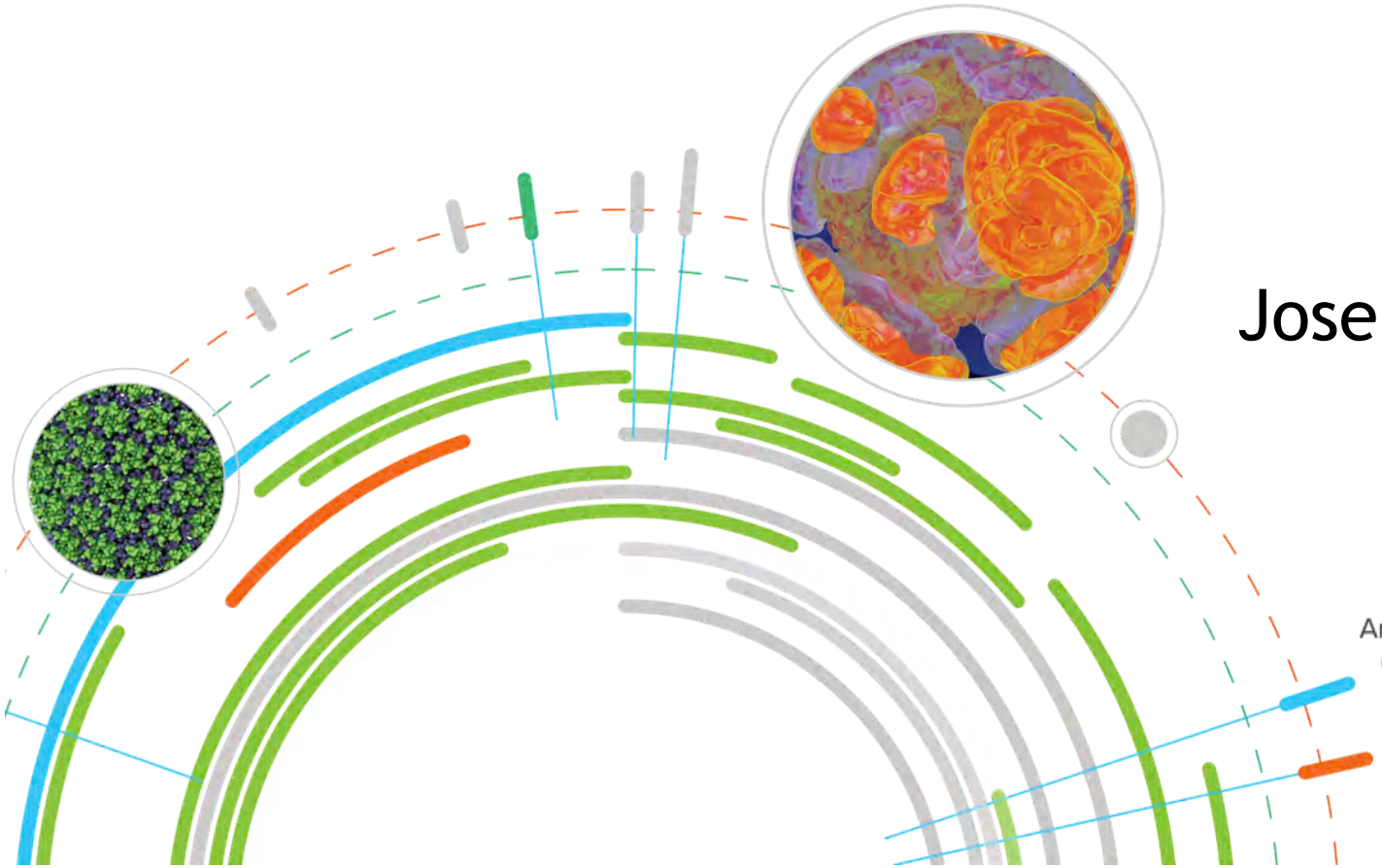
Your Data Analysis Needs and Tukey



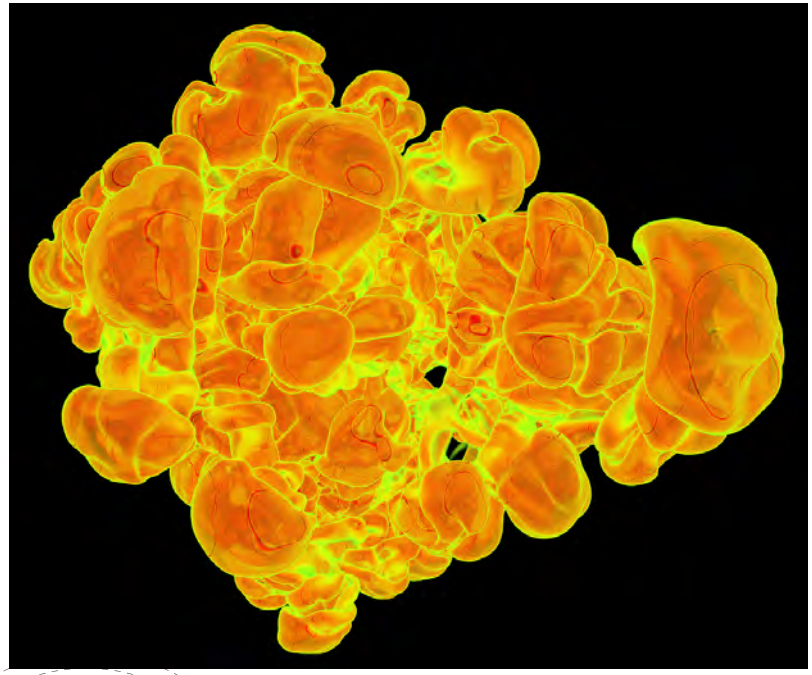
Joseph A. Insley

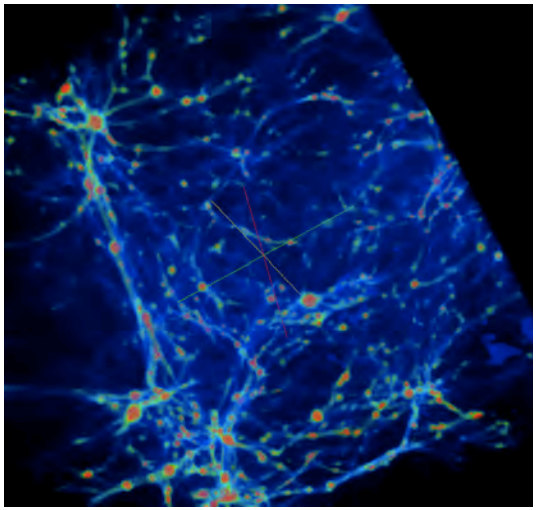
Argonne Leadership Computing Facility

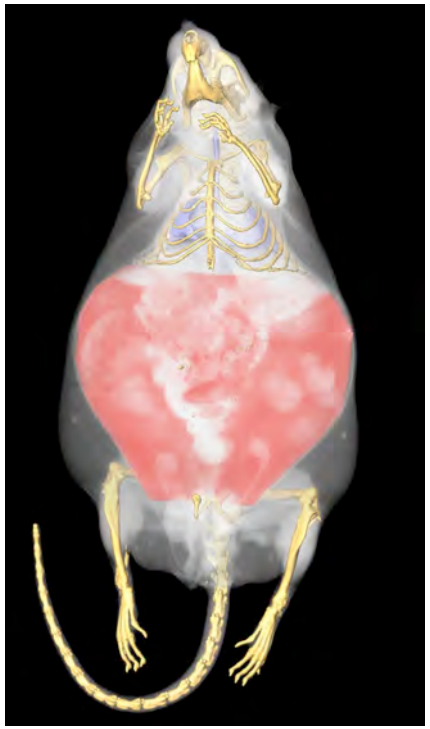
Argonne NATIONAL LABORATORY

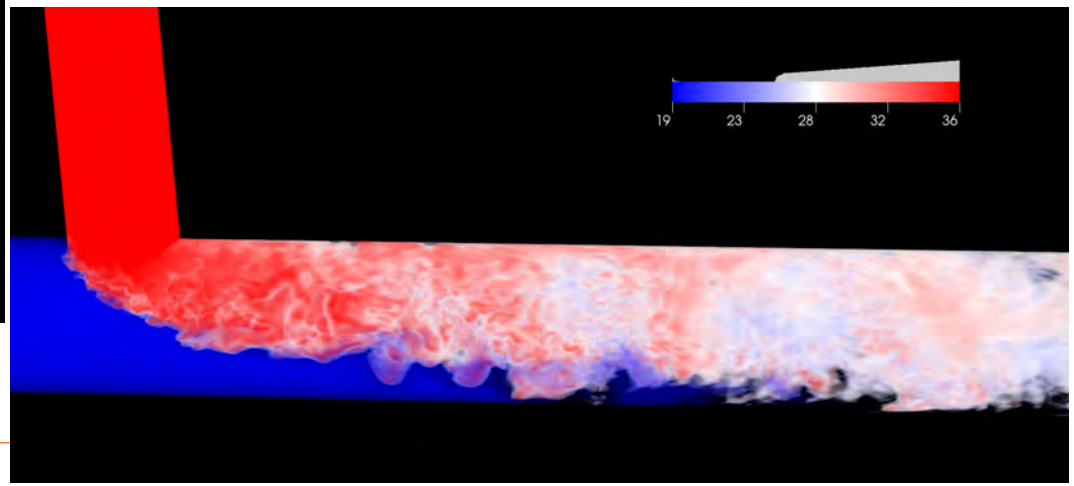
Data Representations: Volume Rendering

- Turn 2- and 3-dimensionsal datasets into 2D images
- Approximation: Volume ray casting



