# Scientific visualization with ParaView
# Part 1

Alex Razoumov
alex.razoumov@westdri.ca


Digital Research Alliance of Canada

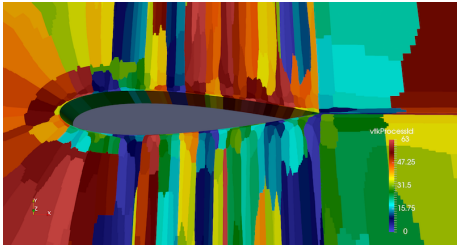
✔ slides, data, codes at `https://folio.vastcloud.org/introparaview`

- the link will download a file `paraview.zip` (∼30MB)
- unpack it to find `codes/`, `data/` and `slides{1,2}.pdf`
- command line: `wget <link> -O paraview.zip`
- alternative temporary link `http://temp.sh/JODMr/paraview.zip`

✔ install ParaView 6.0.x on your laptop from `http://www.paraview.org/download`

# Workshop outline

- Introduction to sci-vis: general ideas, tools, plotting vs. multi-dimensional vis. + Plotly ($\sim$ 30 mins)
- Overview of current general-purpose multi-dimensional sci-vis tools
- ParaView's architecture
- Importing data into ParaView: raw binary, VTK data types, NetCDF/HDF5, OpenFOAM
- Basic workflows: filters, creating a pipeline, working with vector fields

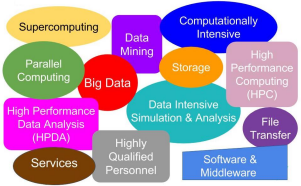



---

- Scripting: ways to run scripts, examples, trace tool, programmable filter/source, camera animations
- Animation: three approaches, camera animation, one big exercise on scripting/animation, single timeline with many properties
- Remote visualization
  - running ParaView in **client-server** mode
  - opening and load balancing of very large, multi-GB datasets
  - recommendations on running on cluster GPUs vs. multiple CPUs
  - writing, debugging, running PV Python scripts as **batch rendering jobs**
- Not covered today: ParaView Cinema, in-situ visualization with Catalyst

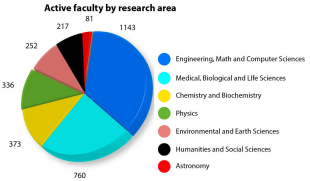
# Who are we?

https://docs.ALLIANCECAN.ca

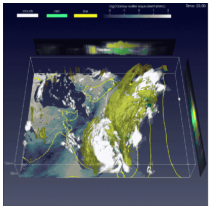


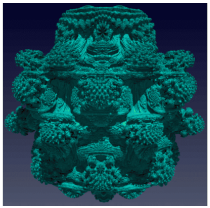

We provide Advanced Research Computing (ARC) infrastructure and services, a.k.a. everything beyond a standard desktop, at *no cost* to researchers

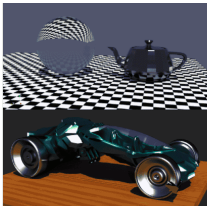
- SUPERCOMPUTERS/HPC, cloud computing
- data storage, management, transfer
- support, training, visualization




- National Visualization Team `https://ccvis.netlify.app`
- Visualize This contests since 2016, IEEE SciVis 2021 Contest `https://ccvis.netlify.app/contests`







# Scientific visualization (of spatially defined data)

- Sci-vis is the process of mapping scientific data to VISUAL FORM
  - much easier to understand images than a large set of numbers
  - for interactive data exploration, debugging, communication with peers

| FIELD | VISUALIZATION TYPE |
|---|---|
| computational fluid dynamics | 2D/3D flows, density, temperature, tracers |
| climate, meteorology, oceanography, planetary interiors | fluid dynamics, clouds, chemistry, etc. |
| astrophysics: from galaxy and star formation to stellar hydrodynamics, accretion, interstellar medium, and everything in between | 2D/3D fluids, particle data, $\leq$6D radiation field, magnetic fields, gravitational fields |
| quantum chemistry | 3D wave functions |
| molecular dynamics (phys, chem, biology) | particles (atoms and molecules) |
| medical imaging | MRI, CT scans, ultrasound, time-series data |
| geographic information systems | elevation, rivers, towns, roads, layers, etc. |
| humanities, social sciences, info-vis | abstract data, or any of the above |
| bioinformatics | networks, trees, sequences |

