

CSCS

Swiss National Supercomputing Centre



# ParaView Visualization Course

## Data formats, data types, data statistics

---

Dr. Jean M. Favre

Chief Scientist

Visualization Group Leader

# Motivations for Data Archiving

---

From the point of view of the *application code*

- Provide an archiving method which maps the application (memory footprint, i.e semantics) as close as possible
  - If data are stored in parallel, save to disk in parallel
  - If data are stored in 5D array, save to disk as a 5D array
- Requires the least amount of supplemental coding
- Allow compilation on heterogeneous platforms

# Motivations for Data Archiving

---

From the point of view of the *data*

- Allow human reading (a.k.a. ASCII)
- Allow portable (heterogeneous platforms) reading
- Allow partial read, sub-setting read/write
- Browsing/editing without developing code

# Motivations for Data Archiving

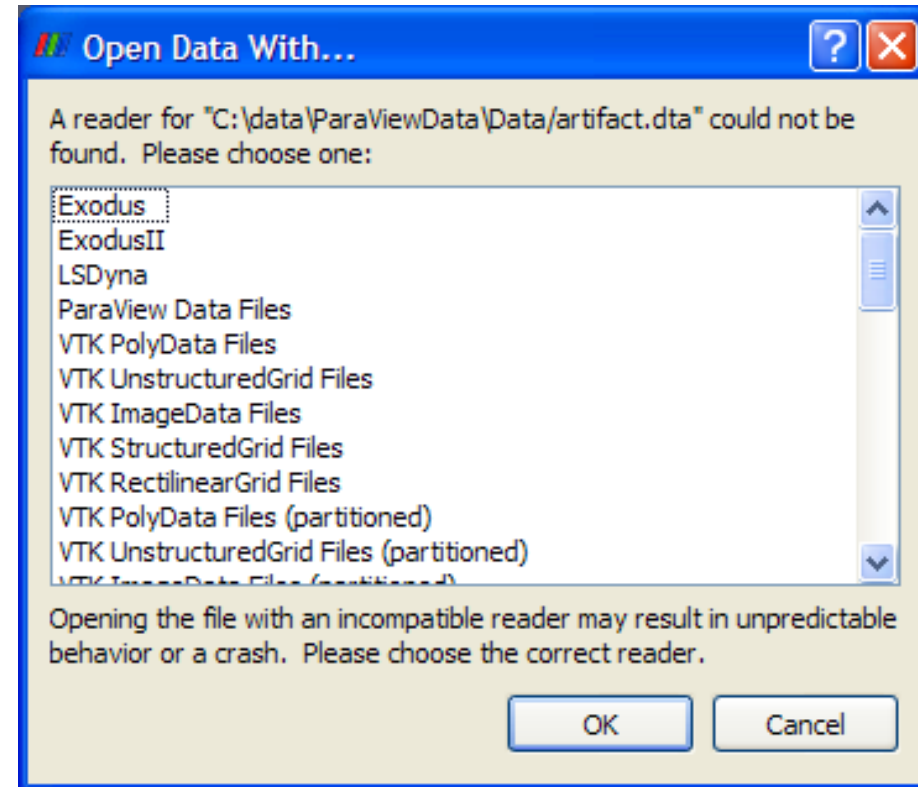
---

From the point of view of the *Visualization Tools*

- Ingest data “as-is”
- Allow random access (don’t read all, be selective)
  - Get to the meta-data easily
- Parallel/distributed/streaming read
  - Display without discontinuities

# ParaView/VTK data formats

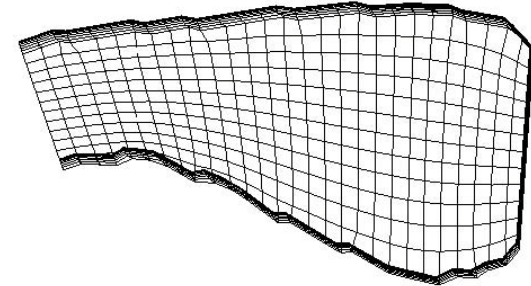
- ParaView supports gridded and mesh-less data and has support for time-dependent results.
- VTK File formats
- What file formats does ParaView support?



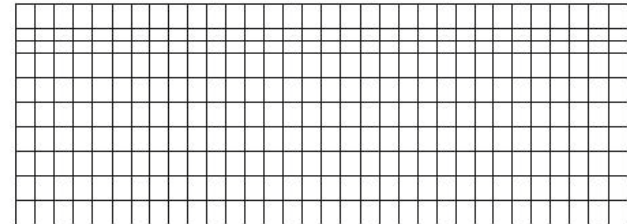
# ParaView/VTK data types

## 3 structured grid types

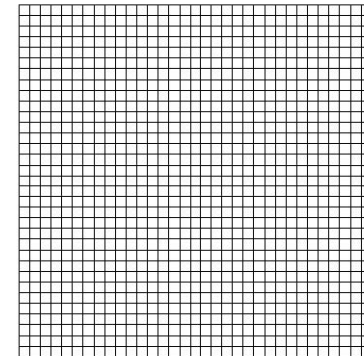
Curvilinear =>



Rectilinear =>



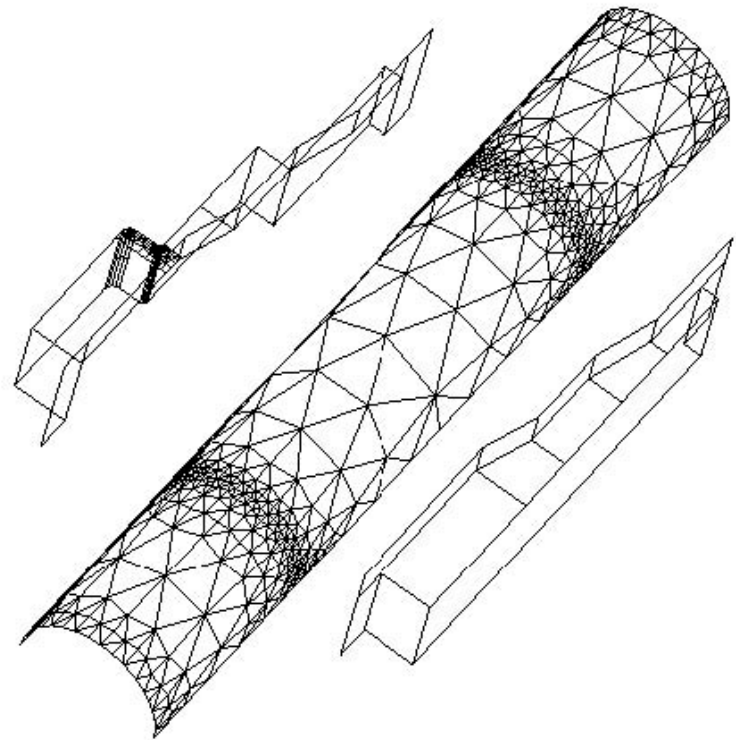
Uniform Rectilinear =>



# ParaView/VTK data types

## 2 unstructured grid types

1. PolyData (for geometry)
2. UnstructuredGrid



# ParaView/VTK data types

The screenshot shows the Properties panel in ParaView/VTK. It is divided into several sections: Properties, Statistics, Data Arrays, Bounds, and Time. The Data Arrays section contains a table of data arrays with their names, data types, and ranges. The Time section contains a table of time indices and their corresponding values.