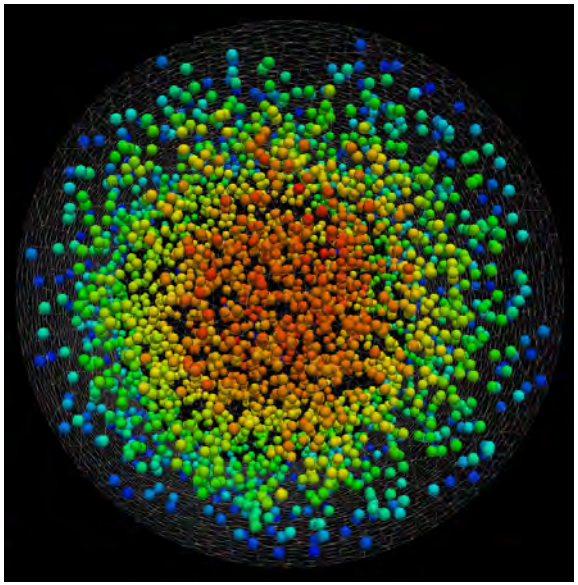


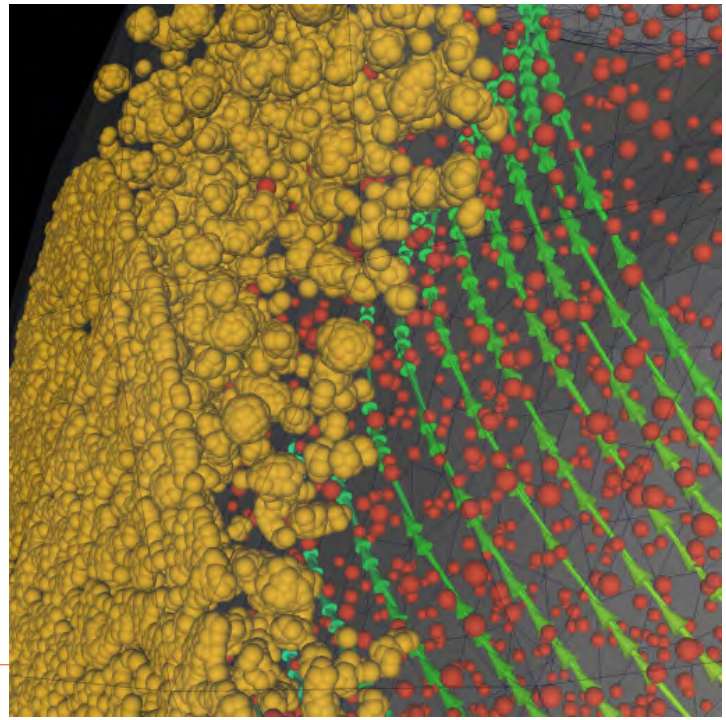


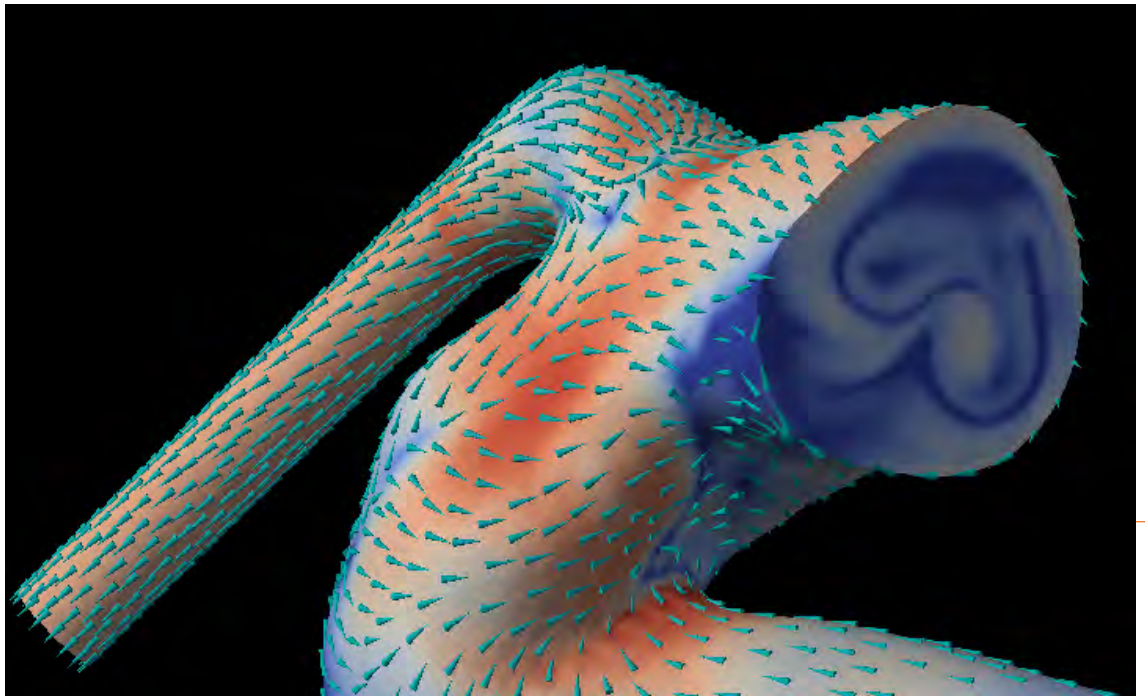


Data Representations: Glyphs

- 2D or 3D geometric object to represent point data
- Location dictated by coordinate
  - 3D location on mesh
  - 2D position in table/graph
- Attributes of graphical entity dictated by attributes of a data
  - color, size, orientation