- "zoom in on a cluster of cells and see their top gene expressed" (as in spatial zoom? can you encode gene expression with a variable?)
- "colour by subsets of cell types upon different clustering resolutions"

# 1D/2D plotting vs. multi-dimensional visualization

- **1D/2D plotting**: plotting functions of one variable, 1D tabulated data
  - something as simple as gnuplot or pgplot
  - highly recommend in Python: • `https://matplotlib.org` (static plots)
    • `https://plotly.com/python` (interactive html5) • `https://bokeh.org` (interactive html5)
    • `https://seaborn.pydata.org` (statistical data visualization)
    • `https://altair-viz.github.io` (declarative visualization library)
    • `https://plotnine.org` (grammar of graphics for Python) • `https://holoviews.org`
  - another excellent option: R's `https://ggplot2.tidyverse.org` (based on the grammar of graphics)
- **2D/3D visualization**
  - displaying multi-dimensional datasets, typically data on structured (uniform and multi-resolution) or unstructured grids (that have some topology in 2D/3D)
  - rendering often CPU- and/or GPU-intensive
- Avoid proprietary tools unless they offer a clear advantage (rarely the case)
  - large $$
  - limitations on where you can run them, which machines/platforms, etc.
  - user base usually smaller than for open-source tools, more difficult to get help
  - once you start accumulating scripts, you lock yourself into using these tools, and consequently paying $$ on a regular basis

# Python visualization landscape by renderer, circa 2017

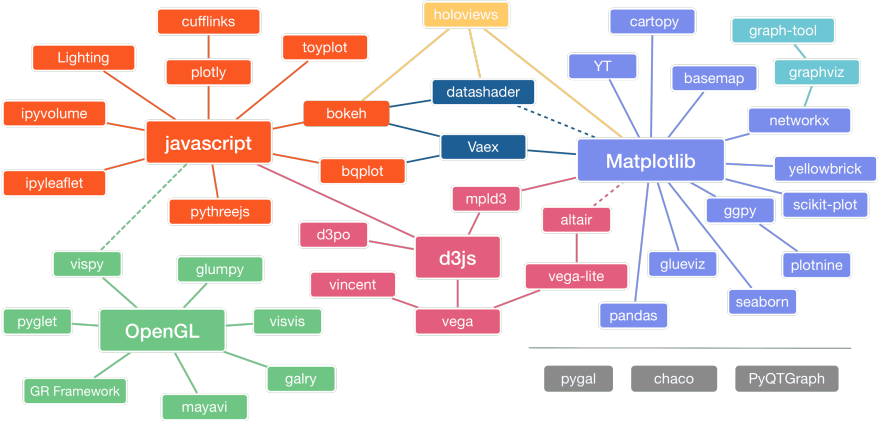
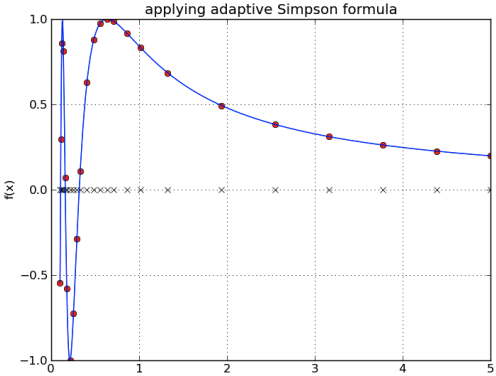



Figure: Nicolas Rougier's famous adaptation of the Python Visualization Landscape slide from Jake VanderPlas's keynote at PyCon 2017 (heavily influenced by traditional 1D/2D plotting workflows)
Source: `https://github.com/rougier/python-visualization-landscape`
Also see `https://pyviz.org` for an up-to-date list of tools

