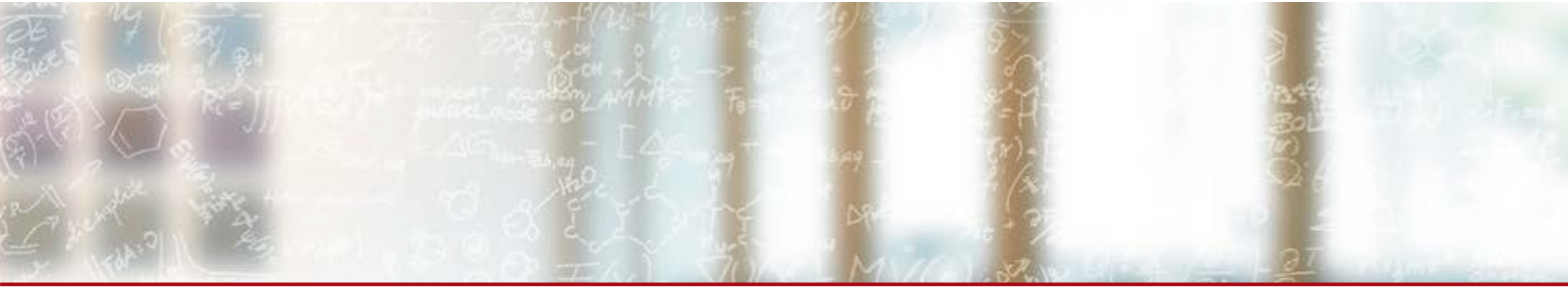
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Data formats, parallel I/O and visualization

Jean M. Favre
Visualization Task Leader

Agenda

- Data formats
- Parallel I/O with MPI-IO

- Scientific Visualization with VisIt
  - General purpose visualization
  - miniApp visualization

  - We will follow some of the contents of the online VisIt tutorial

  - Demonstration of in-situ visualization

Prelude

Data formats

- Interface between simulations and visualization
- Many formats exist. Pick the most appropriate
- High level libraries (HDF5, netCDF, …)
- Raw data vs. self.described formats
- Parallel I/O

Data formats

Purpose of I/O

- Archive results to file(s)
- Provide check-point / restart files
- Analysis
- Visualization
- Debugging simulations

Requirements

- Fast, parallel, selective
- Independent of the number of processors (tasks)
- Self-documented

Data formats

- Community specific
  - CGNS, CCSM, NEK5000, H5Part, PDB
- Ad-hoc
  - ~~Make up your own~~
  - Many formats exist. Choose the most appropriate
  - High level libraries (HDF5, netCDF, …)

I/O strategies

- Serial I/O: each process writes its own output.
  - Make sure each single file can be read independently
  - Have a method to combine them all (without file concatenation)
- Parallel I/O: a collective operation writes a single file, each process placing data at the correct offset.
- Transient data: use one of the above method, for every N timesteps.

Parallel processing, but serial I/O:

Each process writes its own file independently,
A 64x64x64x4 block of floats

```
re csstaff 1048576 2013-11-25 14:04 benchmark.00000.0004.bof
re csstaff 1048576 2013-11-25 14:04 benchmark.00001.0004.bof
re csstaff 1048576 2013-11-25 14:04 benchmark.00002.0004.bof
re csstaff 1048576 2013-11-25 14:04 benchmark.00003.0004.bof
re csstaff 1048576 2013-11-25 14:04 benchmark.00004.0004.bof
re csstaff 1048576 2013-11-25 14:04 benchmark.00005.0004.bof
```

Each file can also be gzipped

```
re csstaff 22177 2013-11-25 14:04 benchmark.00000.0004.bof.gz
re csstaff 22094 2013-11-25 14:04 benchmark.00001.0004.bof.gz
re csstaff 22398 2013-11-25 14:04 benchmark.00002.0004.bof.gz
re csstaff 21958 2013-11-25 14:04 benchmark.00003.0004.bof.gz
re csstaff 21838 2013-11-25 14:04 benchmark.00004.0004.bof.gz
re csstaff 22220 2013-11-25 14:04 benchmark.00005.0004.bof.gz
re csstaff   422 2013-11-25 14:26 benchmark.0004.bov
```

The miniApp output is very simple, but still poses many challenges

Read it in VisIt. Visualize in parallel?

- Find gradient, magnitude of gradient
- Find erroneous data. Where? i,j coordinates? rank ? value? neighbors?
- Plot node's value over time?
- Find mean, at current time, a mean-over-time?
- Find first timestep when "condition is true"
- Compare serial output with:
  - OpenMP output ?
  - MPI output ?
  - CUDA output ?
- Compare solution at time T1, with solution at time T2?
- Compare solution on grids of different resolutions?

The miniApp out is raw-binary data

- What is good about it?

Performance! Subsettting!

- What is bad about it?

What's inside? variable name? resolution? Endiandness? Precision? Intended use? Who created it? When? Which version of code? Which compiler? Architecture?

Meta-data versus raw-data

- We'll see two ways to define the meta-data necessary to be able to make sense of the binary data

- BOV format (starting at page 9)
- Xdmf format

Brick of Values (BOV) format read by VisIt

VisIt can read raw binary data with the following header file

```
# BOV version: 1.0
# serial I/O output file
TIME: 0.01
DATA_FILE: output.80.000.bin
DATA SIZE: 64 64 1
DATA_ENDIAN: LITTLE
DATA FORMAT: DOUBLE
VARIABLE: phi
CENTERING: nodal
BRICK SIZE: 0.5 0.5 1.0
BRICK ORIGIN: 0.0 0.0 0.0
```

The raw data can also be read by numpy

```
import numpy as np
phi = np.fromfile(“output.bin”, dtype=np.float64, count=-1, sep=“”)
```

What are the dimensions of this array?

```
phi.shape  => (16384,)
```

We will need to use it as an array of dimensions (128, 128) later on…

Let us put some erroneous data in the file

```
import numpy as np
phi = np.fromfile(“output.bin”, dtype=np.float64, count=-1, sep=“”)

phi[10000:10010] = -1

phi.tofile(“output.bin”, sep=“”)
```

Use independent (serial) I/O

- Create one file per process

Brick of Values (BOV) format read by VisIt

VisIt can put all the serial pieces together with the following header

```
# BOV version: 1.0
# serial I/O output files, recombined
TIME: 0.01
DATA_FILE: benchmark.200.%03d.bin
DATA SIZE: 128 128 1
DATA_BRICKLETS: 64 64 1
DATA FORMAT: DOUBLE
DATA_ENDIAN: LITTLE
VARIABLE: phi
BRICK ORIGIN: 0.0 0.0 0.0
BRICK SIZE: 1.0 1.0 1.0
```

Xdmf format read by VisIt

Xdmf provides the “Data model”, i.e. the intended use of the data

Its size, shape, dimensions, variable names, etc…

Data format refers to the raw data to be manipulated. Information like number type (float, integer, etc.), precision, location, rank, and dimensions completely describe the any dataset regardless of its size.

The description of the data is separate from the values themselves.

We refer to the description of the data as **Light** data and the values themselves as **Heavy** data. Light data is small and can be passed between modules easily. Heavy data may be potentially enormous.

Example: a three dimensional array of floating point values may be the X,Y,Z geometry for a grid or calculated vector values. Without a data model, it is impossible to tell the difference.

Xdmf enables a richer description than BOV

- A Domain can have one or more *Grid* elements. Each Grid contains a *Topology*, *Geometry*, and zero or more *Attribute* elements. Topology specifies the connectivity of the grid while Geometry specifies the location of the grid nodes. Attribute elements are used to specify values such as scalars and vectors that are located at the node, edge, face, cell center, or grid center.
- **Example: Structured**
  - 2DSMesh - Curvilinear
  - 2DRectMesh - Axis are perpendicular
  - 2DCoRectMesh - Axis are perpendicular and spacing is constant
  - 3DSMesh
  - 3DRectMesh
  - 3DCoRectMesh

Xdmf format can describe the raw data

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf xmlns:xi="http://www.w3.org/2003/XInclude" Version="2.2">
  <Domain>
    <Grid Name="Mesh" GridType="Uniform">
      <Topology TopologyType="3DCORECTMESH" Dimensions="1 128 128"/>
      <Geometry GeometryType="ORIGIN_DXDYDZ">
        <DataItem Name="Origin" NumberType="Float" Dimensions="3" Format="XML">0. 0. 0.</DataItem>
        <DataItem Name="Spacing" NumberType="Float" Dimensions="3" Format="XML">1. 1. 1.</DataItem>
      </Geometry>
      <Attribute Name="phi" Active="1" AttributeType="Scalar" Center="Node">
         <DataItem Dimensions="1 128 128" NumberType="Float" Precision="8“ Format="Binary">output.bin</DataItem>
      </Attribute>
    </Grid>
  </Domain>
</Xdmf>
```

#N. B. Dimensions are given in Z, Y, X order

What is HDF5?

An HDF5 file is a container for storing a variety of scientific data and is composed of two primary types of objects

- HDF5 group: a grouping structure containing zero or more HDF5 objects, together with supporting metadata
- HDF5 dataset: a multidimensional array of data elements, together with supporting metadata

Any HDF5 group or dataset may have an associated attribute list. An HDF5 attribute is a user-defined HDF5 structure that provides extra information about an HDF5 object.

Working with groups and datasets is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, an HDF5 object in an HDF5 file is often referred to by its full path name (also called an absolute path name).

- / signifies the root group.
- /foo signifies a member of the root group called foo.
- /foo/zoo signifies a member of the group foo, which in turn is a member of the root group.

The raw data can also be re-written as HDF5