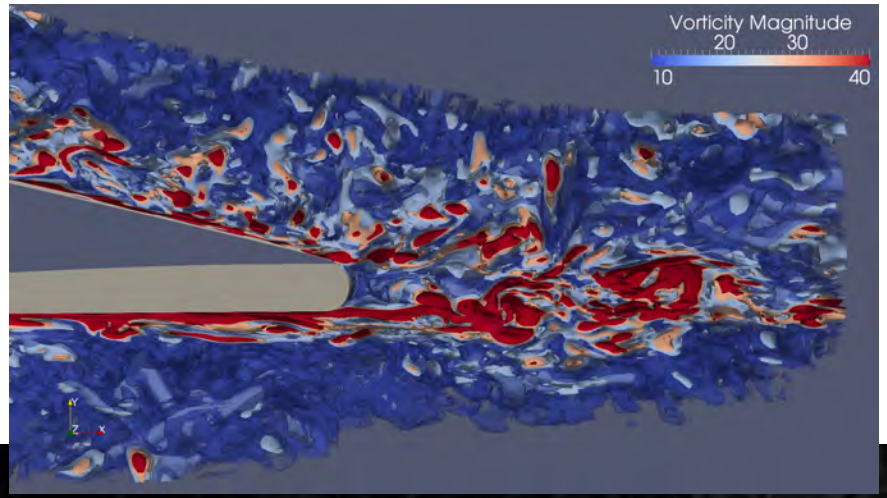


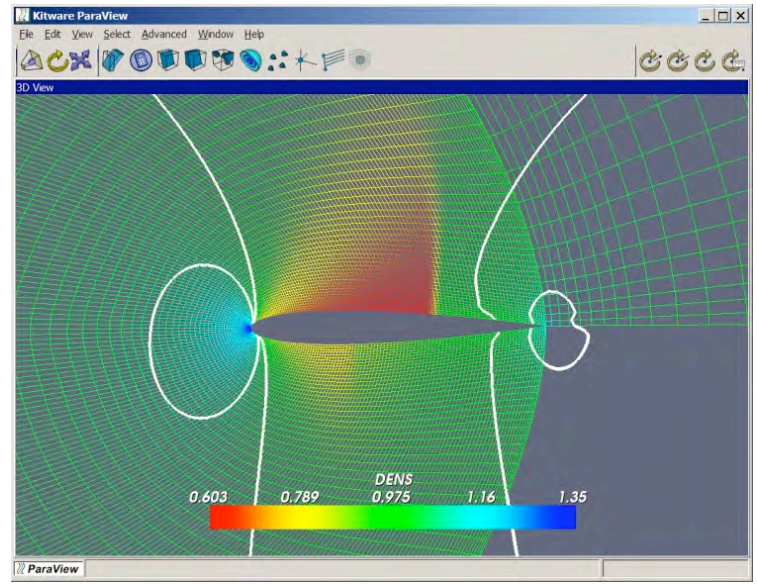


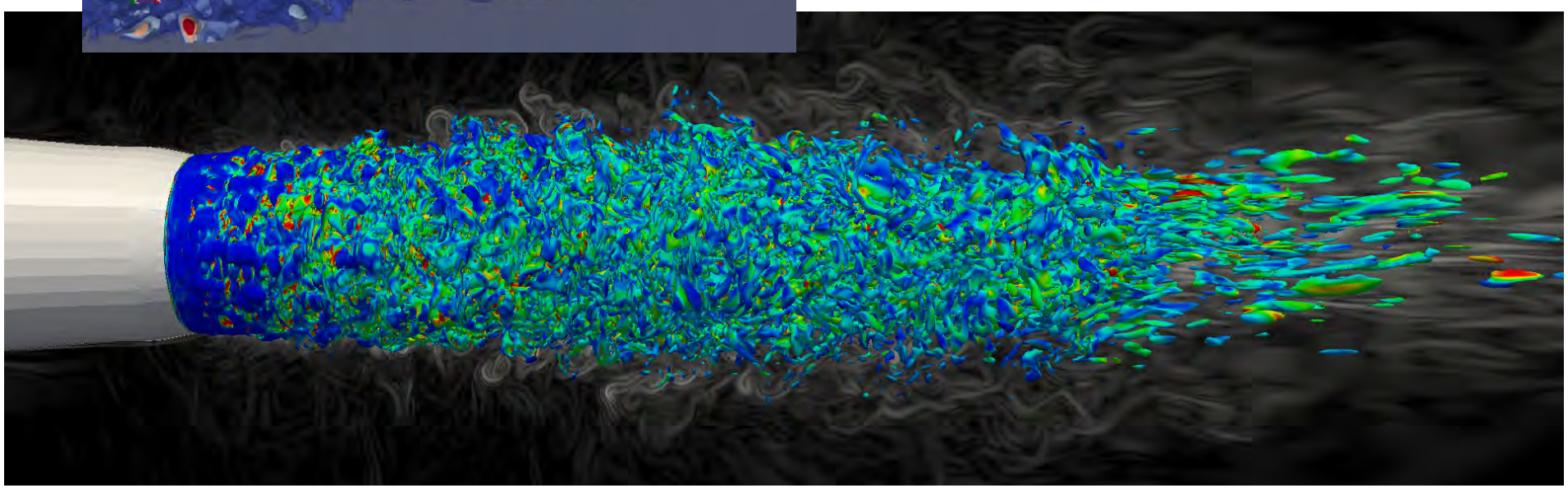


Data Representations: Iso-contours

- A Line (2D) or Surface (3D), representing a constant value

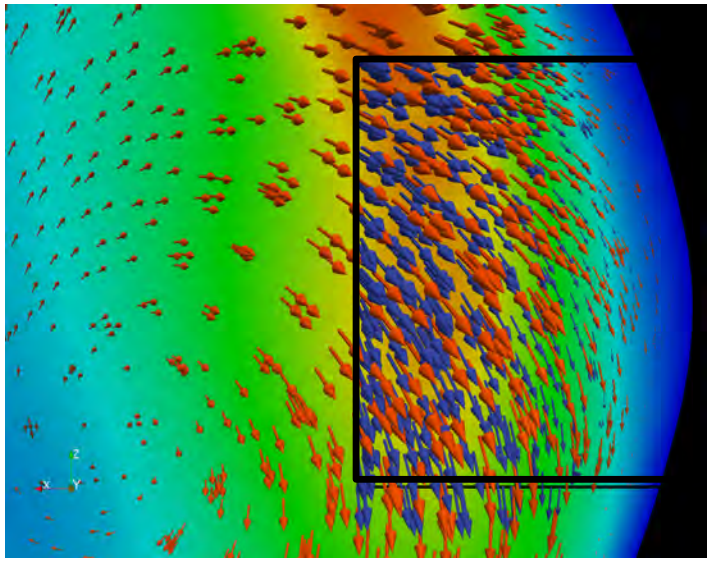


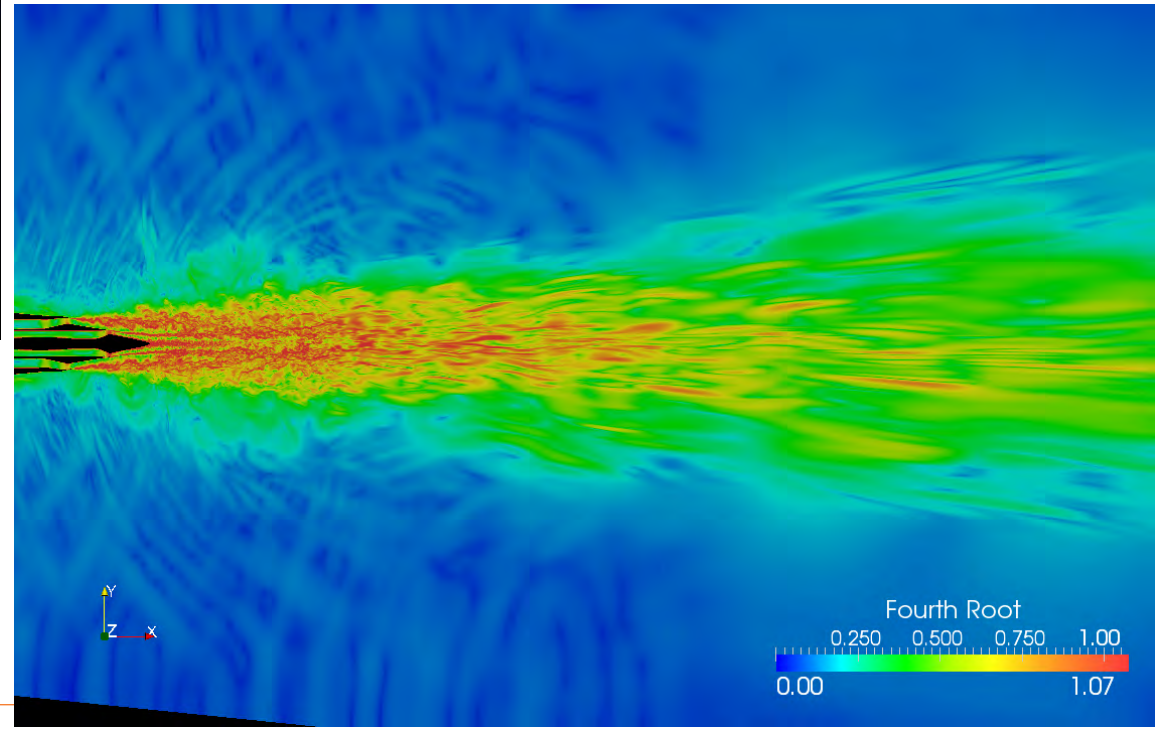




Data Representations: Cutting Planes

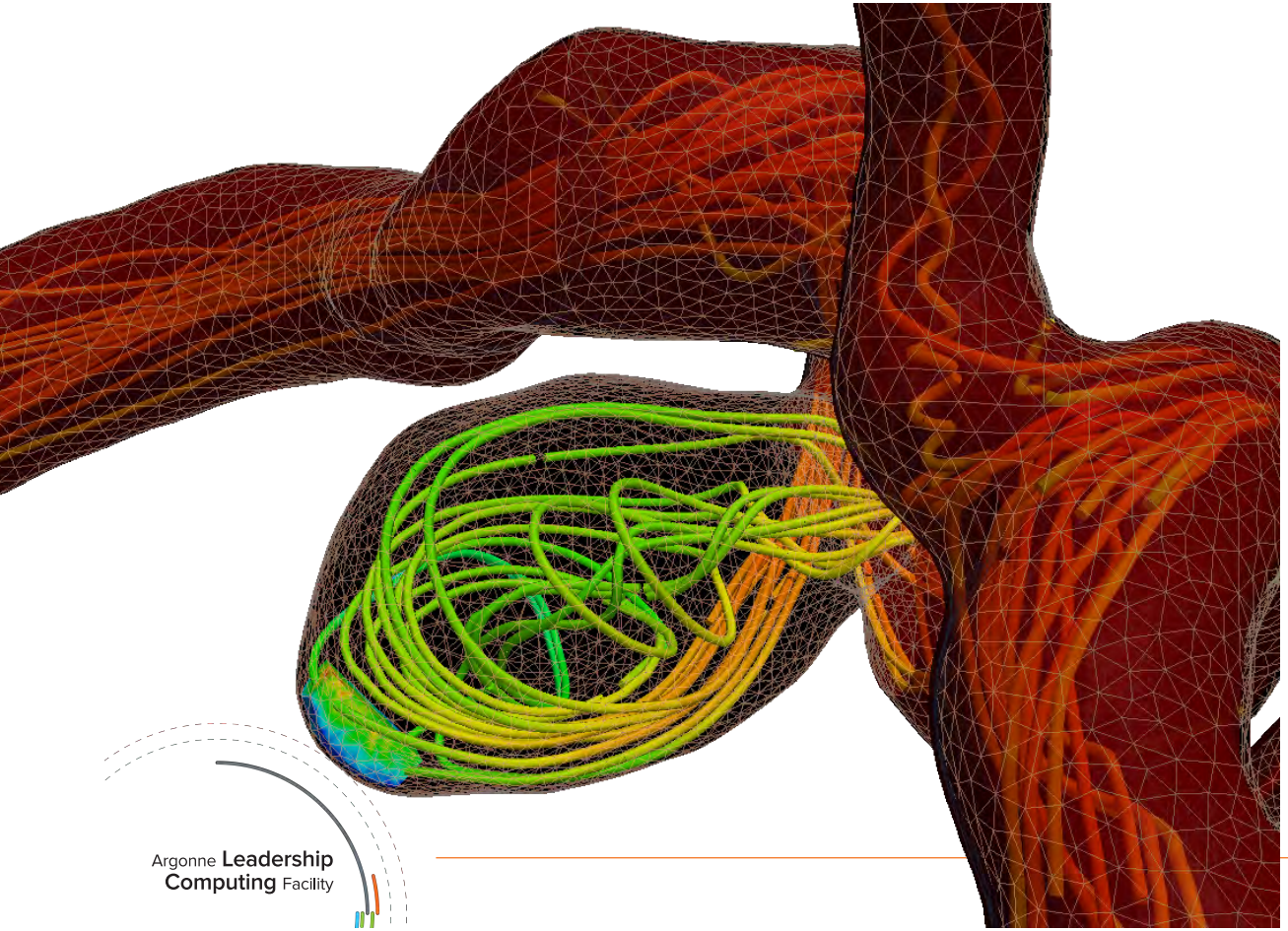
- Slice a plane through the data
  - Can apply additional visualization methods to resulting plane

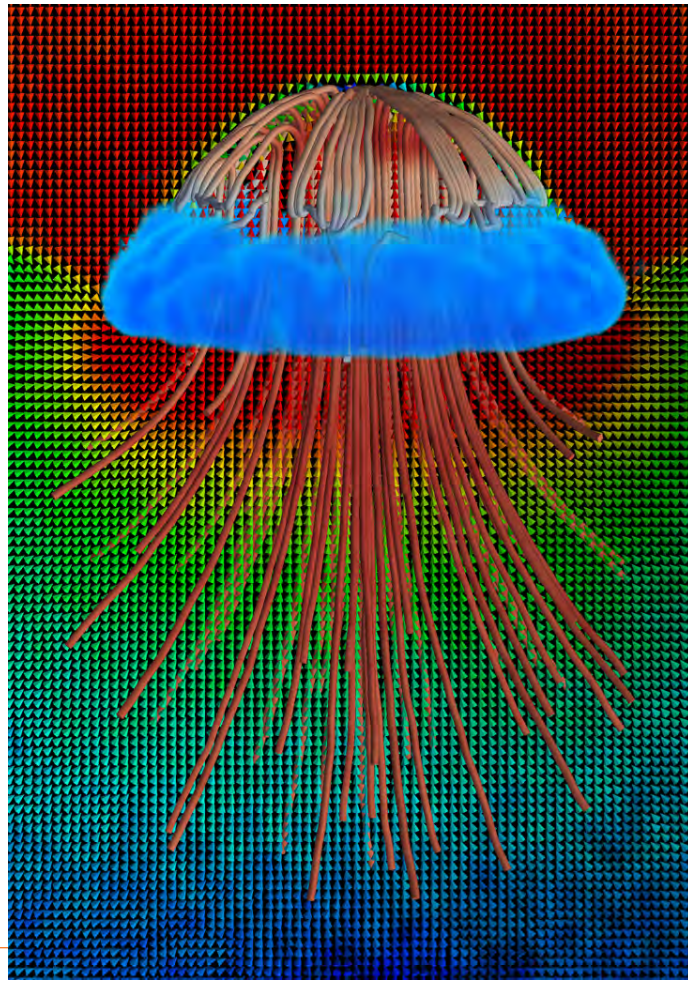