# Matplotlib example: 1D plotting

## Adaptive Simpson integration



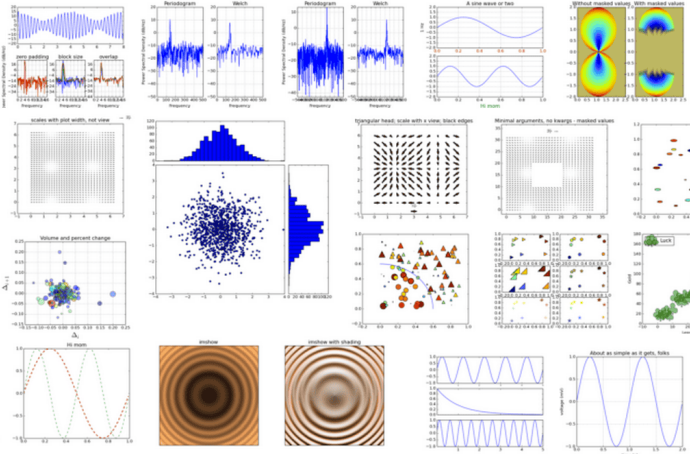

- Simple Python function `simpsonAdaptive(function,a,b,tolerance)` handles both calculation and plotting (∼40 lines of code)
- Code in `codes/adaptive.py`

# Python Imaging Library (PIL) example: 2D plotting

Edge detection using numerical differentiation

- Simple Python script reading a colour PNG image, calculating gradient of the blue filter, plotting its norm in black/white (20 lines of code total)
- Code in `codes/fuji.py`

# Matplotlib gallery contains hundreds of examples



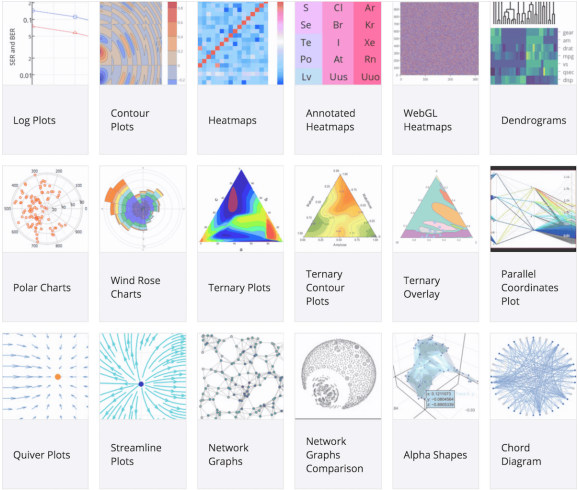

- `https://matplotlib.org/stable/gallery` – click on any plot to get its source code
- `http://www.labri.fr/perso/nrougier/teaching/matplotlib` is a really good introduction

# Bokeh gallery


- Open-source project from Continuum Analytics
  `https://docs.bokeh.org/en/latest/docs/gallery.html`
- Produces dynamic html5 visualizations for the web
- Basic server-less interactivity packed into a json object; more complex interactions via a Bokeh server

# Plotly Python library

- Open-source project from Plot.ly
  `https://plot.ly/python`
- APIs for Python (with/without Jupyter), R, JavaScript, MATLAB
- Produces dynamic html5 visualizations for the web; can also render into static formats
- Interactive elements in self-contained html5 files can even include sliders and animation
- For callback functions and custom interactions, use Plotly Dash
- Quick intro to Plotly and Dash
  `https://folio.vastcloud.org/plotly`

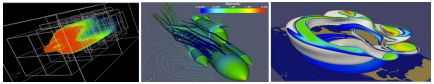


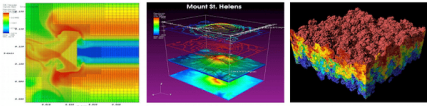
# Today's focus: multi-dimensional sci-vis