```
import numpy as np
import h5py
phi = np.fromfile(“output.bin”, dtype=np.float64, count=-1, sep=“”)
out = h5py.File("output.h5","w")
g_id = out.create_group("data")
g_id.create_dataset("phi", (128, 128), np.double, phi)
out.close()
```

Use “hdfview”, “h5ls –r”, “h5dump –d data/phi”

The raw data can also be converted as HDF5

We need a Data Description Language (DDL) input file:

```
HDF5 "output.h5" {
DATASET "/data/phi" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( 128, 128 ) / ( 128, 128 ) }
   DATA {
   }
}
}
```

```
H5import output.bin –c ddl.txt –o output.h5
```

See online reference

Advantages of HDF5

Self-described (but we're still missing the meaning of the data array)

```
In [11]:    input = h5py.File("output.h5", "r")
In [12]:    input.values()
Out[12]:    [<HDF5 group "/data" (1 members)>]
In [13]:    input['data']
Out[13]:    <HDF5 group "/data" (1 members)>
In [14]:    input['data'].values()
Out[14]:    [<HDF5 dataset "phi": shape (128, 128), type "<f8">]
In [15]:    phi = input['data']['phi']
In [16]:    phi.shape
Out[16]:    (128, 128)
```

Xdmf format can also describe the HDF5 data

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf xmlns:xi="http://www.w3.org/2003/XInclude" Version="2.2">
  <Domain>
    <Grid Name="Mesh" GridType="Uniform">
      <Topology TopologyType="3DCORECTMESH" Dimensions="1 128 128"/>
      <Geometry GeometryType="ORIGIN_DXDYDZ">
        <DataItem Name="Origin" NumberType="Float" Dimensions="3" Format="XML">0. 0. 0.</DataItem>
        <DataItem Name="Spacing" NumberType="Float" Dimensions="3" Format="XML">1. 1. 1.</DataItem>
      </Geometry>
      <Attribute Name="phi" Active="1" AttributeType="Scalar" Center="Node">
         <DataItem Dimensions="1 128 128" NumberType="Float" Precision="8“
Format=“HDF5">output.h5:/data/phi</DataItem>
      </Attribute>
    </Grid>
  </Domain>
</Xdmf>
```

Parallel I/O

Data formats and Parallelism

- MPI-IO
  - Raw data parallelism
  - Some can be read by VisIt (BOV format)
- ADIOS
  - Raw data but complexity is hidden
- HDF5, NetCDF
  - content-discovery is possible, but semantic is not given
- SILO
  - Poor man’s parallelism (1 file per process + metafile) but strong semantic

SILO

- https://wci.llnl.gov/simulation/computer-codes/silo

- A very versatile data format. The "Getting Data Into VisIt" manual covers how to create files of this type. In addition, there are many code examples here

- http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/CreatingCompatible

SILO

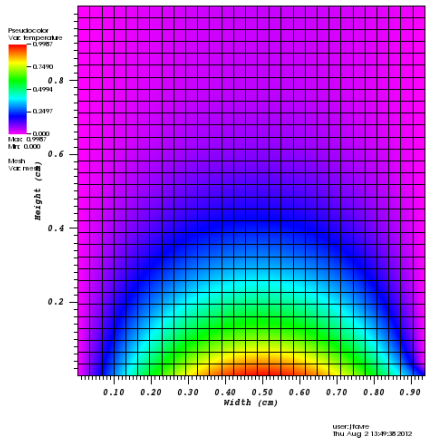
From the User Manual:

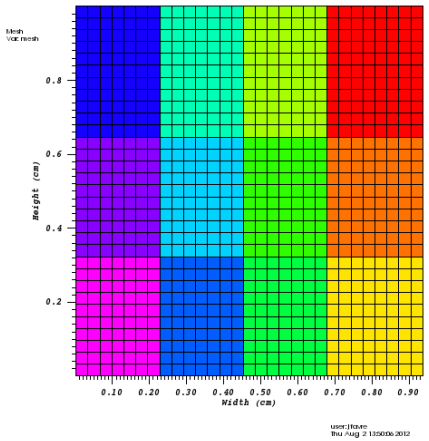
- Silo is a serial library. Nevertheless, it (as well as the tools that use it like VisIt) has several features that enable its effective use in parallel with excellent scaling behavior.

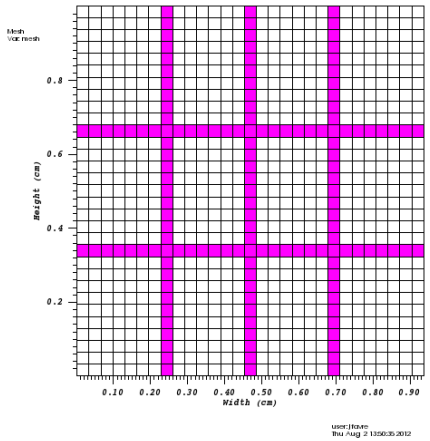
MPI tasks, ghost-cells, hyperslabs

- Grids are sub-divided with ghost regions
- Ghost cells/nodes are usually not archived
- The User (You) is responsible for managing the subdivisions and know what to archive

Example: a 12-processor run

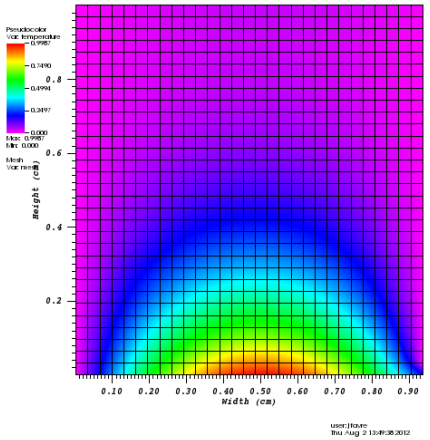


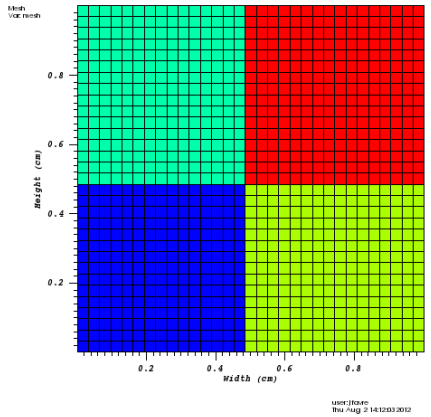


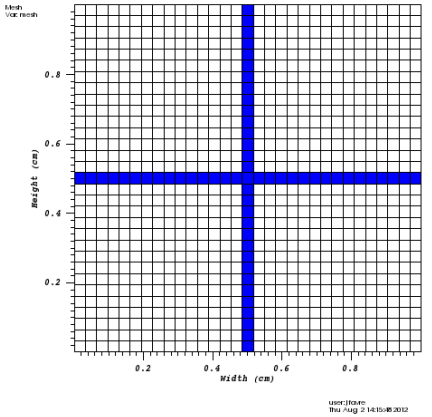


MPI tasks, ghost-cells, hyperslabs

Example: a 4-processor run

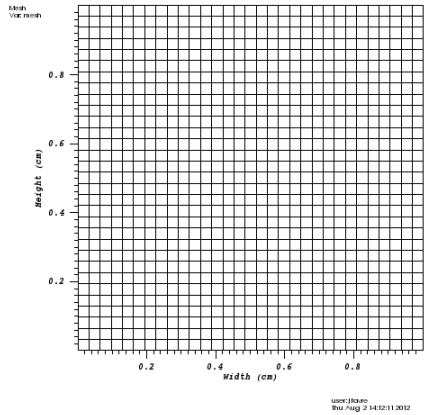






MPI tasks, ghost-cells, hyperslabs

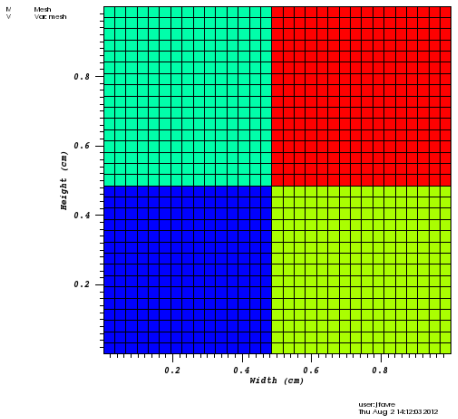
The goal of (parallel) I/O:
Present a uniform grid storage/display



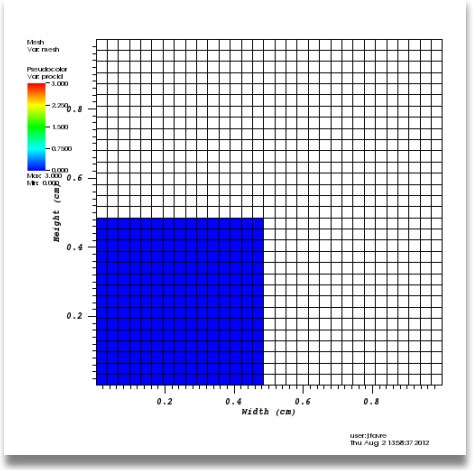
The Visualization software (ParaView or VisIt) will do its own subdivision and re-construct ghost-zones – when necessary

MPI tasks, ghost-cells, hyperslabs

Def: a hyperslab, is a subset in n-D of a larger grid. Parallel I/O is a composition (superposition) of multiple hyperslabs.