Data Representations: Streamlines

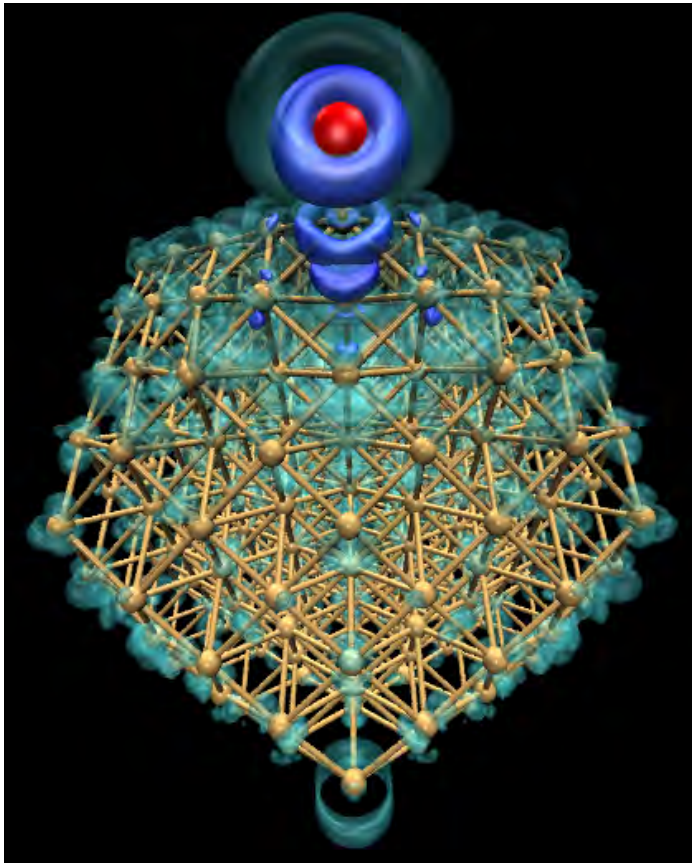
- From vector field on a mesh (needs connectivity)
  - Show the direction an element will travel in at any point in time.

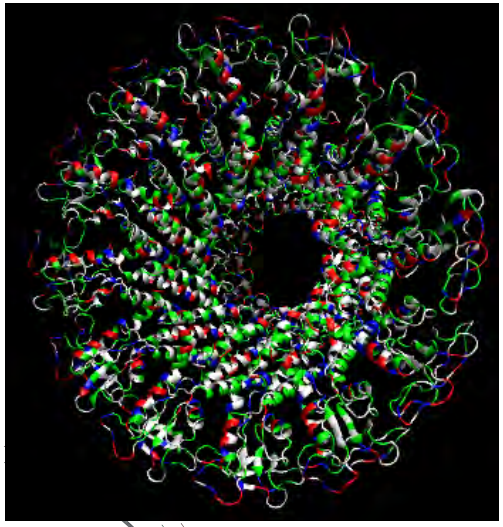


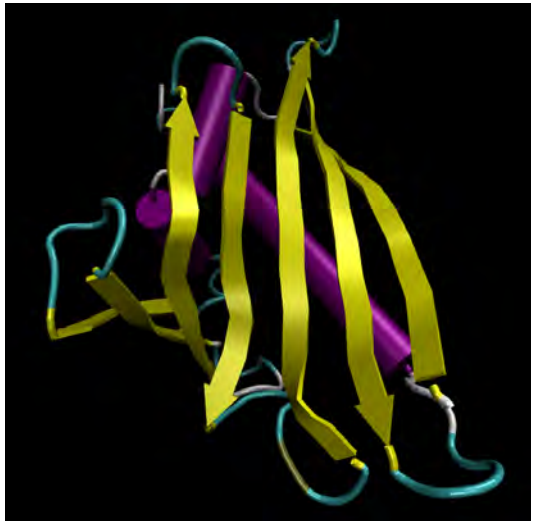


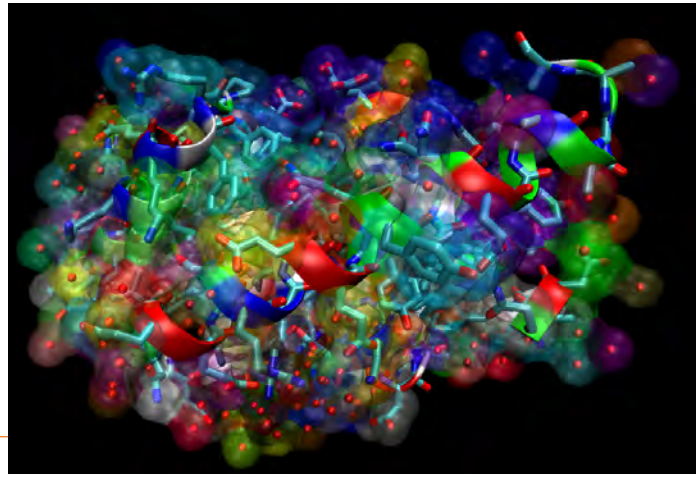
Molecular Dynamics Visualization

- Domain-specific representations
  - Backbone
  - Display attributes by atom type
  - Ball and stick
  - Ribbon representation
  - etc.