- In 3D:
  1. harder to navigate, multiple overlapping fields ⇒ need layers and interactive filters
  2. larger datasets ⇒ processing times can be large ⇒ GPU and distributed rendering
- Requirements:
  - open-source + multi-platform (Linux/Mac/Windows) + general-purpose
  - visualize scalar and vector fields
  - data on top of structured (e.g. Cartesian) and unstructured meshes in 2D and 3D, particles, polygonal meshes, irregular topologies, AMR / multi-resolution datasets
  - ability to handle very large datasets (GBs to TBs), up to $10^{\sim 12}$ resolution elements
  - ability to scale to large ($10^3 - 10^5$ cores) computing facilities + parallel I/O
  - GUI interactive manipulation + Python scripting
  - client-server for remote interactive visualization
  - support for most common data formats and rendering views

# ParaView and VisIt

- For the past 20+ years choice between ParaView and VisIt, with a ∼85/15 breakdown
- We taught full-day workshops on both over the years






- `https://www.paraview.org`
- Started in 2000 as a collaboration between Los Alamos and Kitware Inc., later joined by Sandia and other partners; first release in 2002
- Developed to visualize extremely large datasets on distributed memory machines
- ParaView is based on VTK (Visualization Toolkit): same authors, PV = GUI on top of the C++ VTK classes
- Uses MPI for distributed-memory parallelism on HPC clusters
- Big ecosystem of tools: web, in-situ, Cinema
- Latest 6.0.1

- `https://visit-dav.github.io/visit-website`
- Developed to visualize results of terascale simulations, first release in fall 2002
- Over 80 visualization features (contour, mesh, slice, volume, molecule, ...)
- Reads over 110 different file formats
- Full integration with VTK library
- Uses MPI for distributed-memory parallelism on HPC clusters
- Smaller ecosystem
- Latest 3.4.2

# Why ParaView for this workshop?

- Back in ∼2010, I had to pick one
  - both ParaView's and VisIt's binaries are widely available, in active development
  - both can do remote client-server visualization, very good parallel scalability
  - ParaView and VisIt interfaces are very different
- Tight integration with VTK (developed by the same folks), 130 input formats
- A number of add-on projects
  - **ParaViewWeb** is a JavaScript library to write web applications that talk to a remote ParaView server; can reproduce full standalone ParaView in a web browser (WebGL + remote processing)
  - **vtk.js** is a scientific rendering library for the web (standalone WebGL)
  - **ParaView Glance** is a standalone open-source web app for in-browser 3D sci-vis
  - **Catalyst** is an open-source *in-situ visualization* library that can be embedded directly into parallel simulation codes; interaction through ParaView scripts
  - **ParaView Cinema** for interactive visualization from pre-rendered images (rotation, panning, zooming, variables on/off)
- Stereo viewing on 3D hardware; experimental builds for various head-mounted displays (HMDs), Looking Glass displays
- We also use and teach `https://visit-dav.github.io/visit-website`, other open-source packages

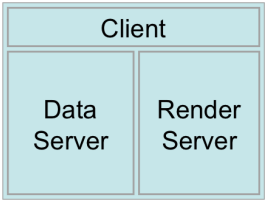
# ParaView architecture and GUI

# ParaView's distributed parallel architecture

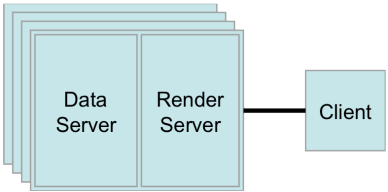
Three logical components inside ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **Data Server** – The unit responsible for data reading, filtering, and writing. All of the pipeline objects seen in the pipeline browser are contained in the data server. The data server can be parallel.
- **Render Server** – The unit responsible for rendering. The render server can also be parallel, in which case built-in parallel rendering is also enabled.
- **Client** – The unit responsible for establishing visualization. The client controls the object creation, execution, and destruction in the servers, but does not contain any of the data, allowing the servers to scale without bottlenecking on the client. If there is a GUI, that is also in the client. The client is always a serial application.