Name	Data Type	Data Ranges
GlobalNodeId	idtype	[1, 10088]
DISPL	float	[0, 0], [0, 0], ...
VEL	float	[0, 0], [0, 0], ...
ACCL	float	[-4.96528e-0...]
PedigreeNodeId	idtype	[1, 10088]
BlockId	int	[1, 2]
GlobalElementId	idtype	[1, 7152]
EQPS	float	[0, 0]
PedigreeElementId	idtype	[1, 7152]




























Index	Value
0	0
1	0.000100074
2	0.000199905
3	0.000299964

Data values can be stored at the nodes, at the cells, using all available types (int, char, float, double, etc.)

Timestep indices and associated values will be shown if present



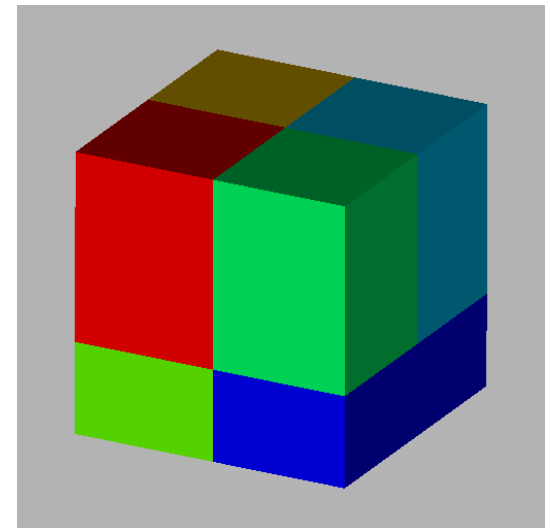
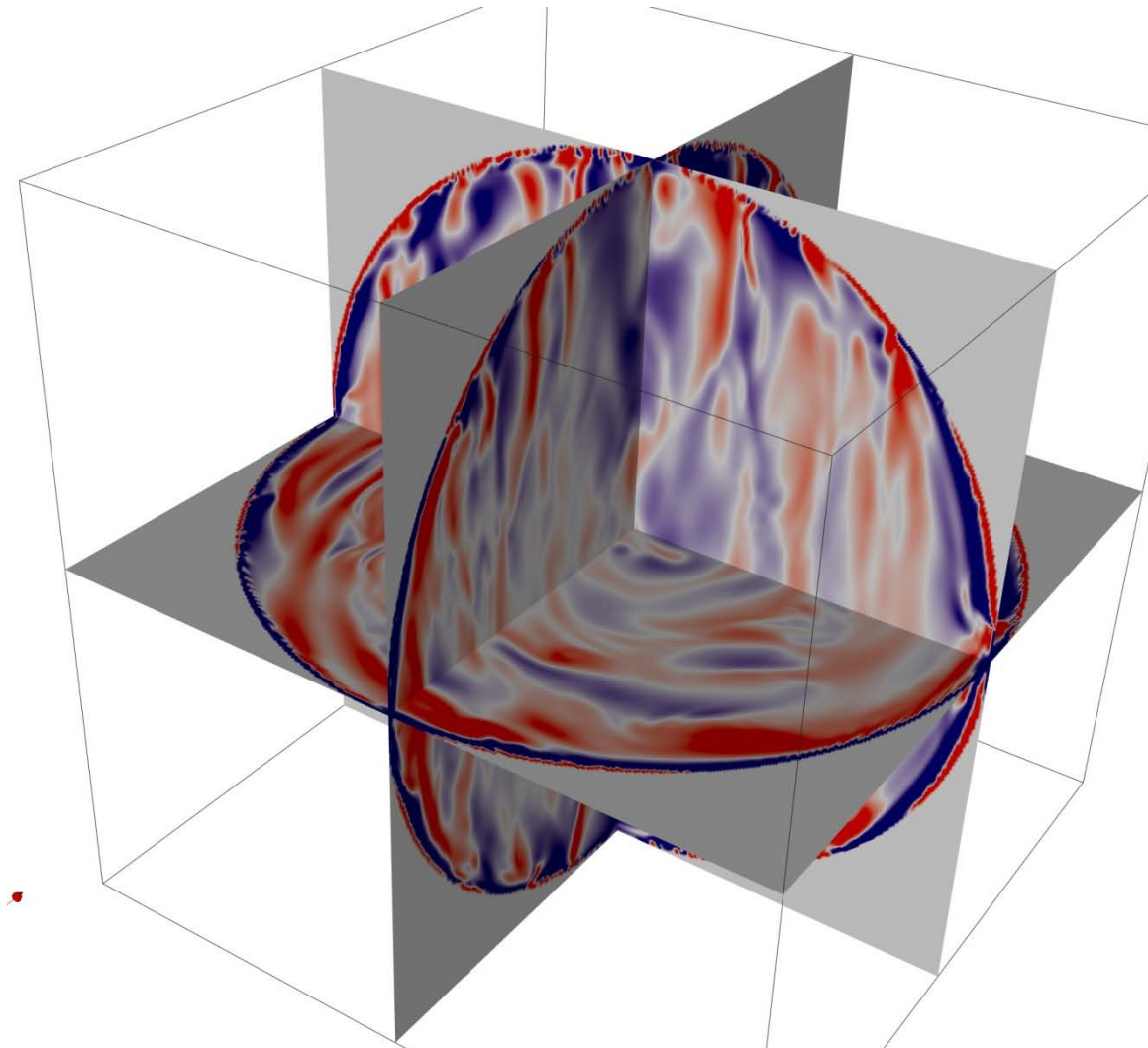
# ParaView “Data Object Generator”

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 ]

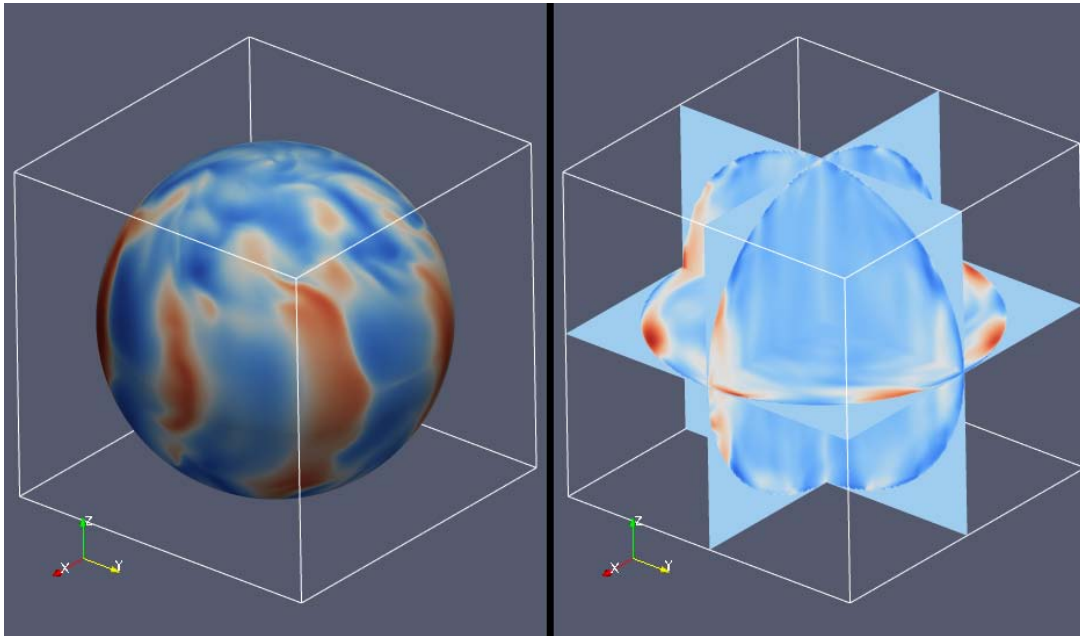
# Structured grids are split by IJK Extents