Each processor must know where each piece fits in the global mesh



Data formats. Parallelism

- Once you know the IJK extents of all your hyperslabs, you can use
  - MPI-IO, or
  - HDF5, or
  - NetCDF, or
  - ADIOS

MPI-I/O. Each process shares a collective view of the global domain (1)

```
MPI_Offset       disp   = 0;
MPI_File         filehandle;
MPI_Datatype     filetype;

int result = MPI_File_open(
        MPI_COMM_WORLD,
        (char*)filename,
        MPI_MODE_CREATE | MPI_MODE_WRONLY,
        MPI_INFO_NULL,
        &filehandle);
MPI_File_set_size(filehandle, disp);  // to truncate the file on open
```

MPI-I/O. Each process shares a collective view of the global domain (2)

```
int ustart[]  = {domain.startx-1, domain.starty-1};  // starting coordinates of subarray
int ucount[]  = {domain.nx, domain.ny};   // number of elements of the subarray
int dimuids[] = {options.nx, options.ny};   // number of elements of the full array
int ndims = 2;                                    // number of array dimensions
MPI_Type_create_subarray(ndims , dimuids, ucount, ustart,
MPI_ORDER_FORTRAN, MPI_DOUBLE, &filetype);


MPI_Type_commit(&filetype);
```

MPI-I/O. Each process shares a collective view of the global domain (3)

```
MPI_File_set_view(filehandle, disp, MPI_DOUBLE, filetype, "native",
MPI_INFO_NULL);

// collective write using individual file pointer
MPI_File_write_all(filehandle, u.data(), domain.N, MPI_DOUBLE,
MPI_STATUS_IGNORE);

MPI_Type_free(&filetype);

MPI_File_close(&filehandle);
```

Summary 1

- Documenting how data files are generated, and what is the intended purpose of the raw data is of utmost importance.
- Think long-term. Think data sharing. Think multiple post-processing tools
- Raw binary data is fine. As long as it is augmented with some meta-data headers such as BOV, or Xdmf, or numpy reading commands.

Summary 2

- Higher level libraries such as HDF5 enable parallel I/O but most importantly, provide self-documentation about the nature of the raw data
- Yet, interpretation is application-dependent

- There exist ‘conventions of usage’ of HDF5, or netcdf.
  - Pixie,
  - H5Part,
  - “CF”
- HDF5 and Xdmf are often used together.

Summary 3

- There exists many formats used by each communities
  - Molecular science
  - Fluid dynamics
  - Astronomy, …
- Your first thought should be to see if you can re-use such formats.
- Warning. Just because you have a visualization software which runs in parallel does not guarantee that you can read data in parallel…

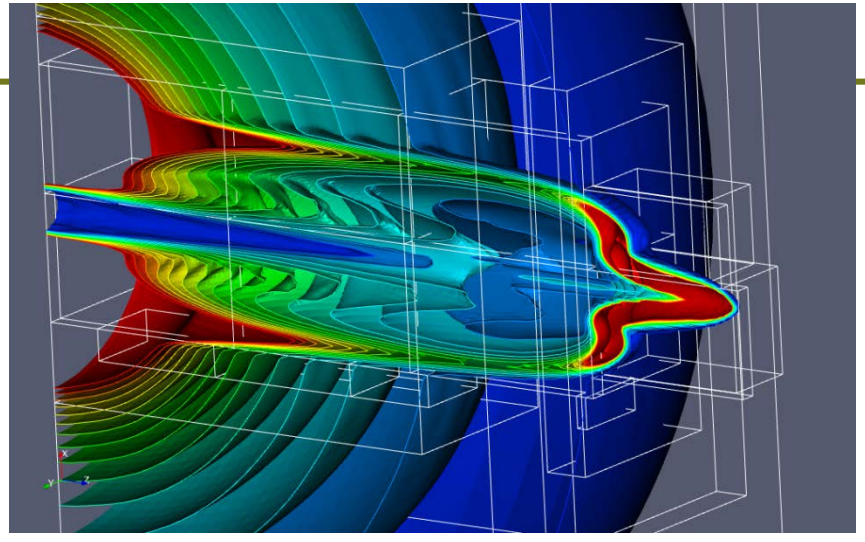
Scientific Visualization

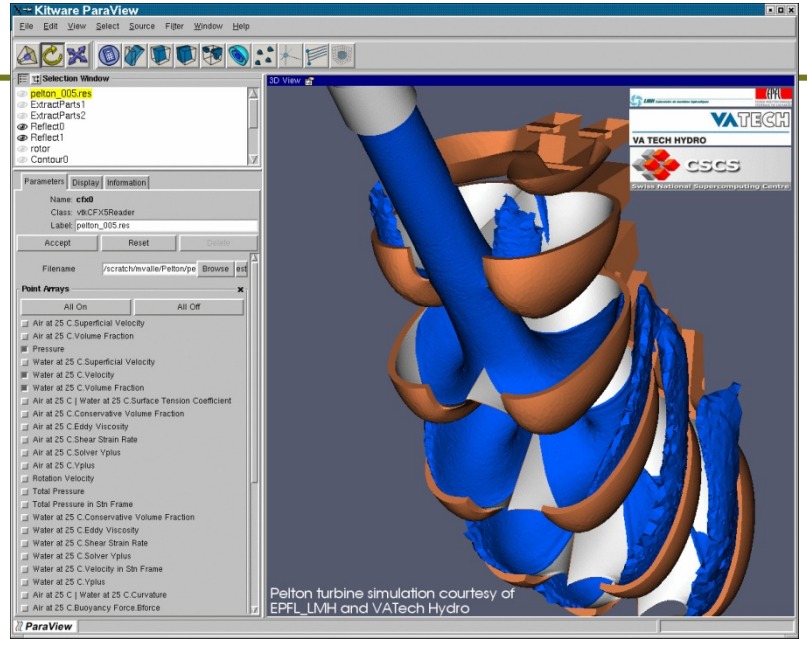
- Why visualization?
  - **How to:**
    - **Remote Visualization**
    - **Client server**
    - **Parallel Visualization**
    - **In-situ Visualization**

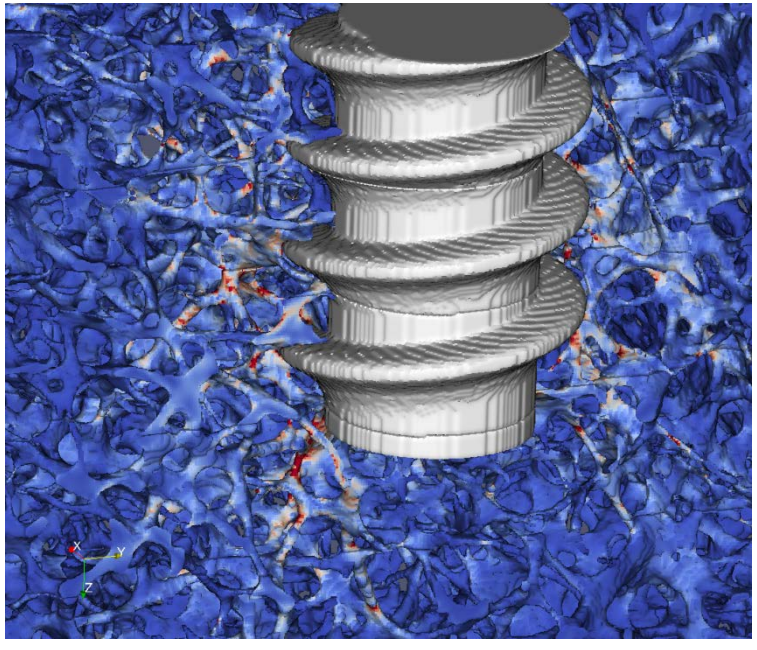
Scientific Visualization

- Debug your code
  - **Demonstrate your simulation does the right thing**
  - **Analyse the results**
  - **“A picture is worth a 1000 words”**