# Two major workflow models




Standalone mode: computations and user interface run on the same machine




Client-server mode: `pvserver` on a multi-core server or distributed cluster

# Advantages of remote client-server rendering

- Standalone ParaView has its limitations: limited memory, limited I/O bandwidth, and limited CPU / GPU power
- For example, on a workstation with 48GB memory works well up to $2048^3$ single-precision float variable on structured grids stored locally
- Larger / higher-res datasets, more complex grids, or datasets requiring complex filters won't fit
- Typical problem that won't fit on a 48GB workstation and is too slow to read via sshfs: simulation of the airflow around a wing on an *Unstructured Grid* (`*.vtu`) with $246 \times 10^6$ cells (equiv. to $627^3$), one variable takes 25GB — however, can do this interactively without problem on 64 cores on a cluster with `pvserver` taking $\sim$ 120 GB memory
- We'll study remote ParaView in more detail towards the end of this workshop

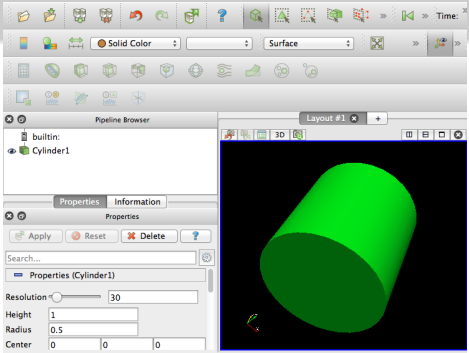
# Starting ParaView

- Today we'll do everything in standalone ParaView on your desktop

- Linux/Unix: type paraview at the command line
- Mac: click paraview in Applications folder
- Windows: select paraview from start menu

- ParaView GUI should start up
- The server `pvserver` is run for you in the background

# User interface

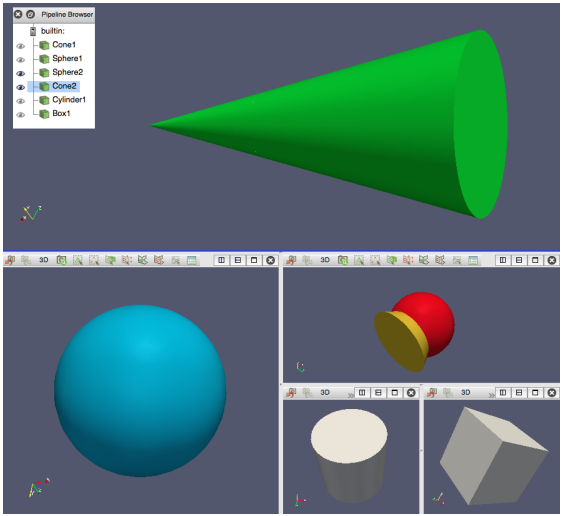



- **Pipeline Browser**: data readers, data filters, can turn visibility of each object on/off
- **Object Inspector**: view and change parameters of the current pipeline object (via tabs properties-display-info or properties-info)
- **View window**: displays the result

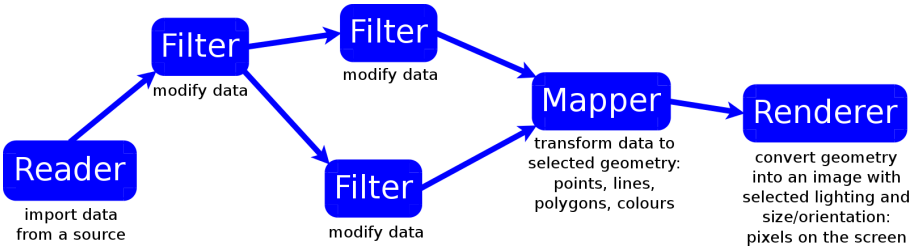
1. Find the following in the toolbar: “Connect”, “Disconnect”, “Toggle Colour Legend Visibility”, “Edit Colour Map”, “Rescale to Data Range”
2. Load a predefined dataset: in ParaView select Sources → Cylinder
3. Try dragging the cylinder using the left mouse button; also try the same with the right and middle buttons
4. Identify drop-down menus; try changing to a different view (e.g. from Surface to Wireframe) or changing colour via “Edit Colour Map”

# ParaView windows

- Reproduce this image
- Use objects from the “Sources” menu (cone, sphere, cylinder, box), can edit their properties
- Use the icons in the upper right of each window to split the view
- Optionally can link any two views by right-clicking on an image, selecting “Link Camera”, and clicking on a second image