Use ExtractSubset

Parallel processing will enable requests for any subsets, including ghost-cells

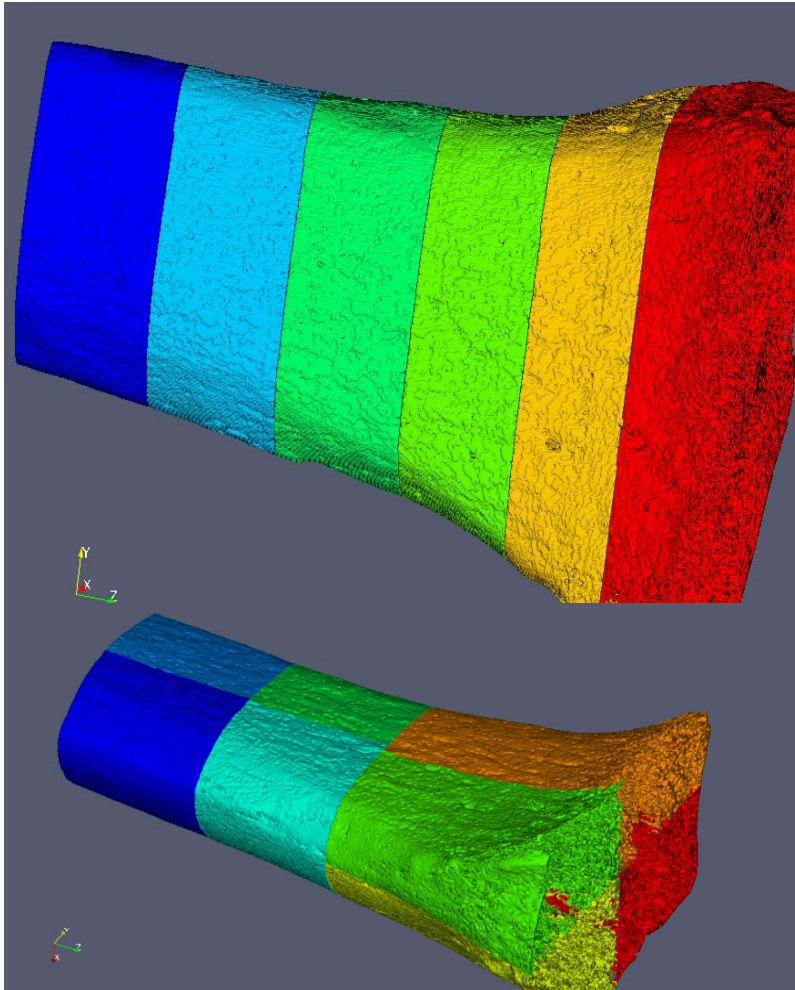


# Structured grids can become unstructured...



- many visualization filters transform structured grid data into unstructured data
- The memory footprint and cpu load turns into a titanic task
- $400^3$  clipped to 156 million cells in 1h27 minutes

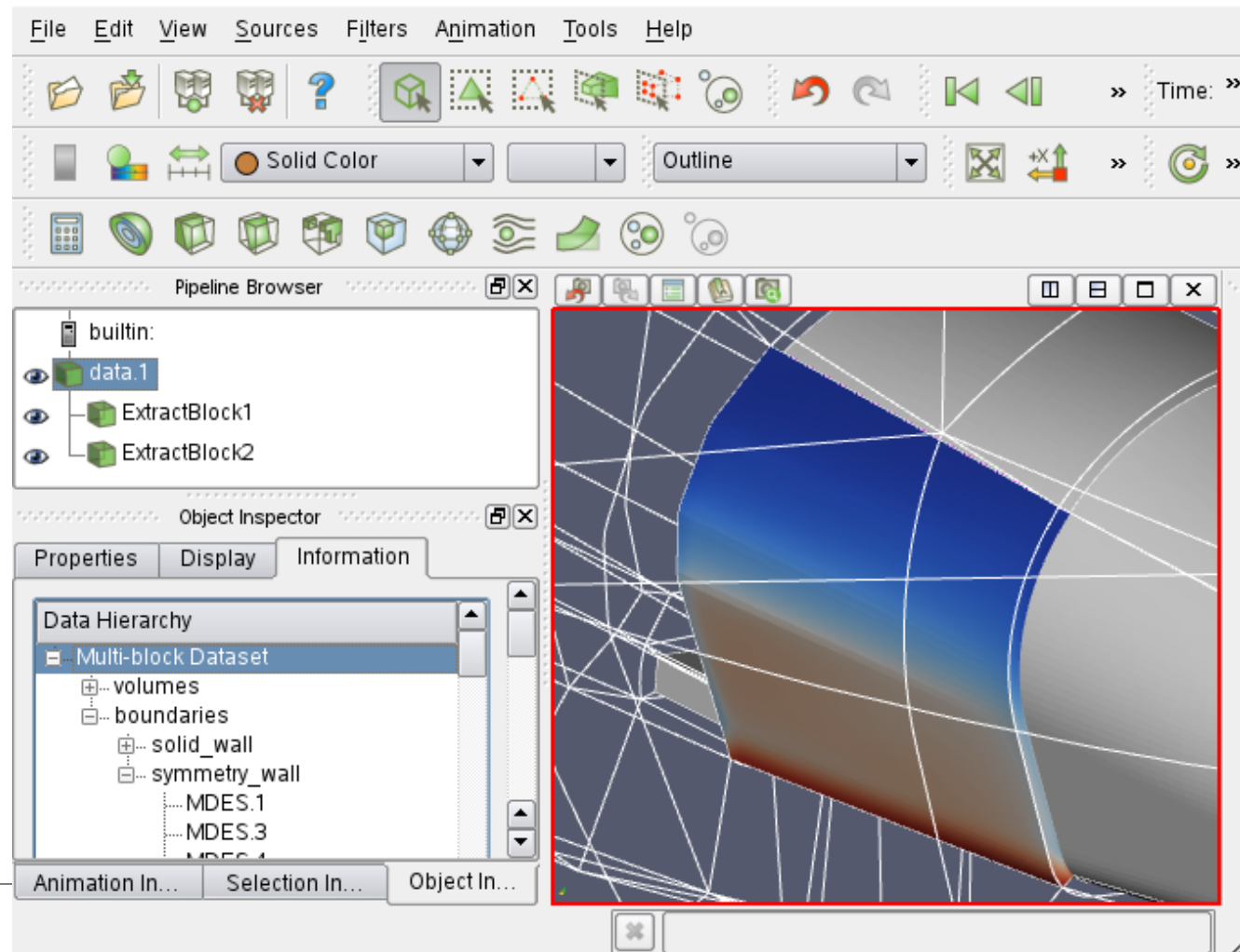
# Unstructured grids are split by XYZ Extents



- Use D3 to divide data
- D3 has 3 options
  - Assign cells uniquely
  - Duplicate cells
  - Divide cells
- For volume rendering, cells on the boundaries are copied to all regions and then clipped by the bounds of each region