Corollary:
- **“A bad picture is worth nothing**







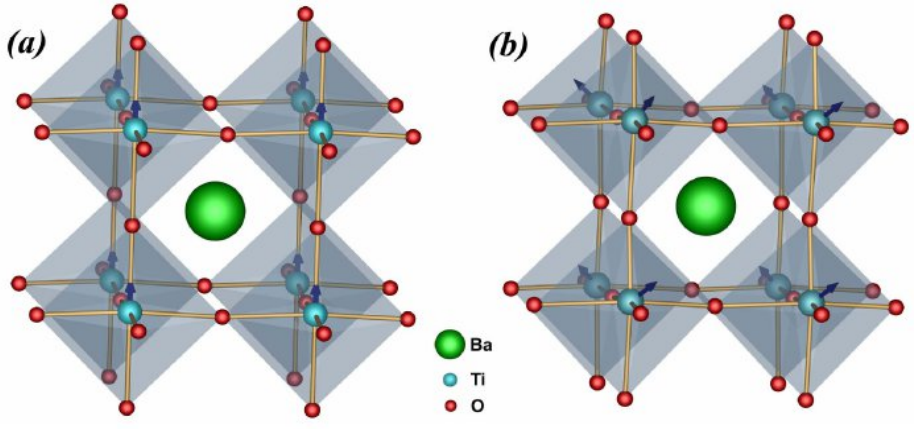
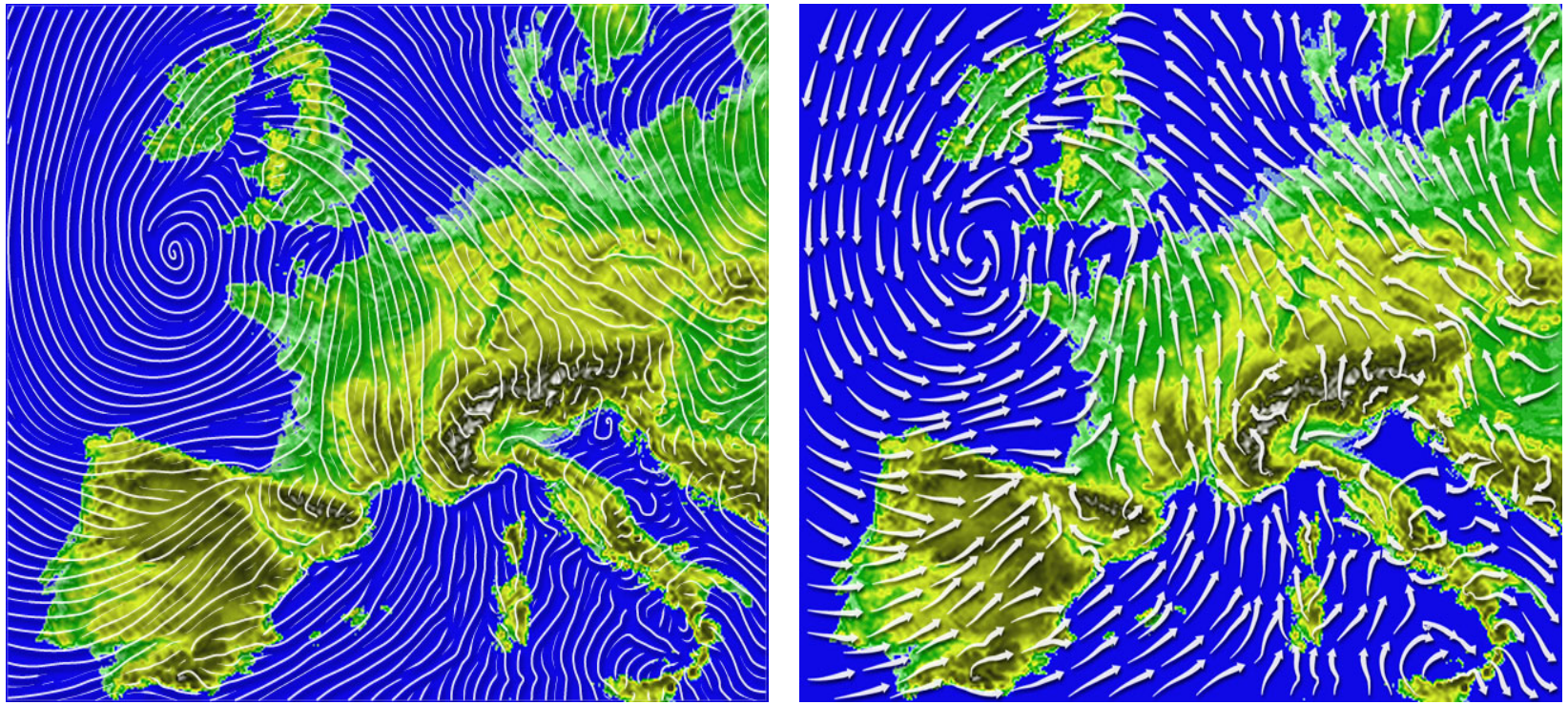
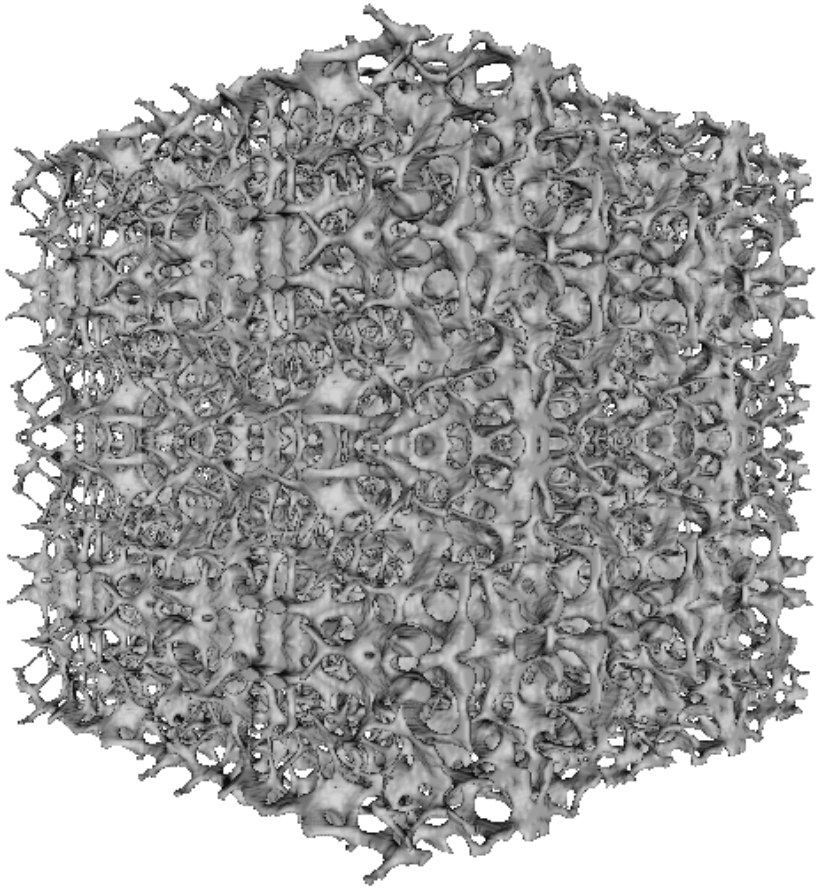


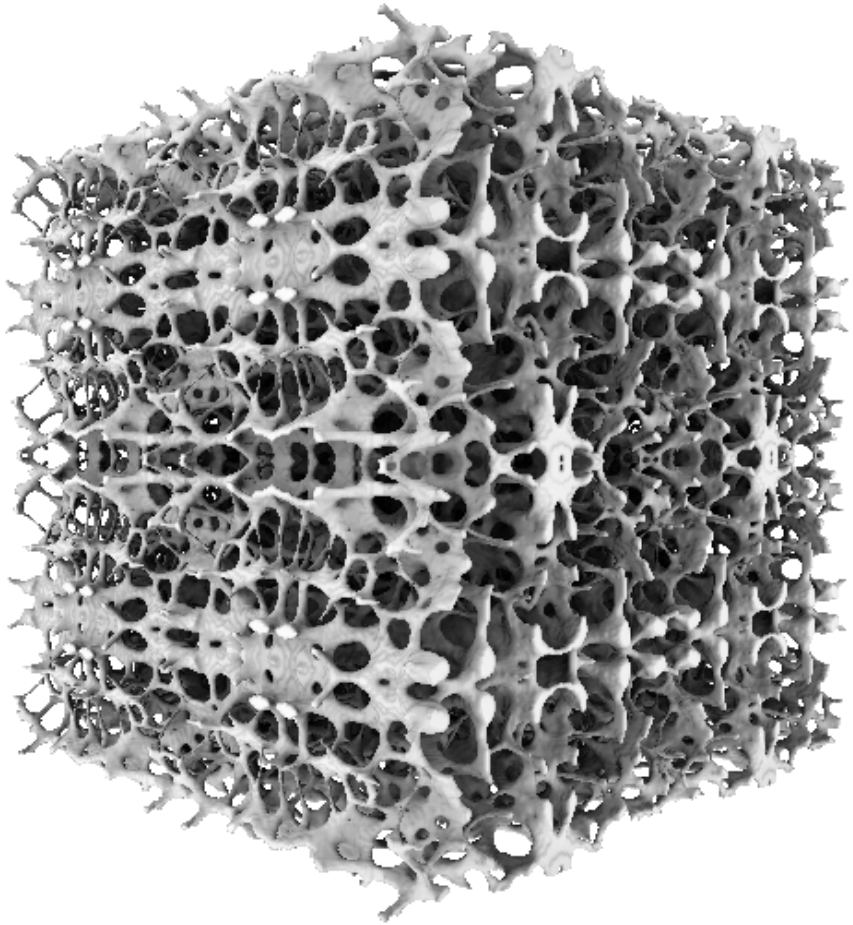
Figure 2. Displacement pattern of Ti atoms in tetragonal $BaTiO_3$. (a) Average picture with Ti shiftings along the direction of tetragonal distortion. (b) Local, instantaneous Ti shifts along directions close to the body diagonal. Ti displacements are represented as bold blue vectors.