# Data flow in VTK




- Data goes through Mapper which knows how to draw it, places that data into the rendered scene via a VTK Actor
  - `mapper.setInputConnection(object.getOutputPort())`
- Actor is an OpenGL object = the part that is rendered
  - takes data from Mapper: `actor.setMapper(mapper)`
  - passed to Renderer: `renderer.addActor(actor)`
- Renderer can hold multiple actors
- RendererWindow (on the screen) can hold multiple renderers

# Importing data into ParaView

# Data sources

- Generate data with a *Source* object
- Read data from a file

ADAPT Files (*.nc *.cdf *.elev *.ncd)
Adaptive cosmo files (*.cosmo)
ADIOS2 BP3 File (CoreImage) (*.bp)
ADIOS2 BP4 Directory (CoreImage) (*.bp)
ADIOS2 BP4 Metadata File (CoreImage) (md.idx)
AMR Enzo Files (*.boundary *.hierarchy)
AMR Flash Files (*.flash *.flash)
AMReVolodyne Files (*.samr *.Xamr *.XAMR)
AMReX/BoxLib plotfiles (grids) (plt*)
AMReX/BoxLib plotfiles (particles) (plt*)
ANSYS Files (*.inp)
AU File Files (*.aux)
AVS UCD Binary/ASCII Files (*.inp)
BOV Files (*.bov)
BYU Files (*.g)
CAM NetCDF (Unstructured) (*.nc *.ncdf)
Case file for restarted CTH outputs
CCSM MTSD Files (*.nc *.cdf *.elev *.ncd)
CCSM STSD Files (*.nc *.cdf *.elev *.ncd)
CEAucd Files (*.ucd *.inp)
CGNS Files (*.cgns)
Chombo Files (*.hdf5 *.h5)
CityGML files (*.gml *.xml)
Claw Files (*.claw)
CMAT Files (*.cmat)
CML (*.cml)
CONVERGE CFD (*.h5)
Cosmology Files (*.cosmo64 *.cosmo)
CTRL Files (*.ctrl)
Curve2D Files (*.curve *.ultra *.ult *.u)
DDCMD Files (*.ddcmd)
Delimited Text (*.csv *.tsv *.txt *.CSV *.TSV *.TXT)
DICOM Files (directory) (*.dcm)
DICOM Files (single) (*.dcm)
Digital Elevation Map Files (*.dem)
Dyna3D Files (*.dyn)
EnSight Files (*.case *.CASE *.Case)
EnSight Master Server Files (*.sos *.SOS)
ENZO AMR Particles (*.boundary *.hierarchy)
Exodus11 (*.g *.gc *.ex2 *.ex2v2 *.exo *.gen *.par
ExtrudedVol Files (*.esvol)
Facet Polygonal Data Files (*.facet)
Fides Data Model File (JSON) (*.json)
Fides Files (ADIOS2 BP) (*.bp)
FLASH AMR Particles Reader (*.flash *.flash)

FLASH Files (Visit) (*.flash *.f5)
Fluent Case Files (*.cas)
Fluent Files (Visit) (*.cas)
FVCOM MTMD Files (*.nc *.cdf *.elev *.ncd)
FVCOM MTSD Files (*.nc *.cdf *.elev *.ncd)
FVCOM Particle Files (*.nc *.cdf *.elev *.ncd)
FVCOM STSD Files (*.nc *.cdf *.elev *.ncd)
Gadget Files (*.gadget)
Gaussian Cube Files (*.cube)
GDAL Raster (*.tif *.gen *.thf *.adf *.arg *.blx *.xlb)
GDAL Vector (*.shp *.faa *.bnn *.dxf *.csv *.geojson)
Generic10 files to MultiBlockDataSet (*.gio)
Generic10 files to UnstructuredGrid (*.gio)
GGCM Files (*.3df *.mer)
gITF 2.0 Files (*.gltf *.glb)
GTC Files (*.h5)
GULP Files (*.trg)
H5NImross Files (*.h5nimrod)
H5Part particle files (*.h5part)
HyperTreeGrid (*.htg)
HyperTreeGrid (partitioned) (*.phtg)
Image Files (*.pnm *.ppm *.xbi *.spr *.imgvol)
JPEG Ima
LAMMPS Dump Files (*.dump)
LAMMPS Struct. (*.eam *.meam *.rigid *.lammps)
Legacy VTK files (*.vtk *.vtk.series)
Legacy VTK Files (partitioned) (*.pvtk)
Lines Files (*.lines)
LODI Files (*.no *.cdf *.elev *.ncd)
LODI Particle Files (*.nc *.cdf *.elev *.ncd)
LSDyna (*.k *.lsdyna *.d3plot d3plot)
M3DC1 Files (*.h5)
Meta Image Files (*.mhd *.mha)
Meta-Generic10 files (*.gios)
Metafile for restarted exodus outputs
MFIX netcdf Files (*.nc)
MFiX Res Files (Visit) (*.RES)
MFIX Unstructured Grid Files (*.RES)
Mill Files (*.m1)
Miranda Files (*.mir *.raw)
MM5 Files (*.mm5)
MotionFX CFG Files (*.cfg)
MPAS NetCDF (Unstructured) (*.ncdf *sic)
MRC Image Files (*.mrc *.ali *.st *.rec)
Multilevel 3D Plasma files (*.m3d *.h5)