# Composite datasets: Multi-block

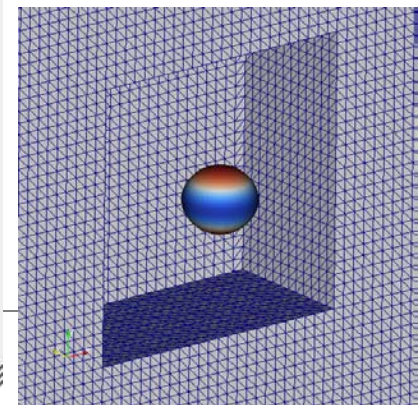
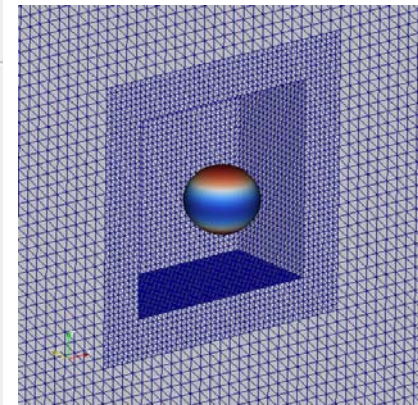
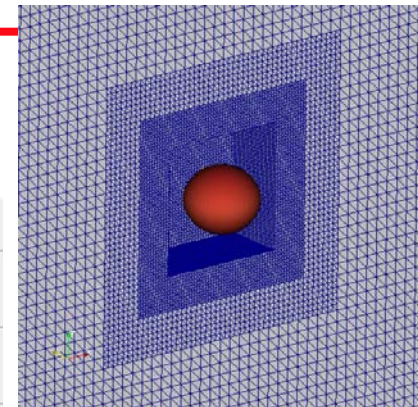
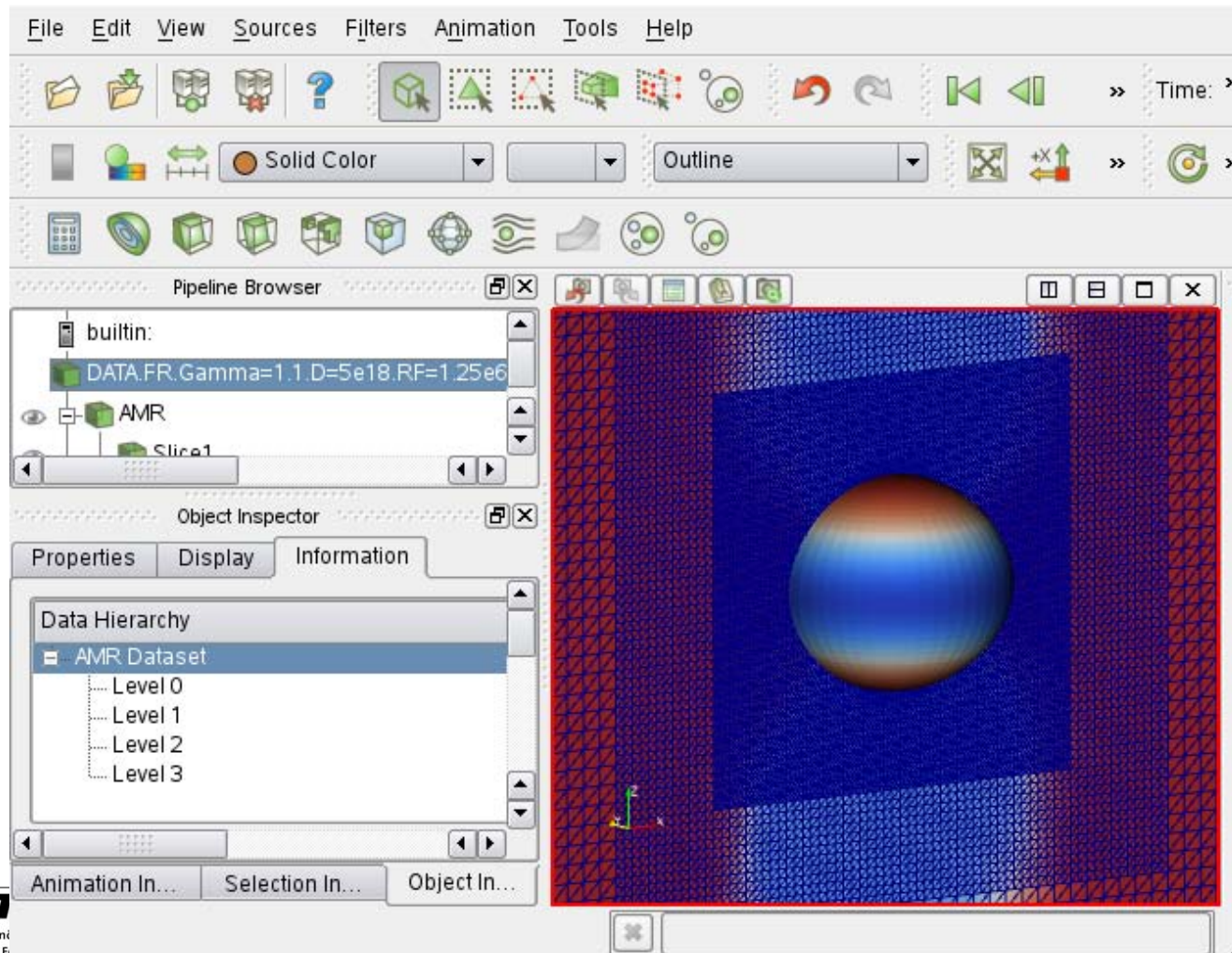
- Can be nested
- Sub-blocks can be selected for other processing





# Composite datasets: Hierarchical

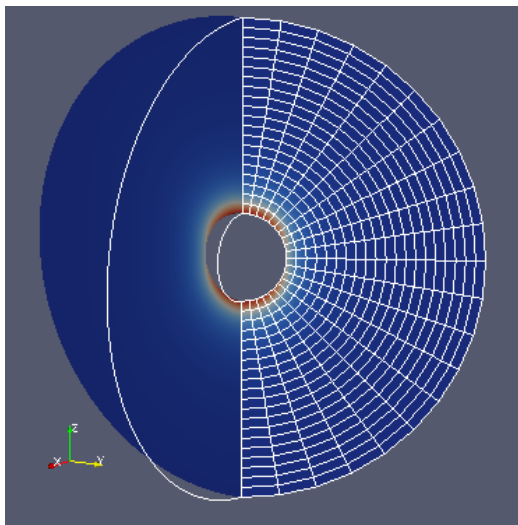
- Hyper-octree or AMR dataset



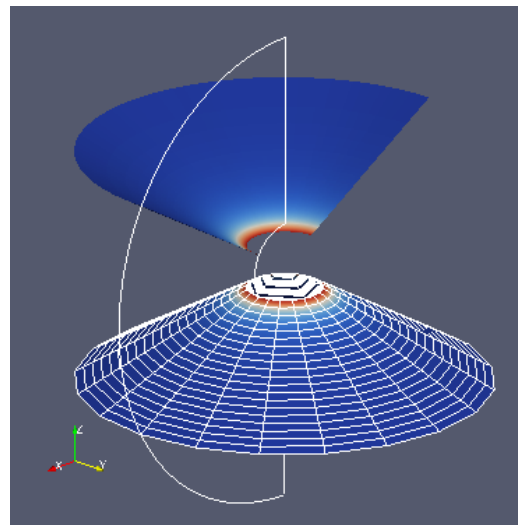
# Spherical [or cylindrical] Grids?