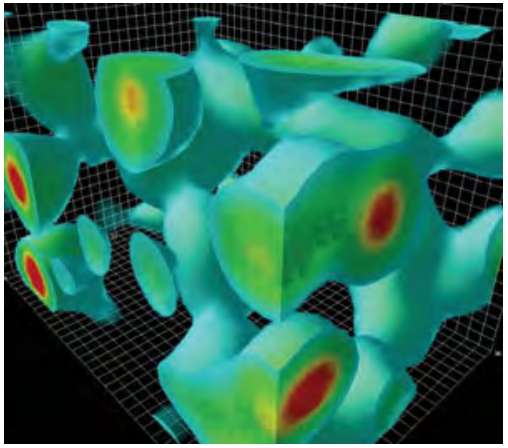


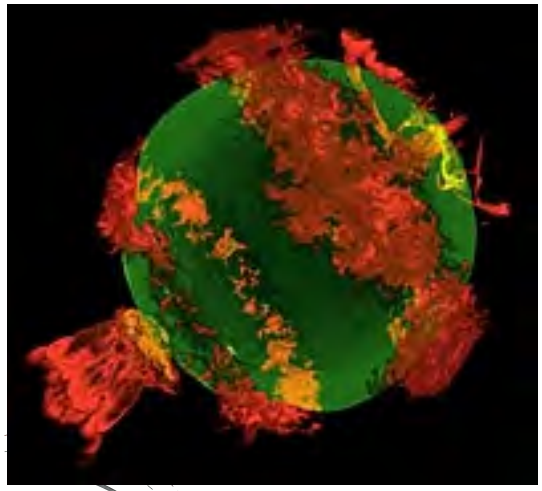


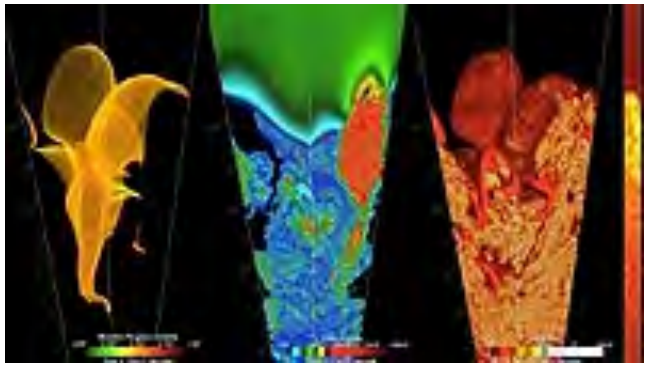


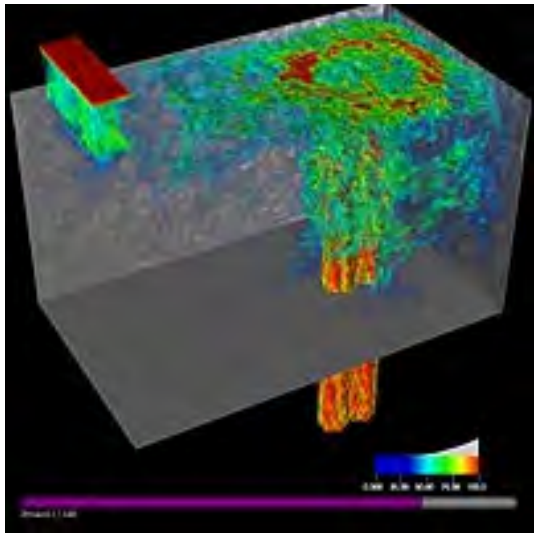
Tukey- High Performance Visualization Cluster

- 96 AMD Dual Opteron 6128 Compute Nodes
  - 16 total CPU cores per node
  - 64 GB RAM
  - 2 NVIDIA Tesla M2070 GPUs, 6 GB RAM each
- 6 terabytes of CPU RAM, 1.1 terabytes of GPU RAM
- Peak GPU Performance: Over 98 TeraFLOPS
- Cross-Mounted Filesystem with Mira









All Sorts of Tools

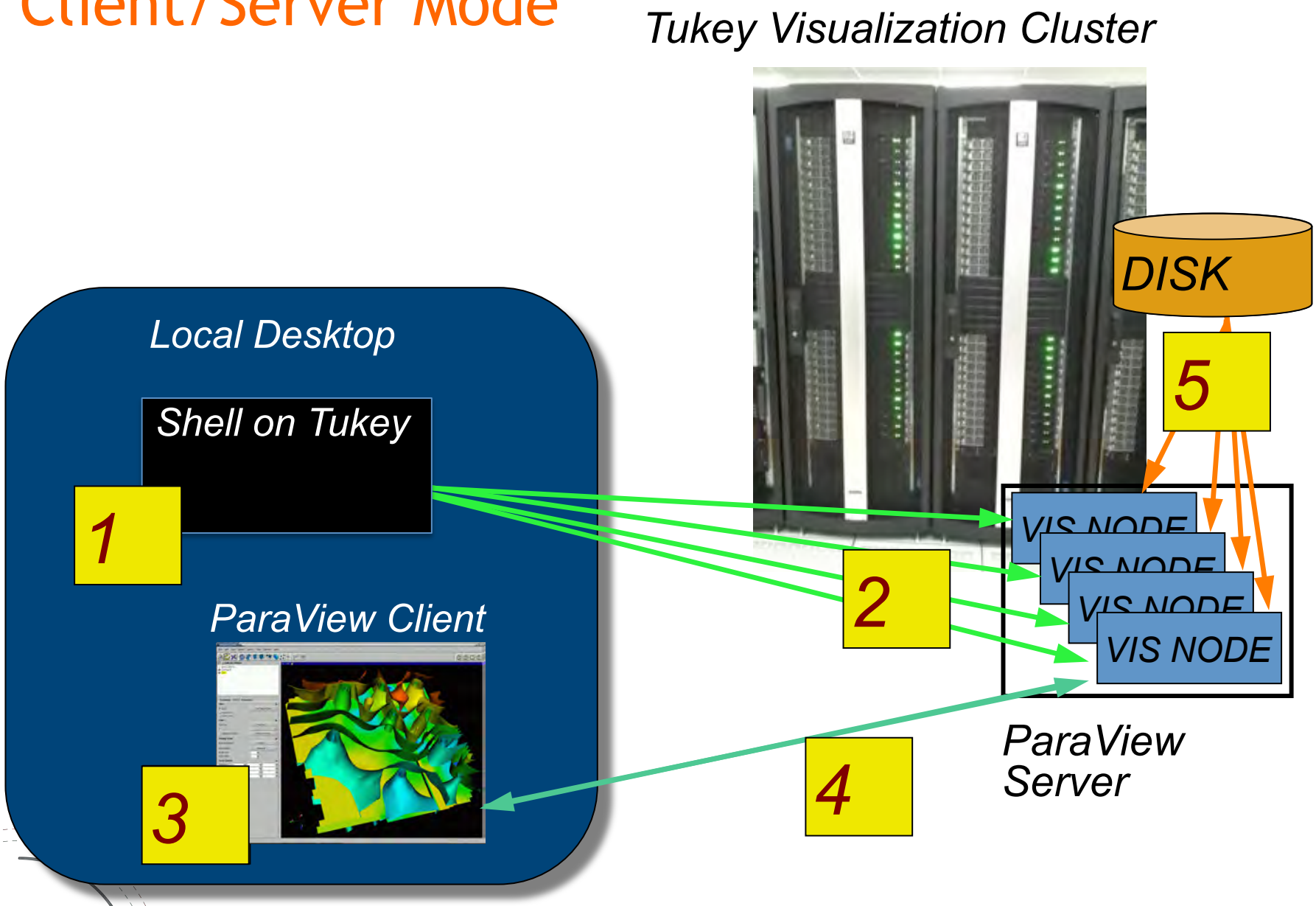
- Visualization Applications
  - VisIt
  - ParaView
  - EnSight
- Domain Specific
  - VMD, PyMol, RasMol
- APIs
  - VTK: visualization
  - ITK: segmentat & registration
- GPU performance
  - vl3: shader-based vol ren
  - Scout: GPGPU acceleration
- Analysis Environments
  - Matlab
  - Parallel R (ORNL)
- Utilities
  - GnuPlot
  - ImageMagick
- Visualization Workflow
  - VisTrails