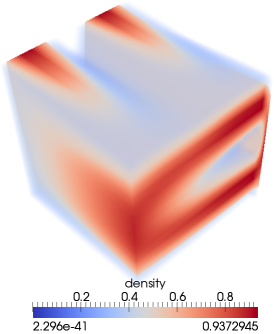
NASTRAN Files (*.nas *.f06)
Nek5000 (*.nek3d *.nek2d *.nek5d *.nek5000 *.nek)
netCDF generic and CF conventions (*.ncdf *.nc)
Nrrd Raw Image Files (*.nrrd *.nhdr)
OME TIFF Files (*.ome.tif *.ome.tiff)
OpenFOAM (*.foam)
OpenFOAM Files (Visit) (*.controlDict)
openPMD files (*.pmd)
OVERFLOW Files (Visit) (*.dat *.save)
ParaDIS Files (*.pds *.data *.dat)
ParaDIS Tecplot (Mid *.field *.cyl *.cylinder *.dat)
Parallel POP Ocean NetCDF (*.pop.ncdf *.pop.nc)
ParaView Data Files (*.pvd)
ParaView Ensemble Data (*.pve)
PATRAN Files (*.neu)
PFLOTRAN Files (*.h5)
Phasta Files (*.pht)
PIO Dump Files (*.pio)
Pixie Files (*.h5)
PLOT2D Files (*.p2d)
PLOT3D Files (*.xyz)
PLOT3D Meta Files (*.p3d)
PLY Polygonal File Format (*.ply *.ply.series)
PNG Image Files (*.png)
POINTED Files (*.3D)
POP Ocean NetCDF Rectilinear (*.pop.nc*)
POP Ocean NetCDF Unstructured (same as prev.)
proSTAR Files (*.cel *.vrt)
Protein Data Bank Files (*.pdb)
Protein Data Bank Files (Visit) (*.ent *.pdb)
PTS (Point Cloud) Files (*.pts)
Radiance HDR files (*.hdr)
Raw (binary) Files (*.raw)
RAW Files (*.raw)
SAMRAI series files (*.samrai)
SAR Files (*.SAR *.sar)
SAS files (*.sasgeom *.sas *.sasdata)
SEG-Y Files (*.sgy *.segy)
SEP file (*.H *.H1)
Silo Files (*.silo *.pdb *.silo.series *.pdb.series)
SLAC Mesh files (*.ncdf *.nc)
SLAC Particle Files (*.ncdf *.netcdf)
Spherical Files (*.spherical *.sv)
Spy Plot History Files (*.hscth *hsct*)
SpyPlot CTH dataset (*.spct* spot*)