- VTK does not support spherical grids natively, but curvilinear grids can be used (with explicit x,y,z coordinates)
- Read the spherical axis data (phi, radius, theta) and create a curvilinear grid
- [Phi, Theta, R] is transformed into [I, J, K]

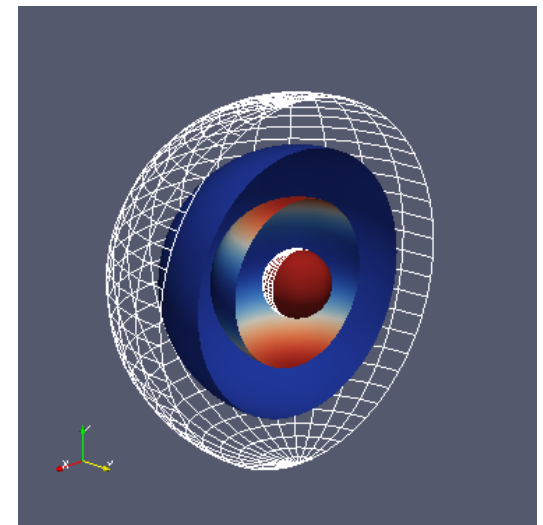
I-surfaces



J-surfaces



K-surfaces



# Time series

- Animating legacy VTK file series
- ParaView recognizes file series named using certain patterns including fooN.vtk, foo-N.vtk, foo.N.vtk, Nfoo.vtk, N.foo.vtk where N is an integer (with any number of leading zeros)
- `paraview --data=/project/F9999.hdf/F..vtk`



# VTK native formats

---

- “Legacy” file format
- “XML”-based formats support many more features to facilitate data streaming and **parallel I/O**. Some features of the format include support for compression, portable binary encoding, random access, big endian and little endian byte order, multiple file representation of piece data, and new file extensions for different VTK dataset types.
- Conversion utility in `VTK/Utilities/vtk2xml.py`
- `./bin/vtkpython vtk2xml.py legacy.vtk`

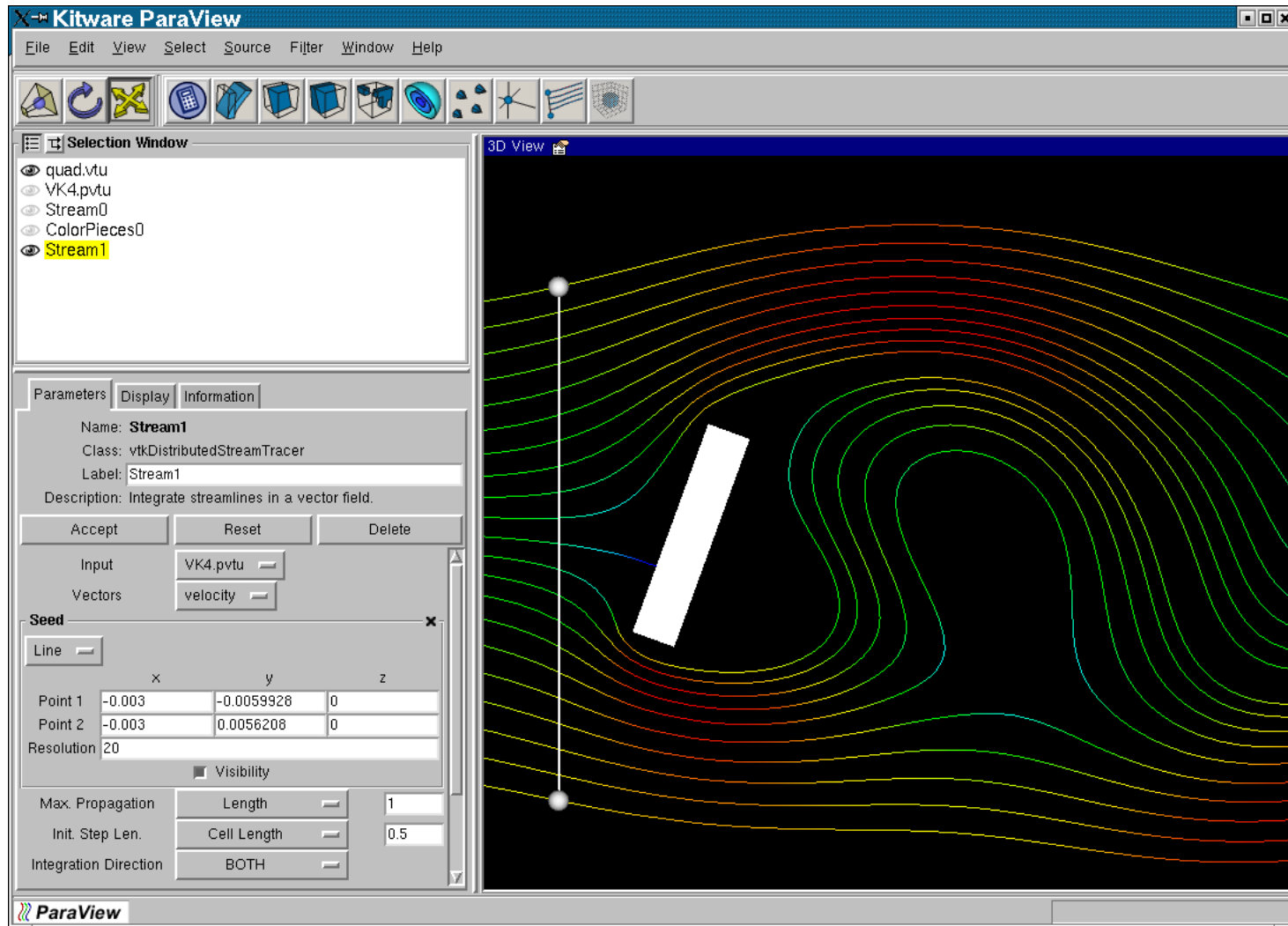
# XML format example with 4 partitions

```

<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian">
  <PUnstructuredGrid GhostLevel="0">
    <PPointData Scalars="velocity">
      <PDataArray type="Float32" Name="velocity" NumberOfComponents="3"/>
    </PPointData>
    <PPoints>
      <PDataArray type="Float32" NumberOfComponents="3" format="appended"/>
    </PPoints>
    <Piece Source="upperleft.vtu"/>      <Piece Source="upperright.vtu"/>
    <Piece Source="bottomleft.vtu"/>    <Piece Source="bottomright.vtu"/>
  </PUnstructuredGrid>
</VTKFile>