different visual representations

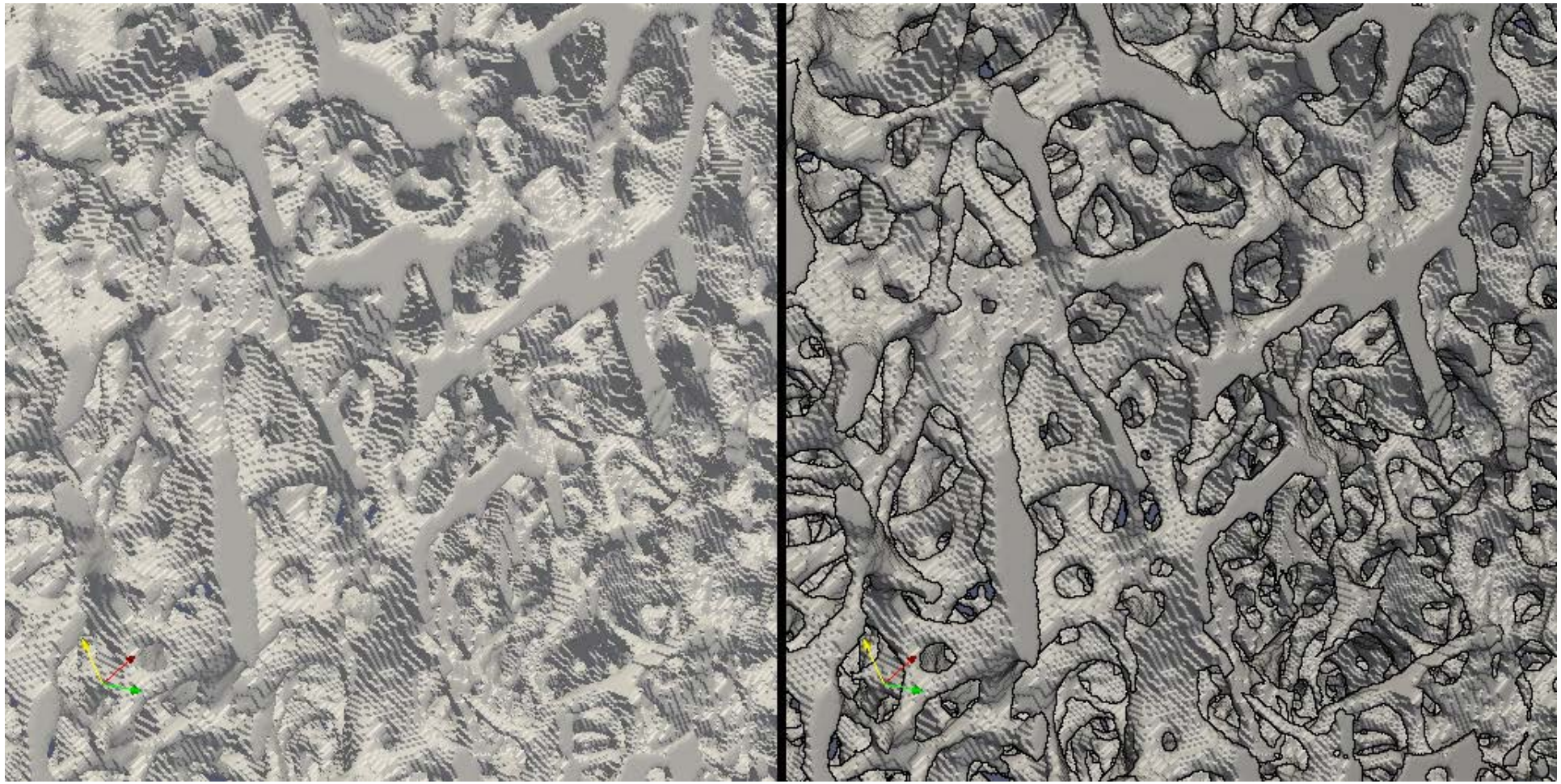


Increased realism with Ambient Light Occlusion

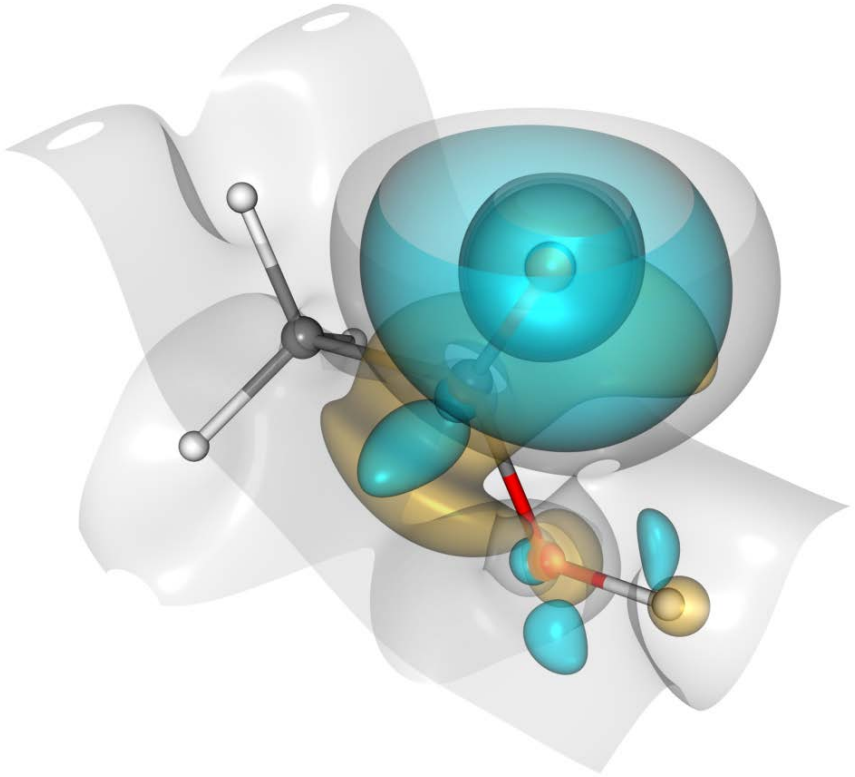




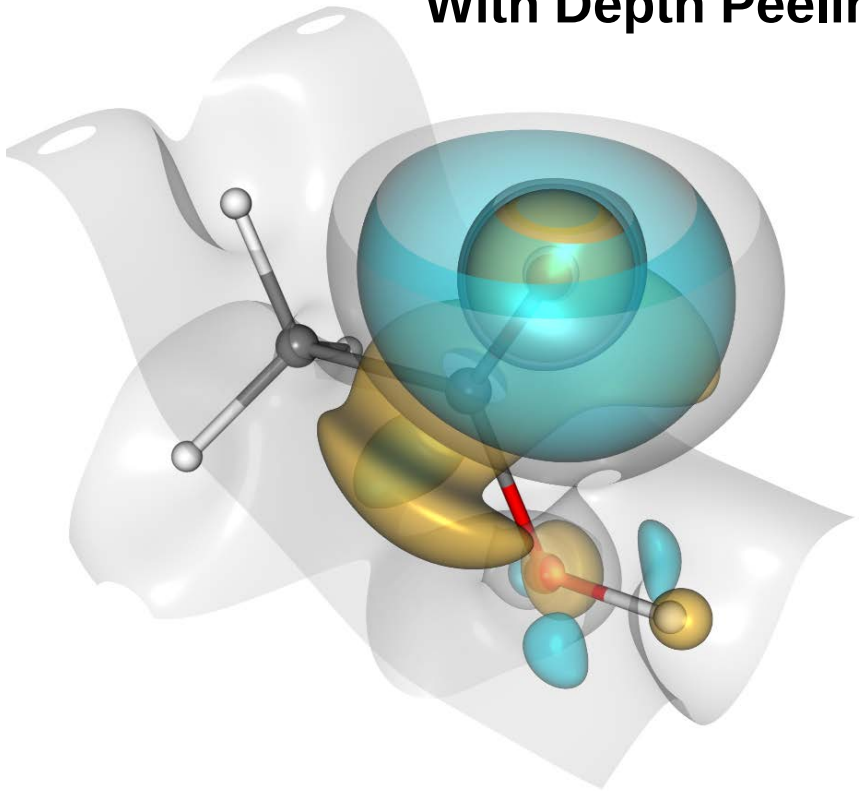
Increased realism with silhouettes



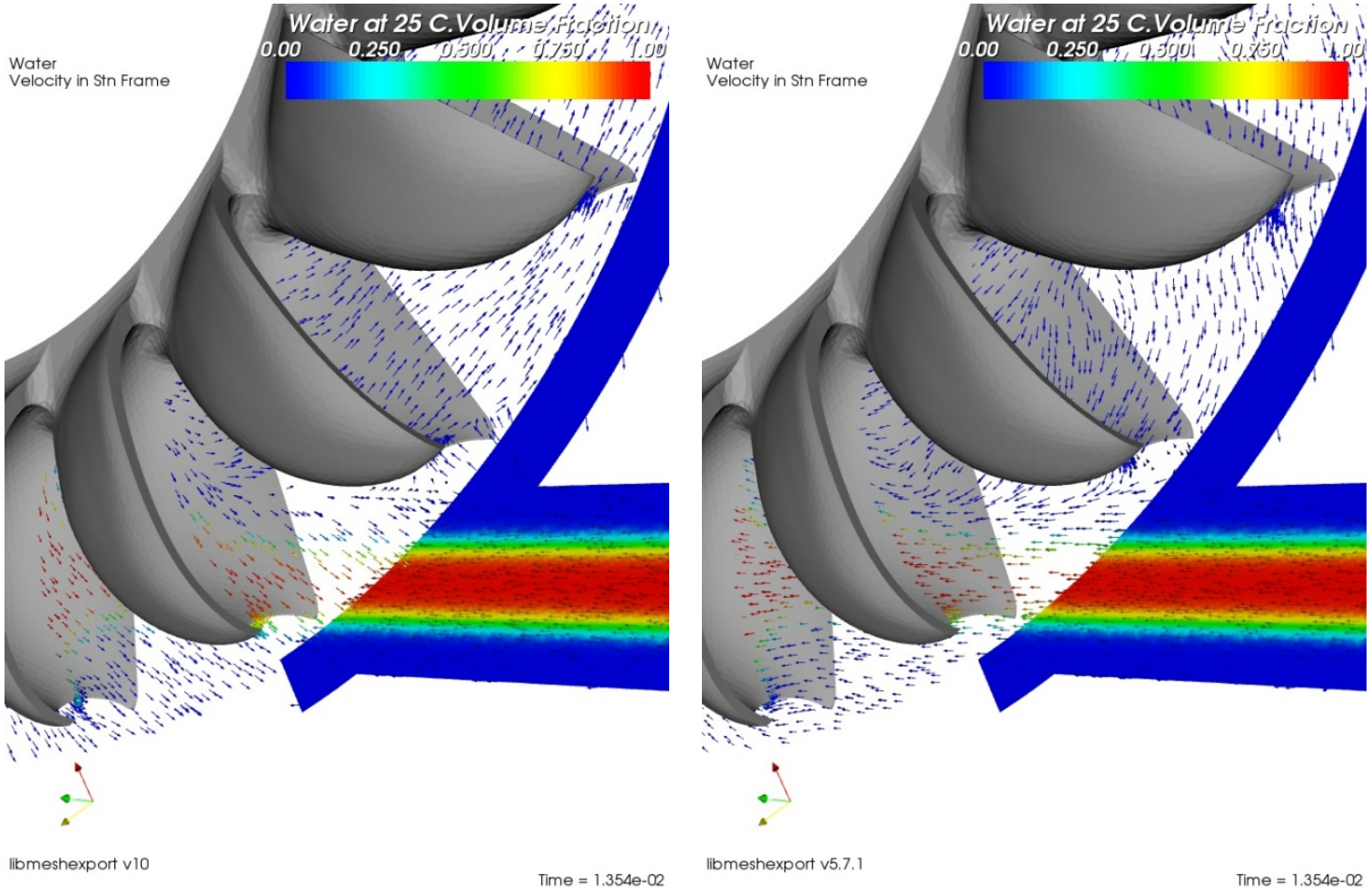
Explain and correct errors



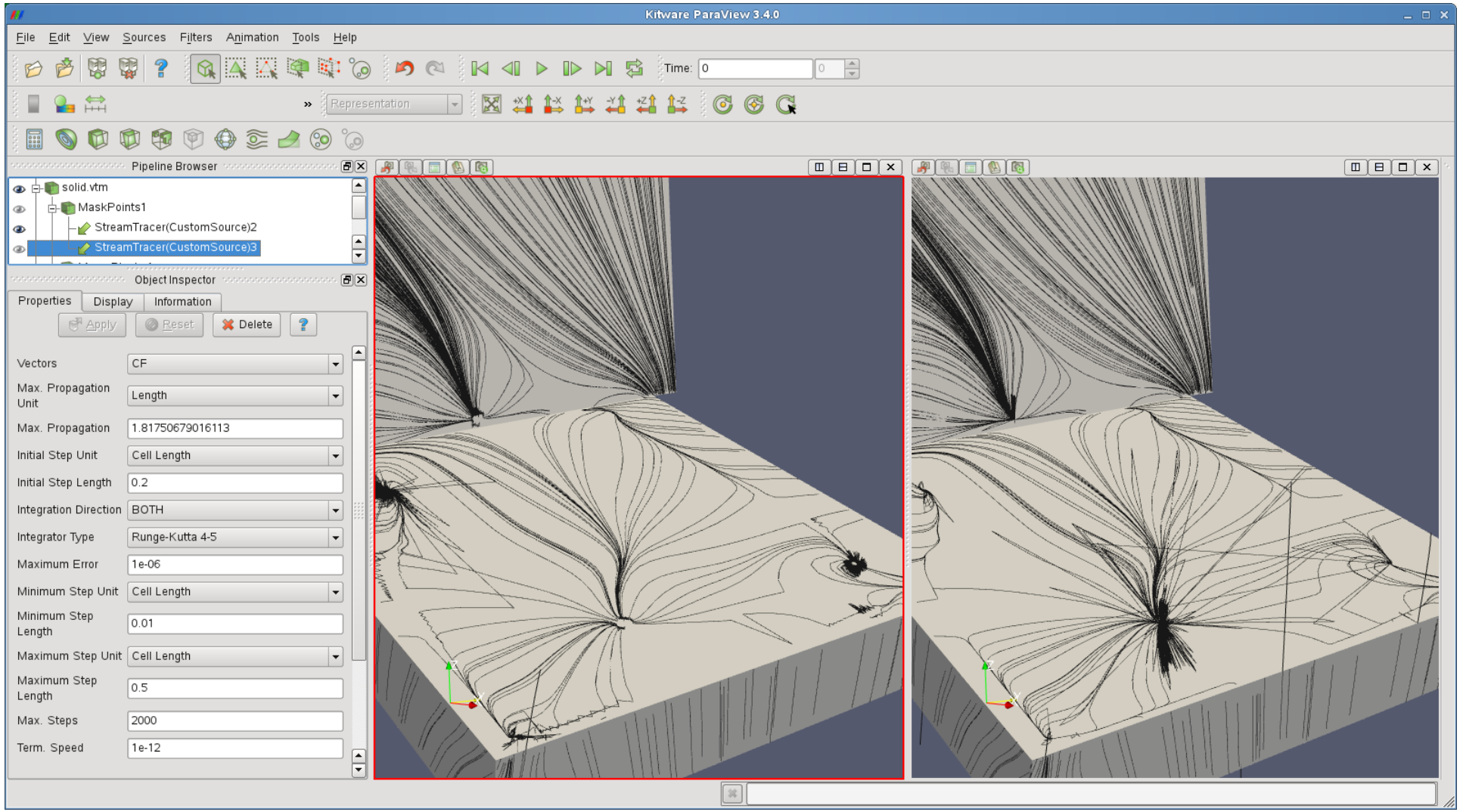
With Depth Peeling



Understand and debug vis. softwares



Correct and fine-tune parameters in numerical analysis



Scientific Visualization: Two modes

- **Post-processing**

This does not mean you can start thinking about it [The Visualization] just after the simulations are done. You should plan it before running your code…

- **Live, a.k.a. in-situ visualization**

Simulation and visualization codes run at the same time, on a shared resource, or a distributed set of machines. An advanced topic, … (see demo)

Scientific Visualization

We advocate two modes of execution:

1. **Interactive imaging, analysis, query…**

Requires a GUI, to test and try multiple visual representations.

2. **A batch-oriented movie-making process**

Requires a script (python), to enable reproducible visualizations, and the support of time series, or multiple experiments

Scientific Visualization Hardware

“Piz Daint” is the only production machine at CSCS with graphics hardware (i.e. hardware accelerated OpenGL) for remote, screen-less, server-side visualization.

It can also offer remote displays (VNC desktops)

From your desktop, parallel visualization jobs can be launched on CSCS production machines (daint, dora, pilatus) where VisIt and ParaView are installed.

Scientific Visualization Software

Two open-source softwares for Large Scale Visualization are available at CSCS

- ParaView
- VisIt

Other graphics applications available on desktops can be run on the VNC remote desktops of daint (example, VMD)

