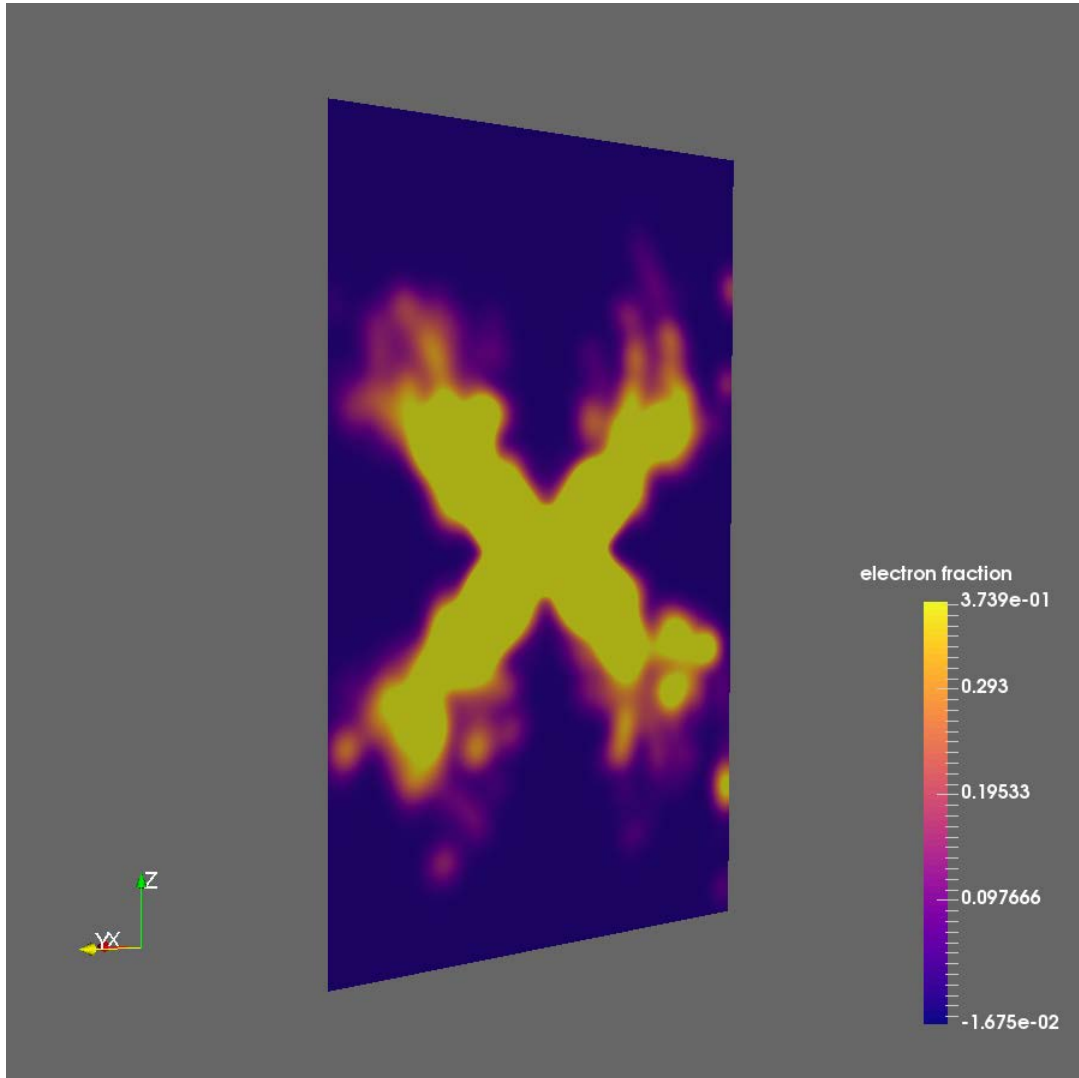
filtering = np.logical_and(data < .5, coord_z < 2)
indices = np.where(filtering)

output.Points = inputs[0].Points[indices]
output.PointData.append(data[indices], "data2")
```



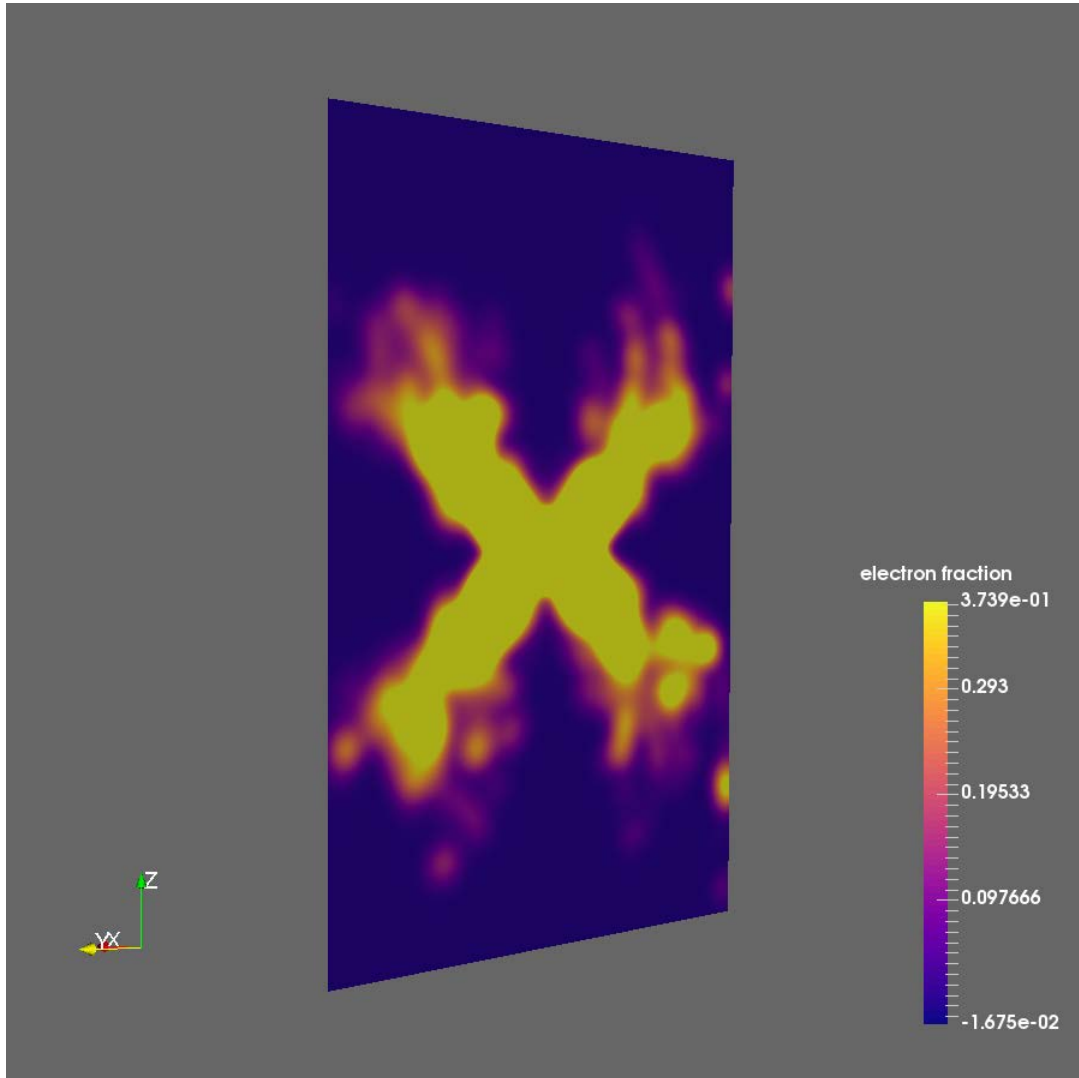
SPH Interpolators (New as of April 2016)

vtkSPHInterpolator, vtkSPH*Kernel



- Added in VTK in April 2016
- Will be released with ParaView 5.1
- The interpolators are available as a special category of Filters, providing the three basic sampling objects (line, slice, grid)
- The Point Sampling is using threaded approaches (vtkSMPTTools) which is typically much faster than any other point locators and is critical to the speed of the SPH operations.
- [kudos to Will Schroeder at Kitware]

vtkSPHInterpolator, vtkSPHQuinticKernel



- Use all particles within a cut-off sphere with a fixed kernel size / specified smoothing length h (called spatial step in ParaView). The cutoff distance (sphere around an interpolated point) is a function of the SPH kernel. A quintic kernel has cutoff distance $3 \cdot h$.
- The current implementation uses a gather method. For each point to be interpolated, the basis neighbors around the point are retrieved. The provided kernel is then invoked to perform the interpolation.