```

# XML format example with 4 partitions

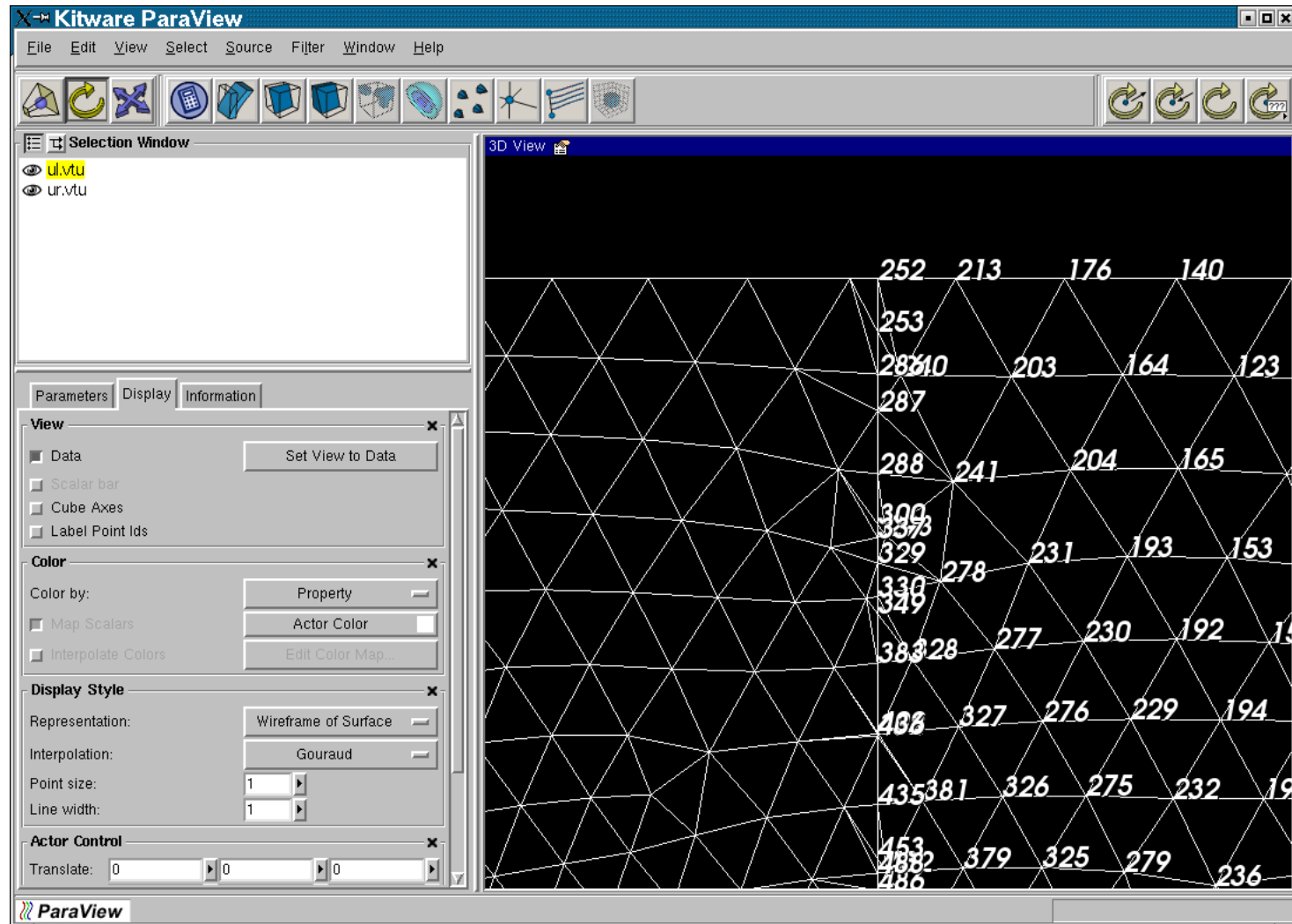


# XML format example with 4 partitions

Each piece uses its own internal node numbering.

No need for global renumbering

Each piece is a 100% compliant dataset. Can be read stand-alone



# XML format example with ghost cells

```

<VTKFile type="PStructuredGrid" version="0.1">
  <PStructuredGrid WholeExtent="0 65 0 65 0 65" GhostLevel="1">
    <Piece Extent=" 0 17 0 17 0 65" Source="d0372_00.vts"/>
    <Piece Extent="16 33 0 17 0 65" Source="d0372_01.vts"/>
    <Piece Extent="32 49 0 17 0 65" Source="d0372_02.vts"/>
    <Piece Extent="48 65 0 17 0 65" Source="d0372_03.vts"/>
    <Piece Extent=" 0 17 16 33 0 65" Source="d0372_04.vts"/>
    <Piece Extent="16 33 16 33 0 65" Source="d0372_05.vts"/>
    <Piece Extent="32 49 16 33 0 65" Source="d0372_06.vts"/>
    ....
  </PStructuredGrid>
</VTKFile>

```

# How to write partitioned files? Structured Grids

*// Use vtkXMLP\*Writer with a serial program*

*int N = 4;*

*vtkXMLPImageDataWriter piw =  
    vtkXMLPImageDataWriter::New();*

*piw->SetInputConnection(reader->GetOutputPort());*

*piw->SetFileName("/pathtofilename/file.pvti");*

*piw->SetNumberOfPieces(N);*

*piw->SetStartPiece(0);*

*piw->SetEndPiece(N-1);*

*piw->WriteSummaryFileOn();*

*piw->Write()*

# How to write partitioned files? Unstructured Grids

*running pvpython with a parallel pvserver*

```
from paraview import servermanager as sm
sm.Connect("name") # with the name of your host running pvserver
reader =
    sm.sources.XMLUnstructuredGridReader(FileName="mesh.vtu")
```

```
d3 = sm.filters.D3()
d3.Input = reader
d3.BoundaryMode = 0 #Assign cells uniquely
```

```
writer =
    sm.writers.XMLPUnstructuredGridWriter(FileName="umesh.pvtu")
writer.NumberOfPieces = 4
writer.Input = d3
writer.UpdatePipeline()
```

# Distributed Data Archiving

- Distributed archiving allows two different data processing methods:
  - Parallel visualization (previous example)
  - Piece-wise visualization
- The VTK format is just one example. We can use other data formats and still use distributed reading/processing in ParaView.
- => for example, HDF5 and Xdmf



# Xdmf (eXtensible Data Model and Format)

**Data Format** refers to the raw data to be manipulated.

Information like number type ( float, integer, etc.),  
precision, location, rank, and dimensions

**Light** data: description of the data (using XML)

**Heavy** data: the values themselves (using HDF5)

Light data is small and can be shared between modules

Heavy data may be potentially enormous; copying  
needs to be kept to a minimum.

# Xdmf (eXtensible Data Model and Format)

**Data Model** refers to the intended use of the data.

For example, a three dimensional array of floating point values may be the X,Y,Z geometry for a grid or calculated vector values.

A data model only describes the data, it is purely light data and thus stored using XML.

It is targeted at scientific simulation concentrating on scalars, vectors, and tensors defined on some computational grid.

Structured and Unstructured grids are described via their topology and geometry.

Calculated, time varying data values are described as attributes.

Actual values for the grid geometry, connectivity, and attribute values are contained in the data format.