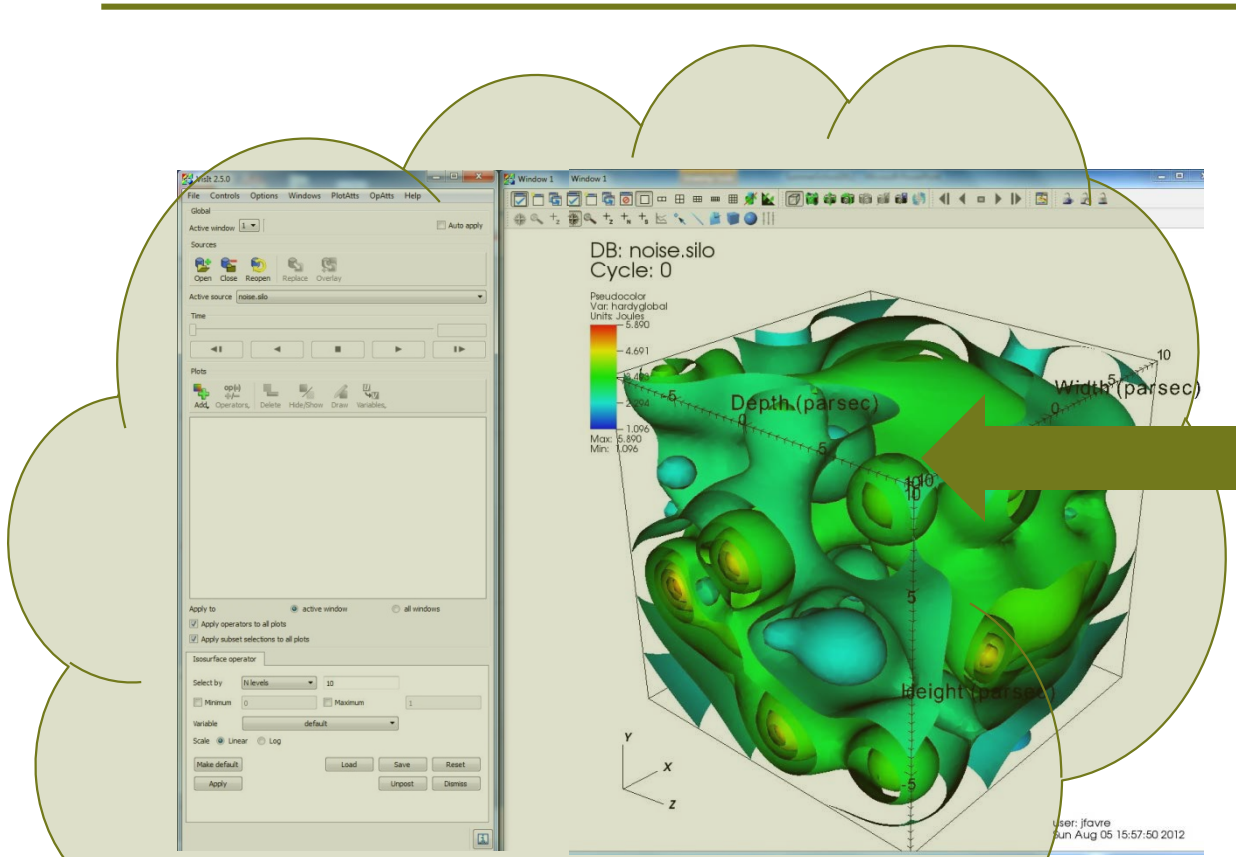
Visualization: Client Server

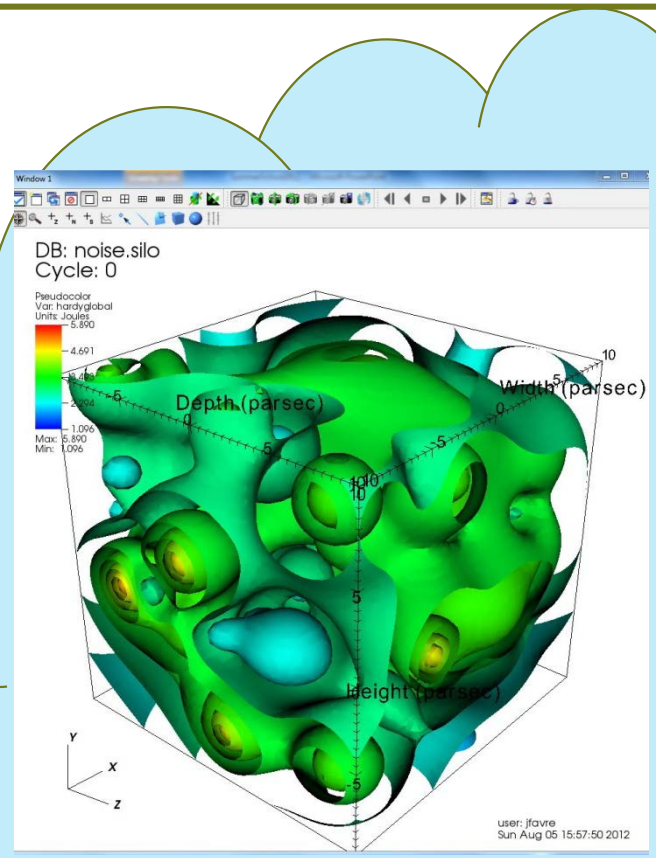
ParaView and VisIt use the client-server concept:

- A client (optional) runs the GUI
- A server, embedded and local (by default), or remote and/or parallel, does the real work:

  - I/O
  - Data analysis
  - Image generation

Remote Visualization (Image generation + copy)





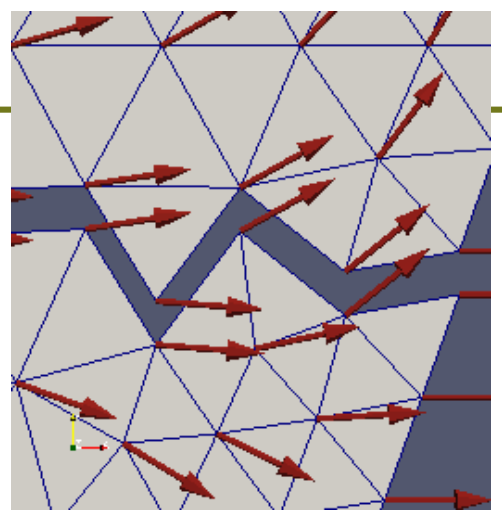
@home
@office

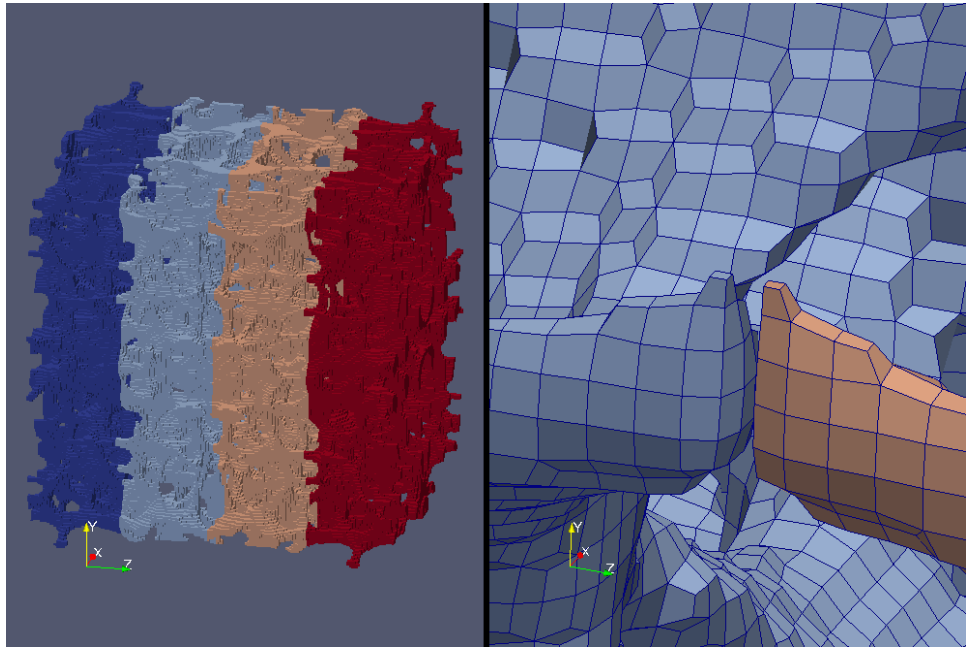
@CSCS

Visualization: Parallelism

Parallelism is a must for big data.

Parallelism is the source of many problems.





Visualization: Parallelism

Should we bother?

Yes!

Interactive visualization is necessary to gain insight from exploration.

Yes!

Parameter tuning should be fast.

VisIt-ing some of the topics of the VisIt tutorial (online)

- Let’s keep it simpler, and practice visualization on the desktop.

- Install VisIt
- There should be a directory called “data” with example files used by the tutorials

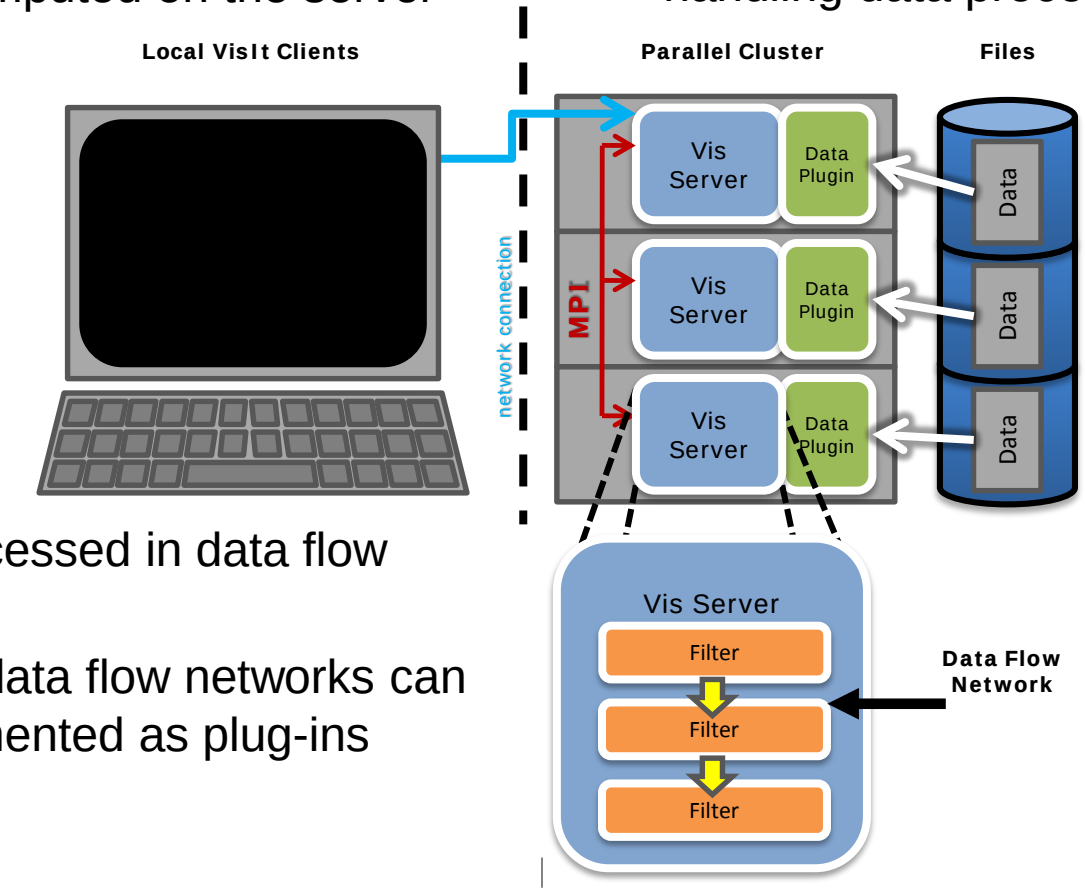
- Files are identical: “noise.silo” **is** “example.silo”
- Variable “Temp” **is** “hardy_global”