Visualization Modes

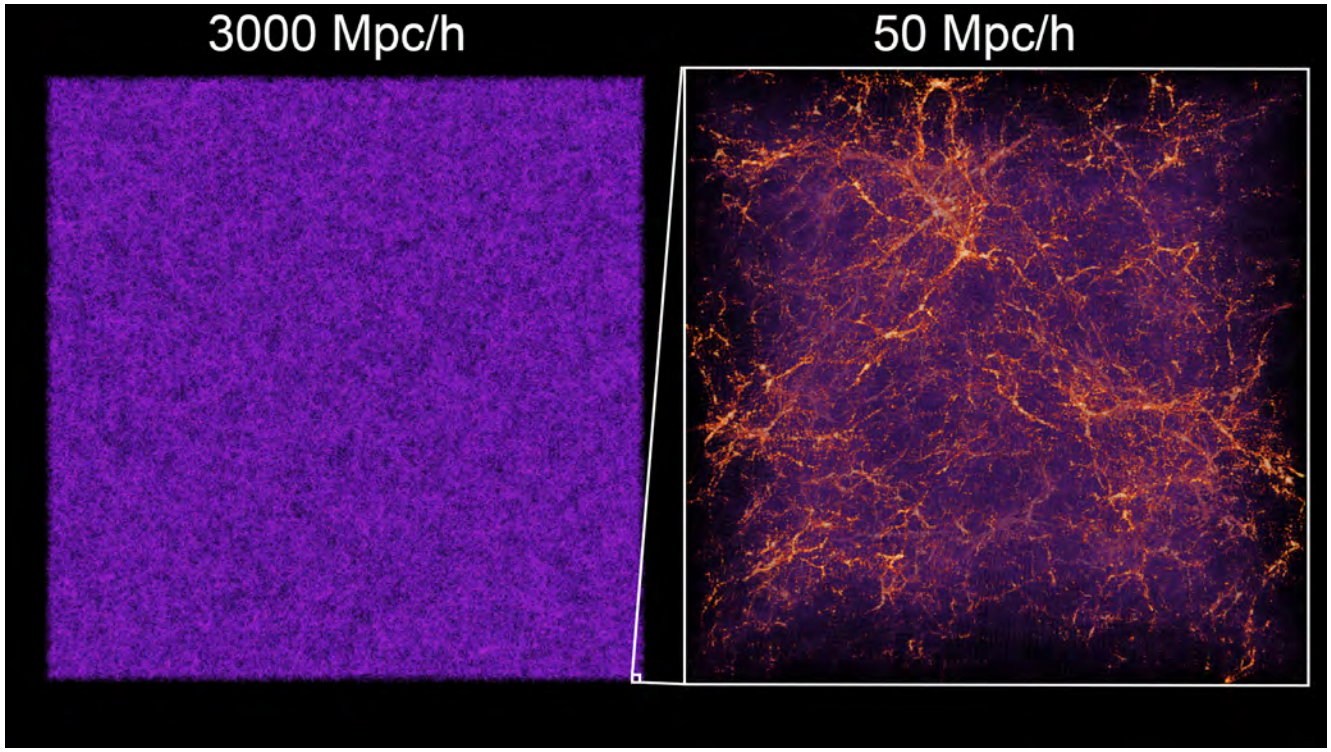
- Interactive
  - Data exploration
  - Generating movies
- Batch
  - Known quantities/configurations
  - Generating movies

- Stand-alone
  - smaller data – move locally and visualize there
- Client/server mode
  - Use interactive sessions to produce state for visualizing larger data / time series in batch mode.

Client/Server Mode



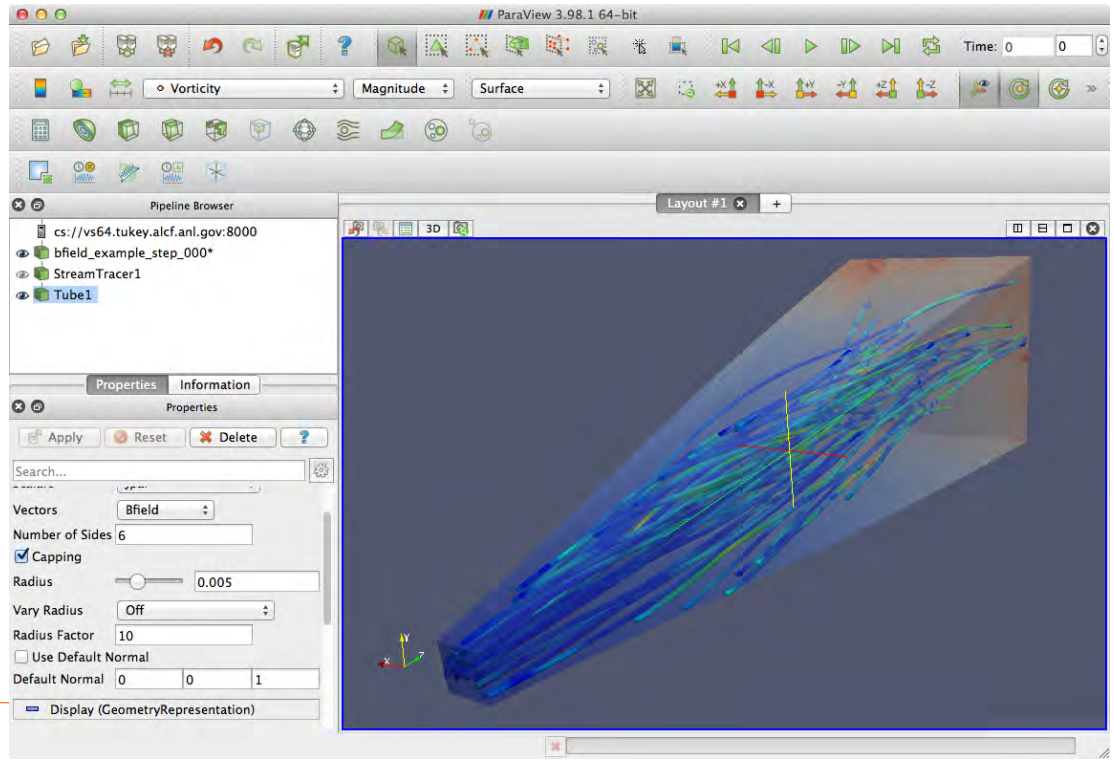
HACC: Cosmology Simulation



- Data: 10Kx10Kx800 (~316GB)
- Visualization considerations:
  - Number of visualization nodes, processes, GPUs
  - Trade-offs
    - Are you memory, compute, or rendering bound?

ParaView Demo

- Tutorial Page:
  - www.alcf.anl.gov/user-guides/paraview-tukey
- Launch pvserver
  - `qsubi -n 4 -t 60 -A project_id -q pubnet`
  - `mpiexec -f $COBALT_NODEFILE -np 4 pvserver --server-port=8000`
- ParaView Client
  - Configure server
  - Connect
  - Do some vis



Argonne Leadership Computing Facility

ParaView States and Scripting

- Save State
  - .pvsm (for restoring state in interactive mode)
  - .py (for use with pvbatch)
  - saved on the client side
- Edit .py script
  - short example, loop over time steps, saving images

```
try: paraview.simple
except: from paraview.simple import *
paraview.simple._DisableFirstRenderCameraReset()
RenderView2 = CreateRenderView()
…
bfield_step_000 =
XMLPartitionedStructuredGridReader( guiName="bfield_step_000*",
PointArrayStatus=['Jpar', 'Wpar', 'Bfield'], CellArrayStatus=[],
FileName=[‘bfield_step_0001.pvts', ‘bfield_step_0002.pvts',
‘bfield_step_0003.pvts', bfield_step_0004.pvts', ‘bfield_step_0005.pvts'] )
```

ParaView States and Scripting

- Edit .py script

```
…
#Render()

time_vals = bfield_step_000.TimestepValues

for i in range(len(time_vals)):

    RenderView2.ViewTime = time_vals[i]
    RenderView2.StillRender()

    IMAGE_FILE=”frame_%04d.png" % int(i)
    print "saving: " + IMAGE_FILE
    WriteImage(IMAGE_FILE)
```

ParaView Batch Mode

- Copy .py script
  - need to copy the .py script to Tukey
- pvbatch
  - `mpiexec -f $COBALT_NODEFILE -np 4 pvbatch <script.py>`
  - could run in qsubi mode, or totally batch

Data Logistics

- Storing data
  - /gpfs/mira-fs0/projects/<project_id>
- Backup data
  - HPSS Tape Archive
  - www.alcf.anl.gov/user-guides/using-hpss
- Moving data
  - scp
    - handy for small manageable data
  - Globus Online (www.globusonline.org)
    - useful for bigger/ more complex data
    - GridFTP servers
    - Local downloads (Globus Connect)
    - Fire and forget

Data File Formats (ParaView & VisIt)