# Advantages of Xdmf

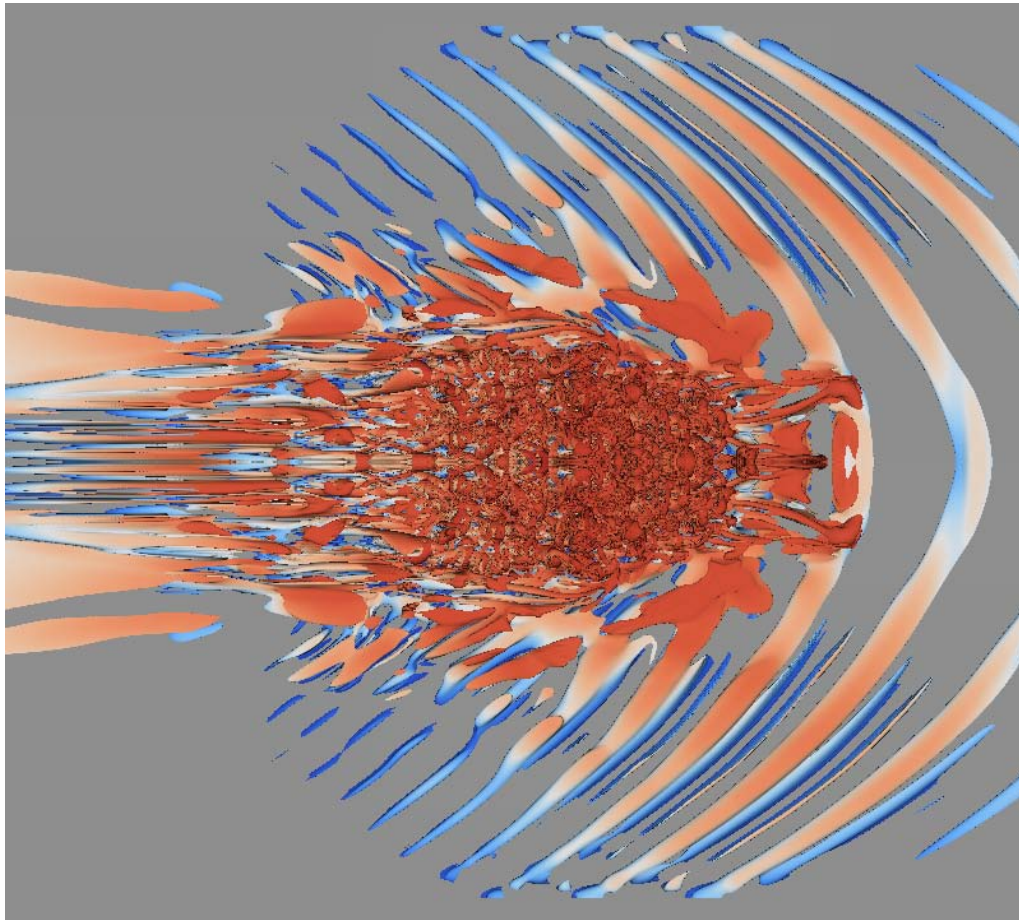
- Raw HDF5
- Some of the HDF5 paradigms are percolated to the XML model (striding, interlacing, on-the-fly data conversion)
- The model allows for a richer definition of the intended use (even if visualization softwares cannot, today, handle the data as advertised). E.g. storing data at the cells centers, faces, edges....
- The model facilitates data sharing (hyperlinks, array indexing, etc...)

# Some practical examples at CSCS

---

- Existing HDF5 encoding. We write the XML wrapper and ParaView can read the data.
- Derived quantities can be calculated and added *lightly* to the data model. The Xdmf model can reference multiple HDF5 files
- Each node of a running simulation can write its own HDF5 datafile (Not worrying about gathering data, or mpi-io). An Xdmf wrapper can concatenate all pieces.
- Fixed geometry and transient data. The geometry can be shared with *lightweight* links.

# Example for a Rectilinear Grid



ParaView can read the data on any number of processors

1. ParaView can read any subsets (hyperslabs)
2. pvpython will read only the hyperslab necessary

# Example of HDF5 & XML encoding

```
<Xdmf><Domain Name="Data">
  <Grid GridType="Collection" CollectionType="Temporal">
<Grid Name="dns"> <Time Value="0.000"/>
<Topology Type="3DRECTMESH" Dimensions="595 1049 1900">
  </Topology>
  <Geometry Type="VXVYVZ">
    <DataStructure NumberType="Float" Rank="1" Name="X"
      Precision="4" Dimensions="1900" Format="HDF">
full.000.h5:/coords/X </DataStructure>
    <DataStructure NumberType="Float" Rank="1" Name="Y"
      Precision="4" Dimensions="1049" Format="HDF">
full.000.h5:/coords/Y </DataStructure>
    <DataStructure NumberType="Float" Rank="1" Name="Z"
      Precision="4" Dimensions="595" Format="HDF">
full.000.h5:/coords/Z </DataStructure>
  </Geometry>
</Grid>
</Domain>
</Xdmf>
```

# Example of HDF5 & XML encoding

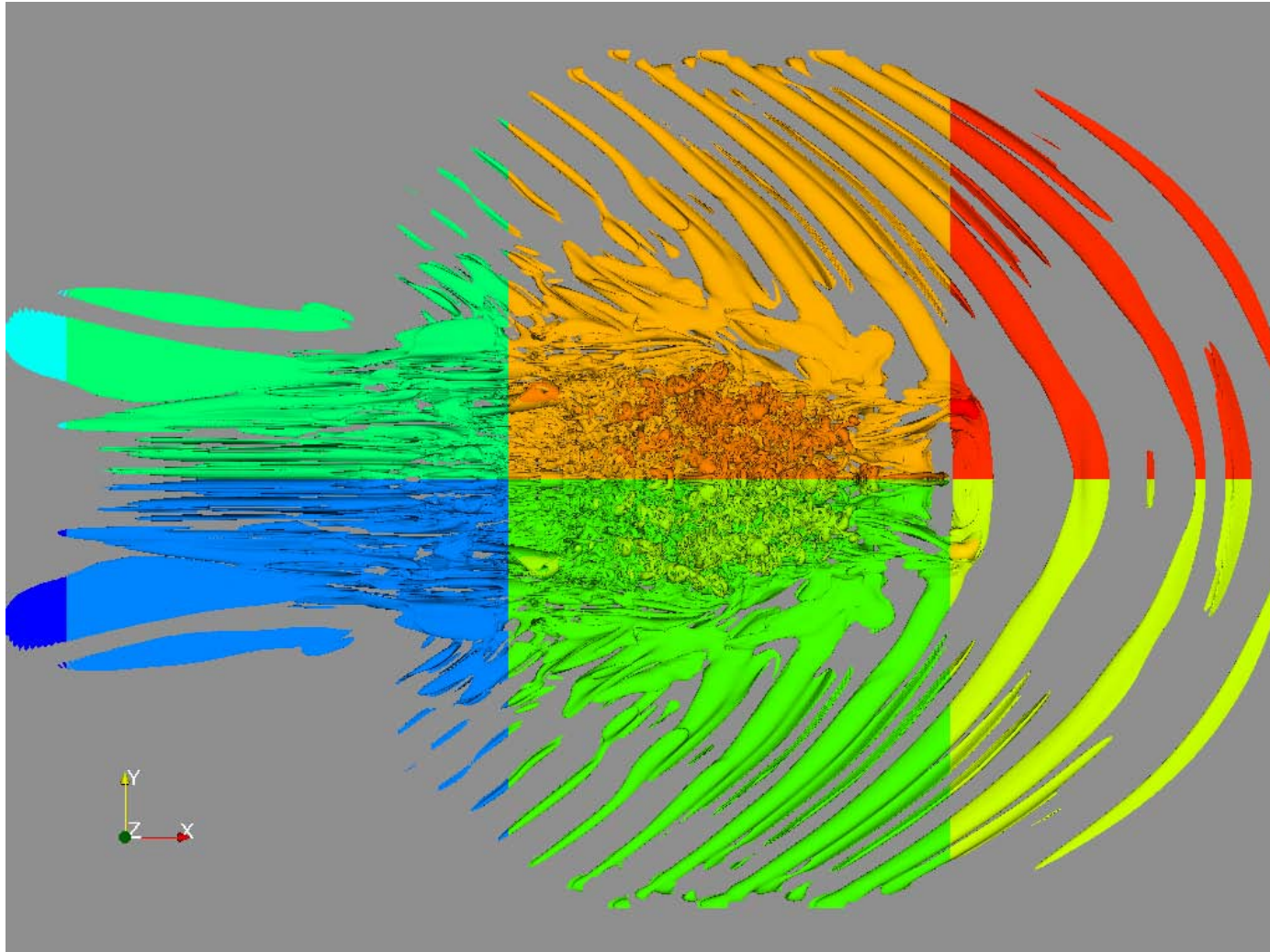
```

<Attribute
  Active="1"
  Type="Scalar"
  Center="Node"
  Name="lambda_2">
  <DataStructure DataType="Float" Precision="4"
    Dimensions="595 1049 1900"
    Format="HDF">
    full.000.h5:/data/lambda_2
  </DataStructure>
</Attribute>