VisIt-ing some of the topics of the VisIt tutorial (online)

- VisIt Basics
- Data Analysis
- Scripting
- Data level comparisons

In-situ Visualization

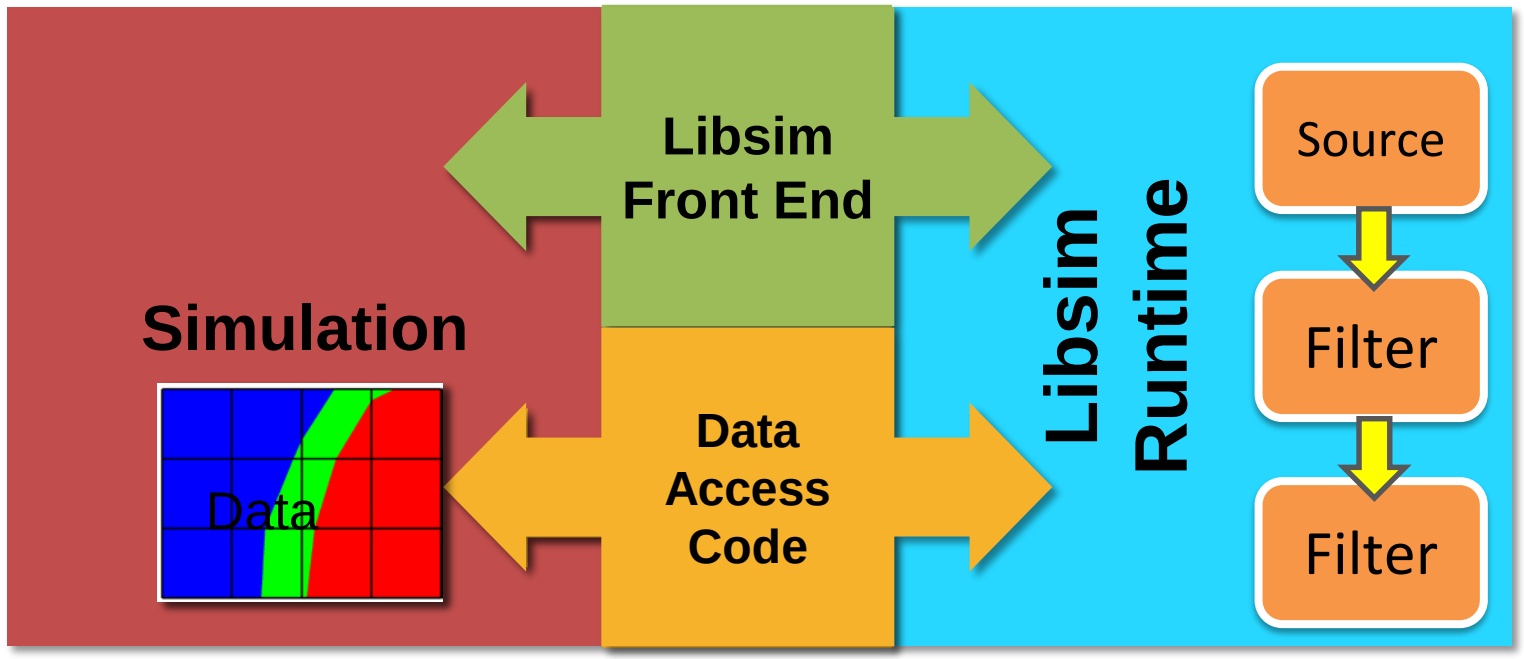
- Clients runs locally and display results computed on the server
- Server runs remotely in parallel, handling data processing for client



- Data processed in data flow networks
- Filters in data flow networks can be implemented as plug-ins

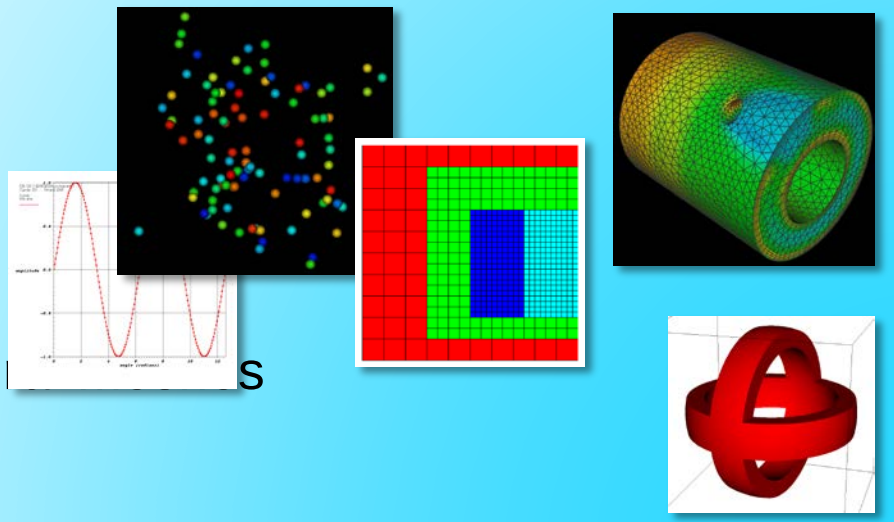
Coupling of Simulations and VisIt

Libsim is a VisIt library that simulations use to enable couplings between simulations and VisIt. Not a special package. It is part of VisIt.

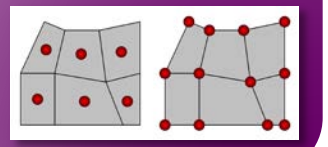


Data model for in-situ visualization

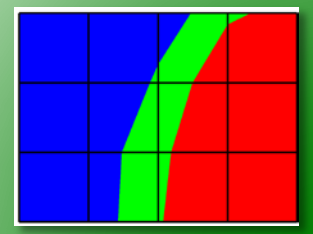
- Mesh Types
  - Structured meshes
  - Point meshes
  - CSG meshes
  - AMR meshes
  - Unstructured & Polyhed s



- Variables
  - 1 to N components
  - Zonal and Nodal



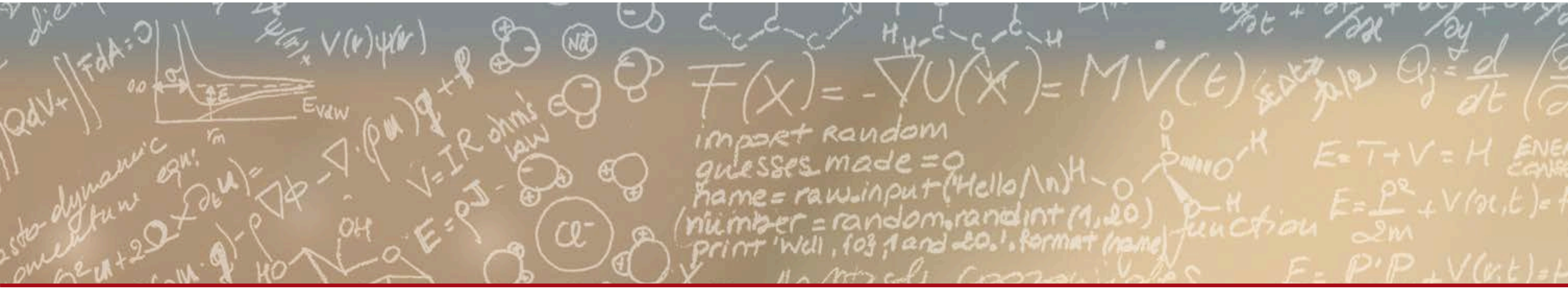
- Materials
- Species



exercises

- Visualize the output of the miniApp
- Create a time-dependent output. 1 file per timestep.
- Evaluate the 2D gradient of phi, the magnitude of gradient
- Find erroneous data. Where? i,j coordinates? rank ? value? neighbors? Use spreadsheet
- Plot node's value over time?
- Find mean, at current time, a mean-over-time?
- Compare serial output with:
  - OpenMP output ?
  - MPI output ?
  - CUDA output ?

- Compare solution at time T1, with solution at time T2?



Thank you for your attention.