- VTK
- Parallel (partitioned) VTK
- VTK MultiBlock (MultiGroup, Hierarchical, Hierarchical Box)
- Legacy VTK
- Parallel (partitioned) legacy VTK
- EnSight files
- EnSight Master Server
- Exodus
- BYU
- XDMF
- PLOT2D
- PLOT3D
- SpyPlot CTH
- HDF5 raw image data
- DEM
- VRML
- PLY
- Polygonal Protein Data Bank
- XMol Molecule
- Stereo Lithography
- Gaussian Cube
- Raw (binary)
- AVS
- Meta Image
- Facet
- PNG
- SAF
- LS-Dyna
- Nek5000
- OVERFLOW
- paraDIS
- PATRAN
- PFLOTRAN
- Pixie
- PuReMD
- S3D
- SAS
- Tetrad
- UNIC
- VASP
- ZeusMP
- ANALYZE
- BOV
- GMV
- Tecplot
- Vis5D
- Xmdv
- XSF

Data Wrangling

- HDF5
  - Get header information
    - h5dump –H filename.h5d
  - List of data sets
    - h5ls filename.h5d
  - Combine data sets
    - h5copy

- netCDF
- Get header information
- ncdump –h filename.nc

- Dump contents of a netCDF file to text
- ncdump infile.nc > outfile.cdl

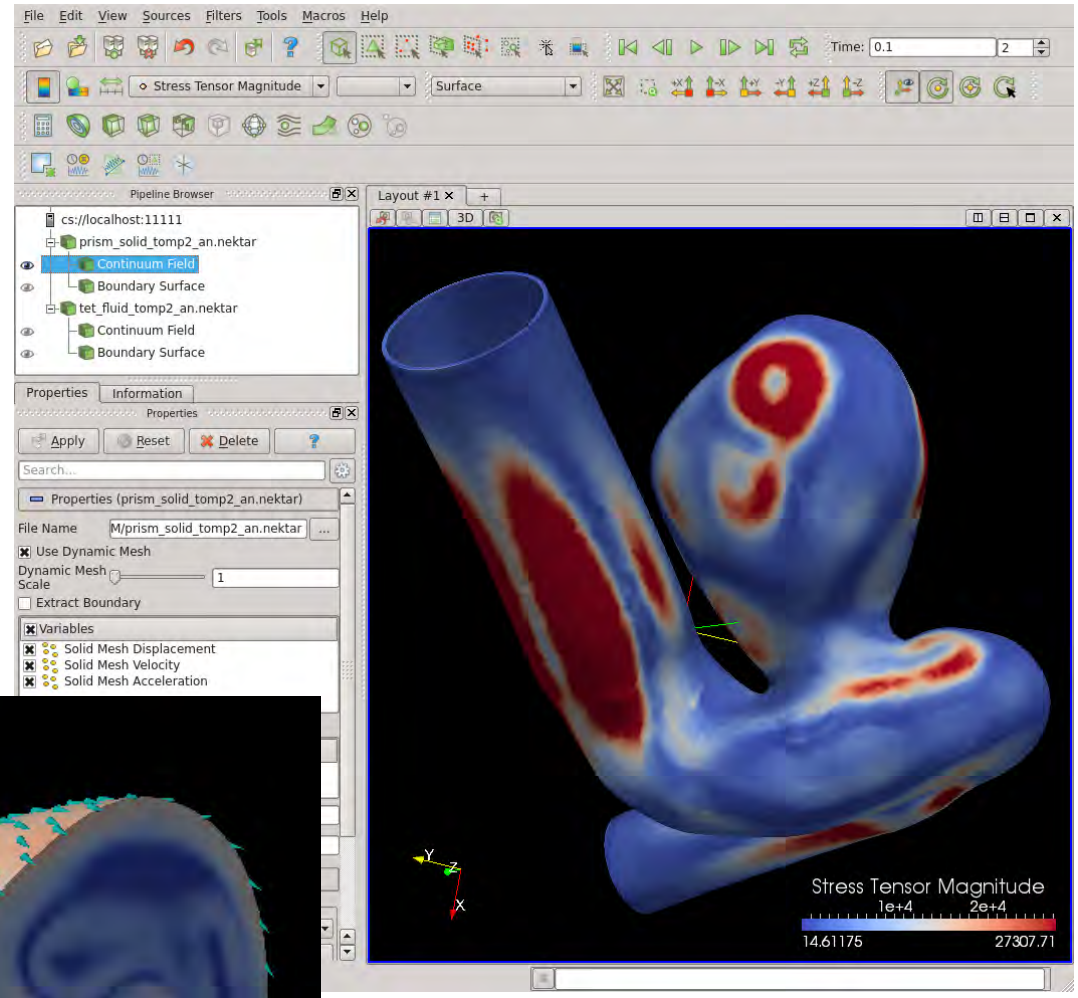
- Generate data sets
- ncgen –o file.nc infile.cdl

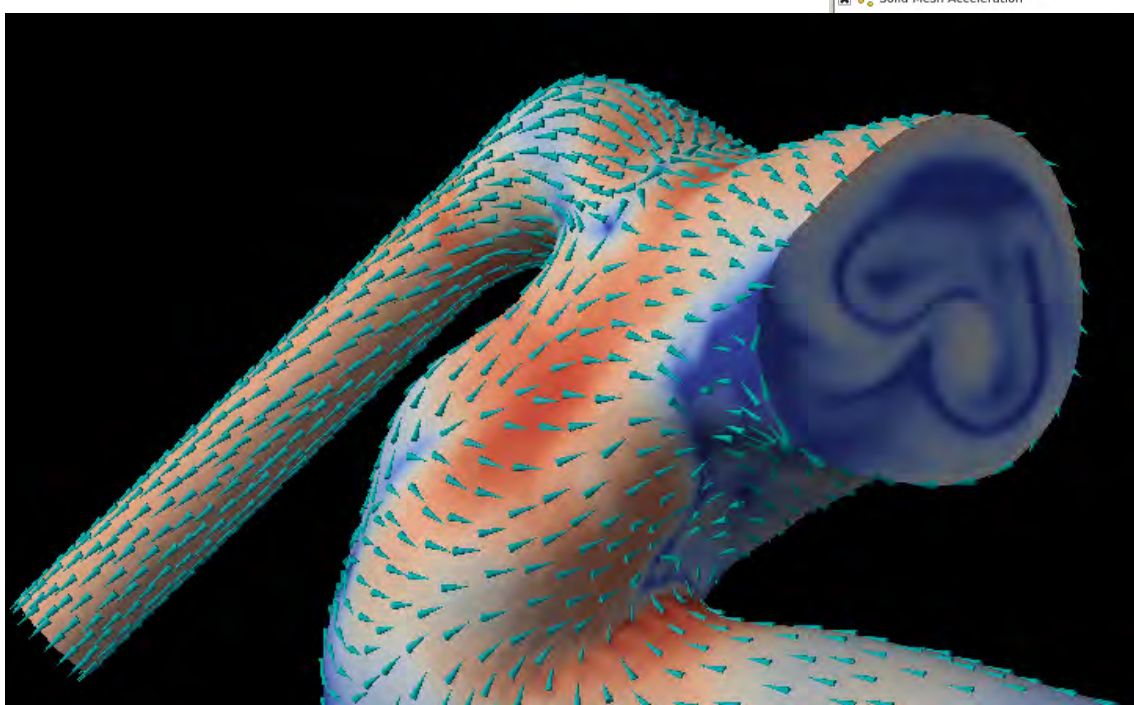
Data Wrangling

- XDMF
  - XML wrapper around HDF5/binary data
  - Can define
    - data sets
    - subsets
    - hyperslabs
- vtk
  - Could add to your simulation code
  - Can write small utilities to convert data
    - Use your own read routines
    - Write vtk data structures
  - C++ and Python bindings

Data Organization

- Format
  - Existing tools support many flavors
  - Use one of these formats
  - Write a custom reader for existing tool
    - NekTar example
  - Write your own custom vis tool





Data Organization

- Serial vs. Parallel/Partitioned
  - Single big file vs. many small files: middle ground generally best
    - vtk data types
    - XDMF for ParaView
    - Custom XML description

```
<DMDatasetConfig>
    <Name>HACC Density Field</Name>
    <Dimensions>
        <Size>4</Size>
        <DIM0 name="T" size="1" />
        <DIM1 name="X" size=”800" />
        <DIM2 name="Y" size=”10240" />
        <DIM3 name="Z" size=”10240" />
    </Dimensions>
    <DataType>RAW_3D</DataType>
    <BytesPerElement>4</BytesPerElement>
    <Endianness>Big</Endianness>
    <FileDB>
        <Directory>/path/to/STEP130</Directory>
        <FileExtents DIM0="1" DIM1=”160" DIM2=”160"
DIM3=”160" />
        <FileList size=”20480">
```

```
<File>
  <Path>rho.0000.raw</Path>
    <Offset DIM0="0" DIM1="0" DIM2="0" DIM3="0" />
</File>
<File>
    <Path>rho.0001.rho.raw</Path>
    <Offset DIM0="0" DIM1="0" DIM2="0" DIM3=”160" />
</File>
<File>
    <Path>rho.0002.rho.raw</Path>
    <Offset DIM0="0" DIM1="0" DIM2="0" DIM3=”320" />
</File>
…
```