Can use a Progr. Filter to create a specific sampling geometry

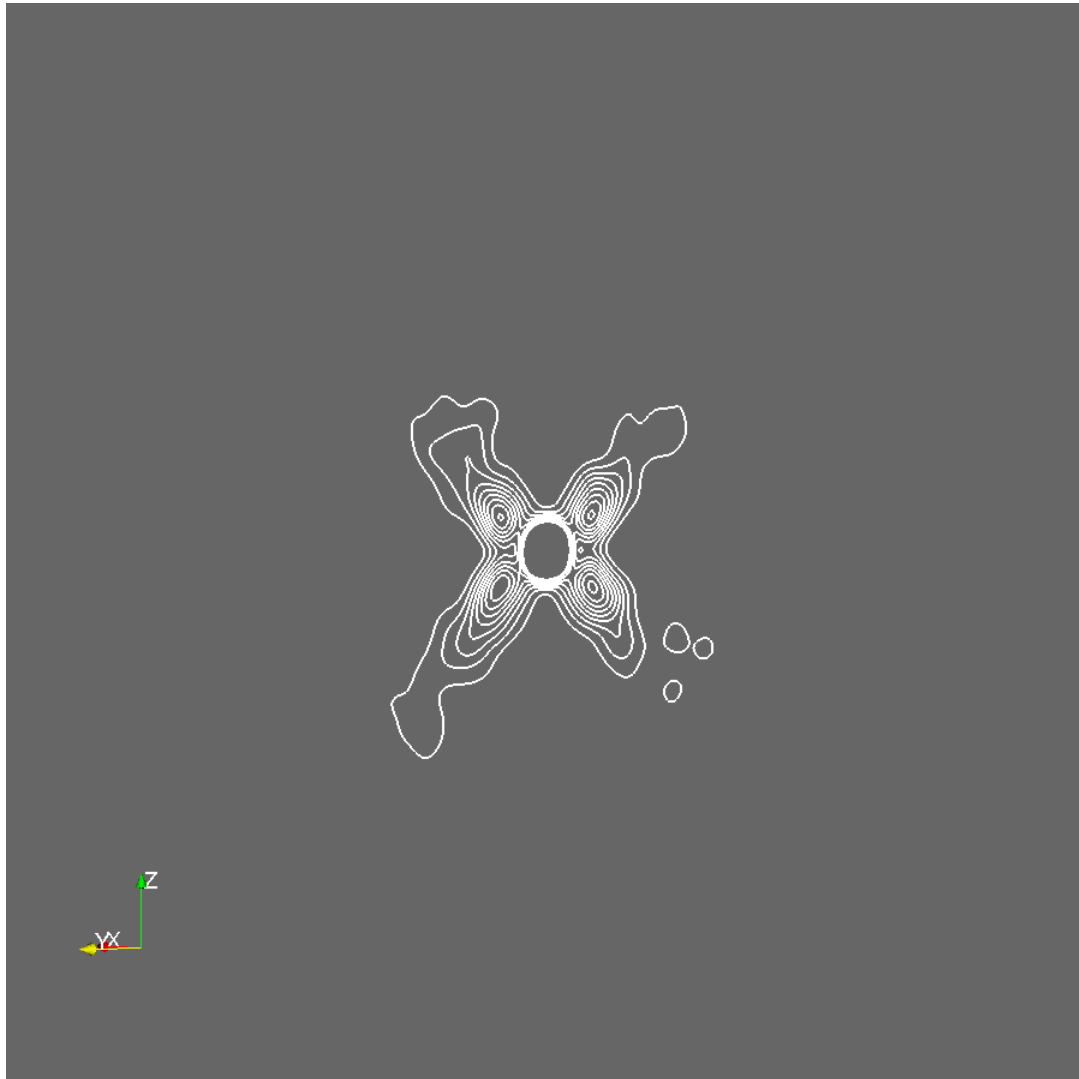
```
from vtk import vtkPlaneSource,  
vtkSPHInterpolator, vtkSPHQuinticKernel  
  
center = inputs[0].GetCenter()  
bounds = inputs[0].GetBounds()  
x_avg = (bounds[1]+bounds[0])*0.5  
plane = vtkPlaneSource()  
plane.SetResolution(800, 800)  
plane.SetOrigin(x_avg,bounds[2],bounds[4])  
plane.SetPoint1(x_avg,bounds[3],bounds[4])  
plane.SetPoint2(x_avg,bounds[2],bounds[5])  
plane.SetCenter(center)  
plane.SetNormal(1,0,0)  
plane.Update()
```

```
sphKernel = vtkSPHQuinticKernel()  
sphKernel.SetSpatialStep(.01)  
  
interpolator = vtkSPHInterpolator()  