```



# Running on 8 pvservers





# Optimizing the reading order (X, Y or Z)

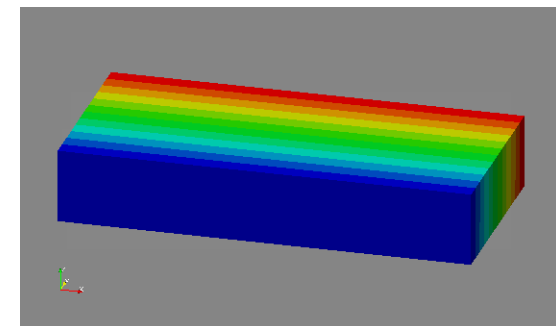
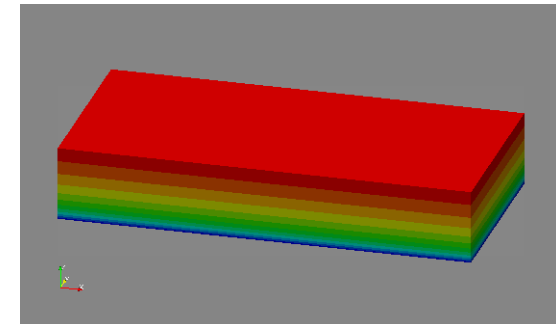
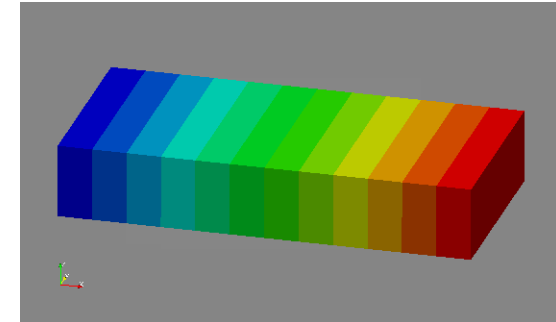
Reading 15 Gb of data with 12 cpus, with  
HDF5 hyperslabs

X hyperslabs: average read: 430 secs

Y hyperslabs: average read: 142 secs

Z hyperslabs: average read: 36 secs

Parallel Visualization is ALL about file I/O



# Zooming in to the interesting zone



How much data was read, isosurfaced, and never displayed in this picture?

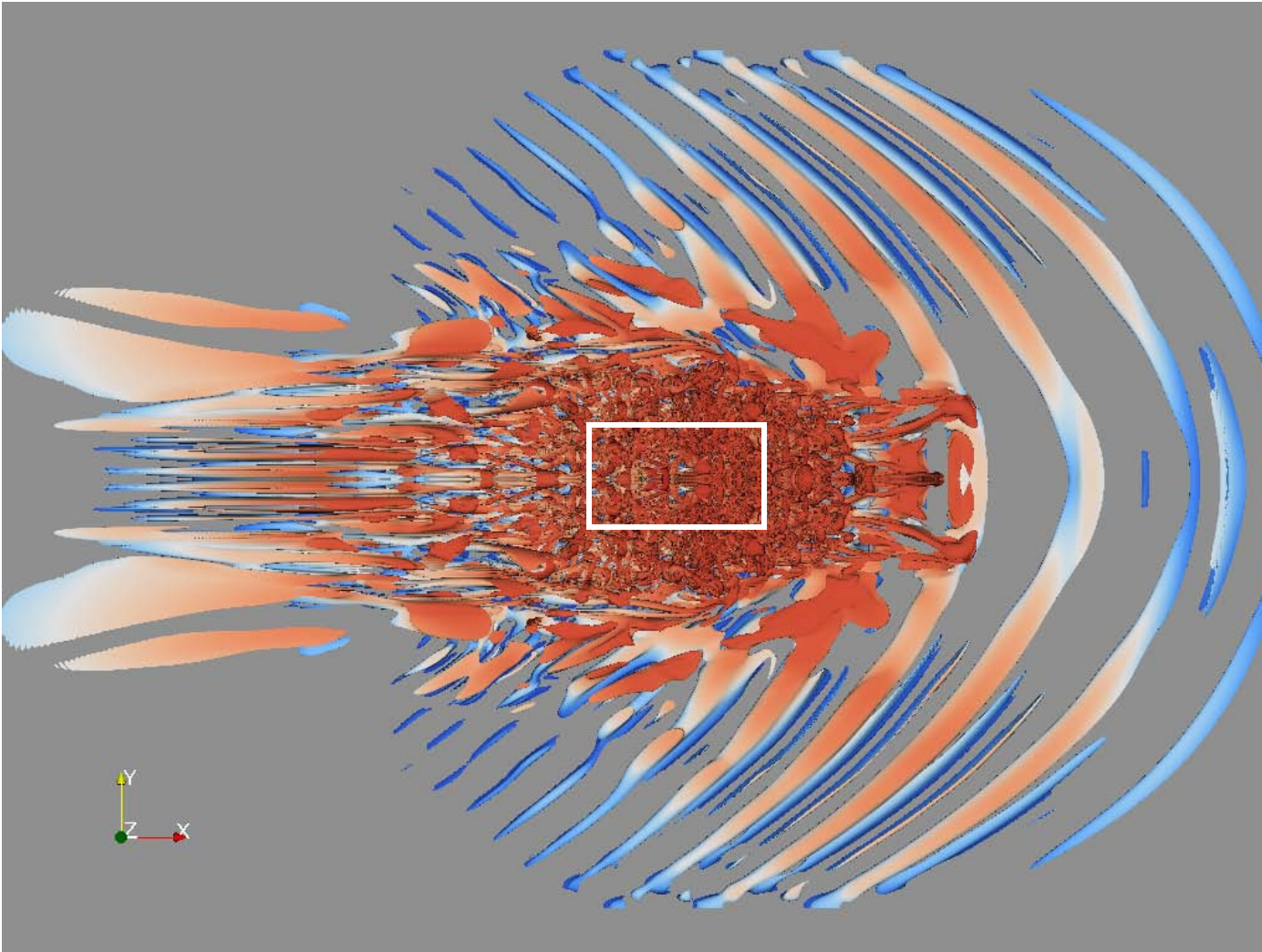


# Adjusting the Data Extents...

Reading much less data

display only 1/40-th of the data volume

25 millions instead of one billion cells



# Summary

---

- VTK data types and data formats offer all standard grid shapes.
- Beware that the VTK XML formats are limited by some 32-bit pointers.
- Many engineering formats are supported, some very well, some less efficiently
- Custom formats readers can be added but a thorough knowledge of internal data structures is required.
- Extensions are coming (SQL databases, Tables, ...)