Data Organization

- Serial vs. Parallel
  - Performance trade-offs
    - vtk/paraview: serial files all data read on head node, partitioned and distributed
    - vtk/paraview: parallel files: serial files partitioned across processes, read in parallel

Performance example:

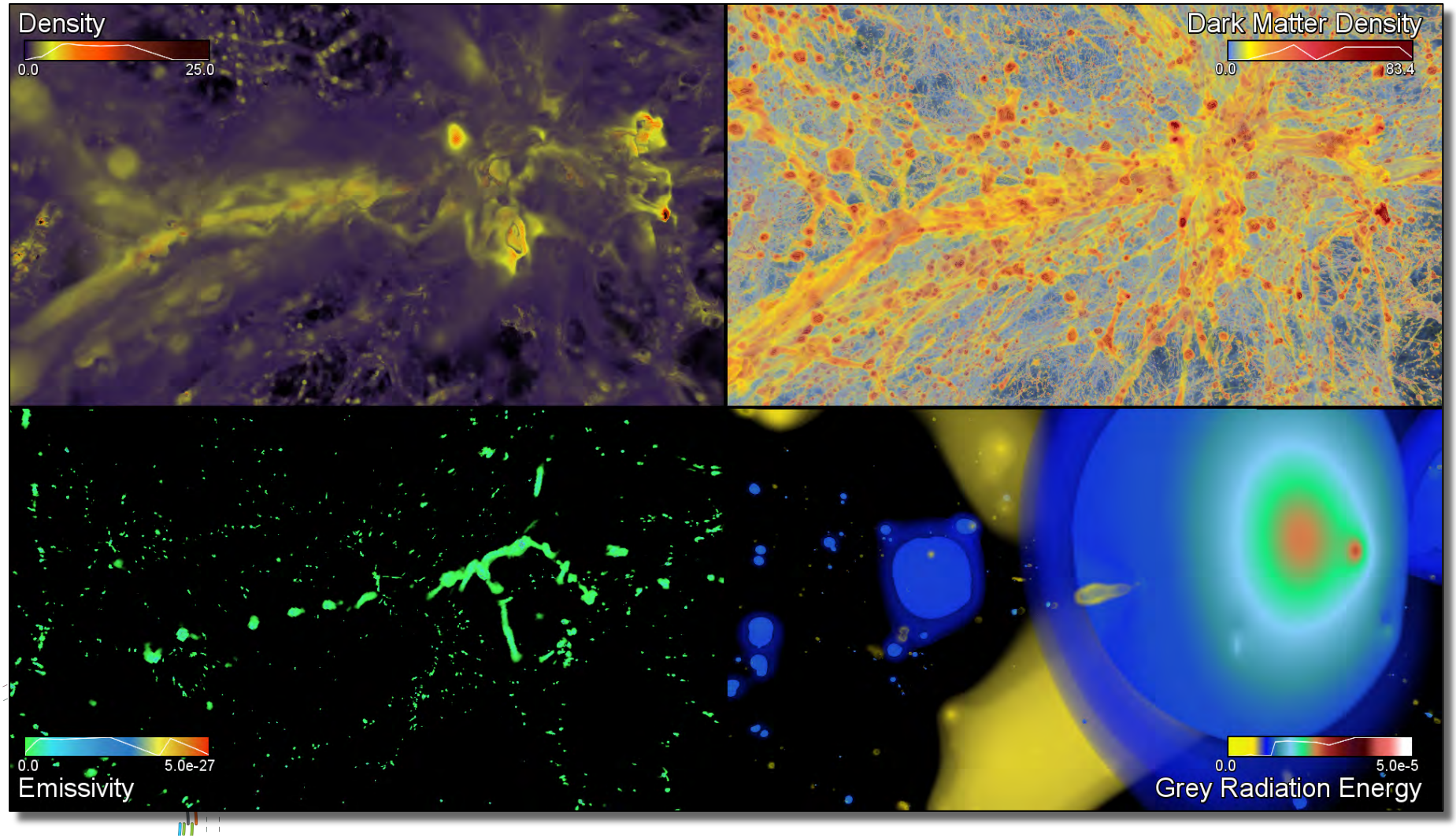
- Single serial .vtu file (unstructured grid)
  - Data size: ~3.8GB
  - Read time on 64 processes: > 15 minutes
    - most of this was spent partitioning and distributing
- Partitioned .pvtu file (unstructured grid)
  - Data size: ~8.7GB (64 partitions)
  - Read time on 64 processes: < 1 second

Movie Making

- VisIt/ParaView: save animation
  - can save animation file
  - can save individual images
    - I tend to do this, allows for more flexibility
- Adobe Premiere, Final Cut, etc.
  - commercial tools enable lots of options, but do cost money (ALCF does not provide these tools)
- ImageMagick tools for annotating and combining images in different ways

Annotation, compositing, scaling…

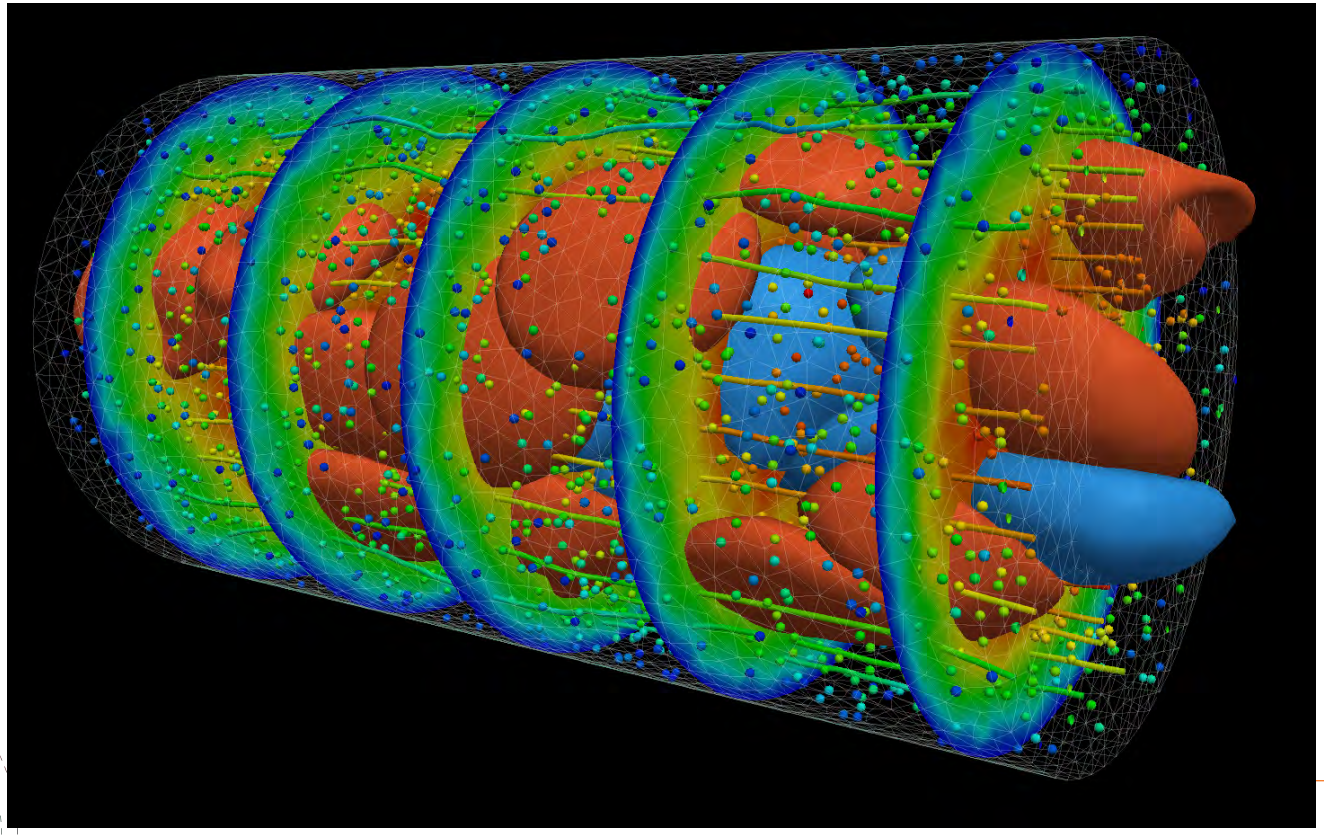
- ImageMagick
  - convert, composite, montage, etc.



Movie Creation

- Combine multiple segments of frames
  - Create a directory of symbolic links to all frames in order

- ffmpeg: Movie encoding
  - ffmpeg –sameq –i frame.%04d.png movie.mp4

More info…

- Tukey user guide:
  - www.alcf.anl.gov/user-guides/tukey
- Online ParaView tutorial:
  - www.alcf.anl.gov/user-guides/vis-paraview-red-blood-cell-tutorial