interpolator.SetInputConnection(plane.GetOutputPort())  
interpolator.SetSourceData(inputs[0].VTKObject)  
interpolator.SetKernel(sphKernel)  
interpolator.SetDensityArrayName("Density")  
interpolator.SetMassArrayName("Mass")  
interpolator.Update()  
  
output.ShallowCopy(interpolator.GetOutput())
```

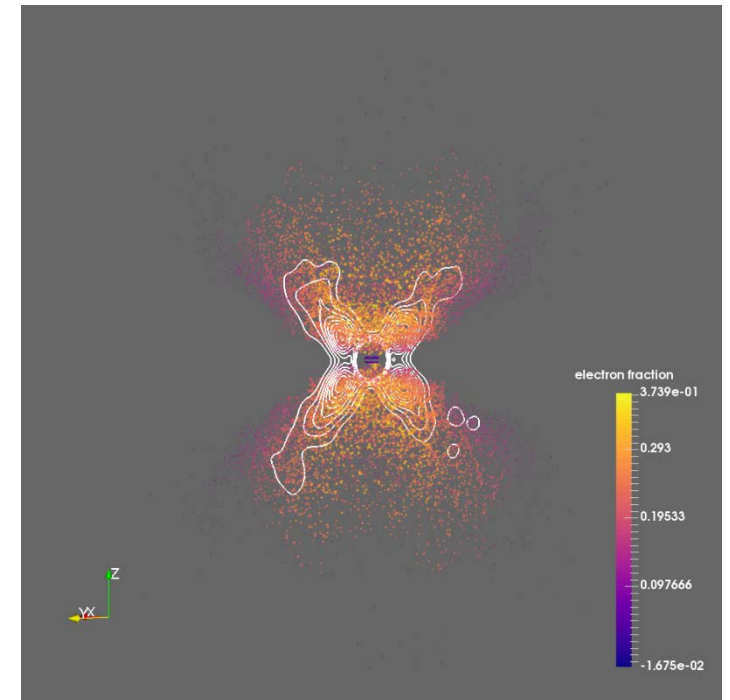
The ParaView GUI offers “standard” sampling geometries

- 3D planar cut
- 3D sub-volume
- High-resolution line probe

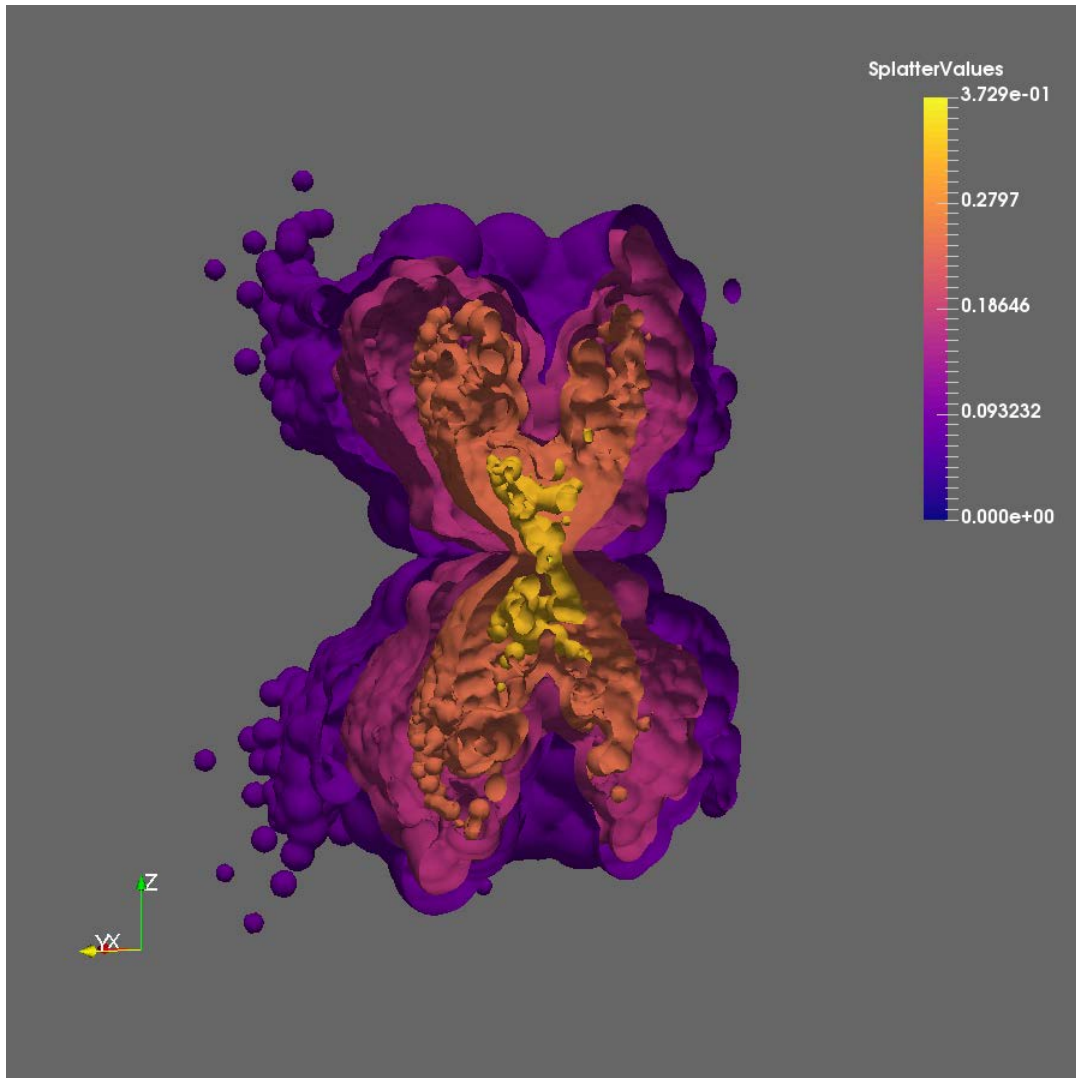
Data on a plane can be contoured



- The plane is a VTKPolyData and can be contoured



The volume (clipped), can be iso-contoured



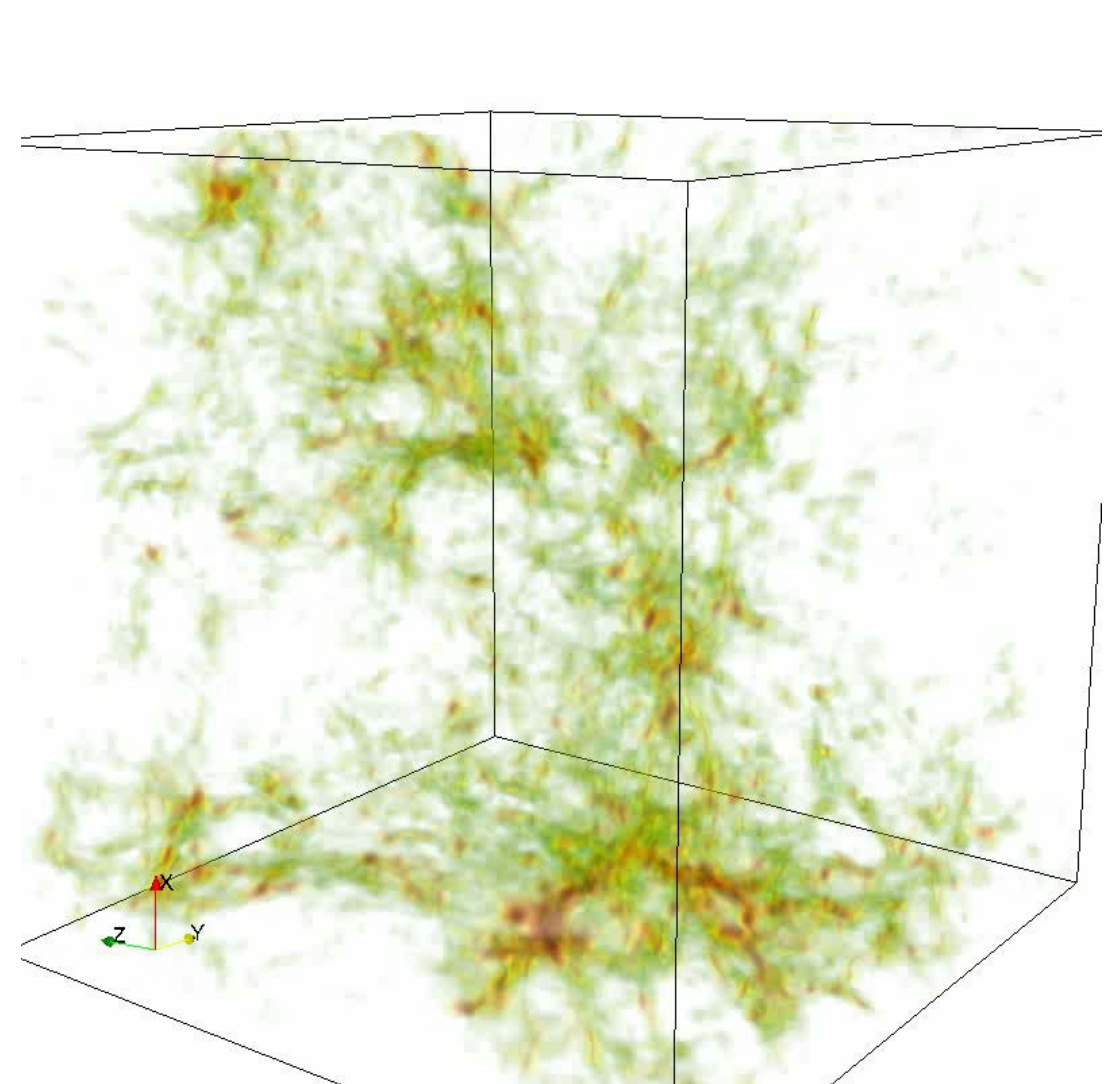
- The vtkSPHInterpolator and the vtkGaussianSplatter can be multi-threaded. Using Intel Threading Building Blocks (TBB) or other non-sequential type may improve performance significantly.
- On my laptop with 8-core, I have an 6x speedup, using Intel Threading Building Blocks (TBB).

“Volume rendering” can also be done

- To get a decent rendering, I interpolated my Gadget data on a small box inside the overall bounding box.
- Because I only had 56 millions particles, it did not make much sense to go to a high resolution volume.

example run-times (laptop 8 cores)

- 56 millions particles resampled in 400^3 grid in 64 seconds using a QuinticKernel.



Summary

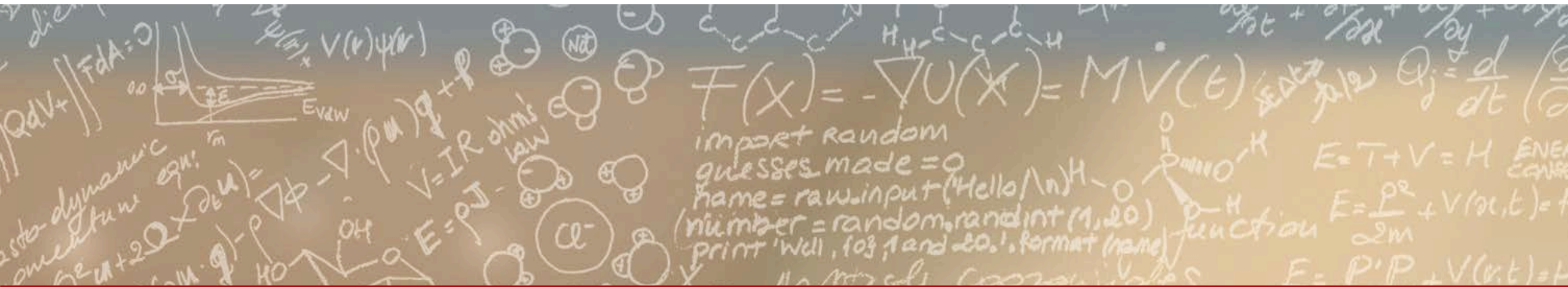
- We have taken a brief tour of ParaView with the emphasis on particle handling.
- We have given examples of data filtering in straight numpy form, and we've seen how the same syntax can be used (with a bit of wrapper code) from within ParaView.
- We've seen how the new SPH interpolator objects can be used.
- Exercises:
 - Create h5py data
 - Use the Python calculator
 - Use numpy array masks for filtering
 - Practice with SPH interpolator
 - Work with the Gadget source code reader
- AMR in ParaView
- Thanks to Ali Rahmati, U of Zurich for the Gadget dataset.



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.