Stereo Lithography (*.stl *.stl.series)
Tecplot Binary Files (Visit) (*.plt)
Tecplot Files (*.tec *.TEC *.Tec *.tp *.TP *.dat)
Tecplot Files (Visit) (*.tec *.TEC *.Tec *.tp *.TP)
Tecplot Table (*.dat *.DAT)
Tetrad Files (*.hdf5 *.h5)
TFT Files (*.dat *.tft)
TIFF Image Files (*.tif *.tiff)
TRUCHAS dataset (*.hdf5 *.h5)
TSurf Files (*.ts_deg83)
UNIC Files (*.h5)
VASP Animation Files (*.out)
VASP CHGCA Files (*.CHG*)
VASP OUT Files (*.OUT*)
VASP POSCAR Files (*.POS*)
VASP Tessellation Files (*.out)
Velodyne Files (*.vld *.rstl
Visit MetaPLOT3D Files (Visit) (*.vp3d)
VizSchema Files (*.h5 *.vsh5)
VPIC Files (*.vpc)
VRML 2 Files (*.wrl *.vrml)
VTK Hierarchical Box Data Files (*.vthb)
VTK ImageData Files (*.vti *.vti.series)
VTK ImageData Files (partitioned) (*.pvti)
VTK MultiBlock Data Files (*.vtm *.vtmb)
VTK Particle Files (*.particles)
VTK Partitioned Dataset Collection Files (*.vtpc)
VTK Partitioned Dataset Files (*.vtpd *.vtpd.series)
VTK PolyData Files (*.vtp *.vtp.series)
VTK PolyData Files (partitioned) (*.pvtp)
VTK RectilinearGrid Files (*.vtr *.vtr.series)
VTK RectilinearGrid Files (partitioned) (*.pvtr)
VTK StructuredGrid Files (*.vts *.vts.series)
VTK StructuredGrid Files (partitioned) (*.pvts)
VTK Table (partitioned) (*.pvtt *.pvtt.series)
VTK Table Files (*.vtt *.vtt.series)
VTK UnstructuredGrid Files (*.vtu *.vtu.series)
VTK UnstructuredGrid Files (partitioned) (*.pvtu)
VTX reader: ADIOS2 BP3 File (*.bp)
VTX reader: ADIOS2 BP4 Directory (*.bp *.bp4)
Wavefront OBJ Files (*.obj)
WindBlade Data (*.wind)
Xdmf Reader (*.xmf *.xdmf *.xmf2 *.xdmf2)
Xdmf3 Reader (*.xmf *.xdmf *.xmf3 *.xdmf3)
Xdmf3 Reader (Top Level Partition) (*.xmf* *.xdmf)
Xmdv Files (*.okc)
XMol Molecule Files (*.xyz)
XYZ Files (*.xyz)

# Example: reading raw (binary) data

Show $f(x,y,z) = (1-z)\left[(1-y)\sin(\pi x) + y\sin^2(2\pi x)\right]$
$+z\left[(1-x)\sin(\pi y) + x\sin^2(2\pi y)\right]$ in $x, y, z \in [0,1]$ sampled at $16^3$

1. File: `data/simpleData.raw` – load it with IMAGE READER
2. Describe the dataset in properties:
   - Data Scalar Type = float
   - Data Byte Order = Little Endian
   - File Dimensionality = 3
   - Data Extent: 0 to 15 in each dimension (1 to 16 won't load data correctly)
   - Scalar Array Name = density
3. Try different views: Outline, Points, Wireframe, Volume
4. Depending on the view, can edit the colour map
5. Try saving data as ParaView data type (`*.pvd`), deleting the object, and reading back from `*.pvd` – this file now contains a full description of the dataset



# VTK = Visualization Toolkit

- Open-source library for 3D computer graphics, image processing and visualization
- Bindings to C++, Tcl, Java, Python, and partial port to JavaScript (started in 2016)
- ParaView is based on VTK ⇒ supports all standard file formats supported by VTK
  `https://docs.vtk.org/en/latest/supported_data_formats.html`
- VTK's own file formats `https://docs.vtk.org/en/latest/vtk_file_formats`
  1. legacy serial formats (*.vtk): ASCII header lines + ASCII/binary data
  2. XML formats (extension depends on VTK data type): XML tags + ASCII/binary/compressed data
     - newer, much preferred to legacy VTK
     - supports parallel file I/O, compression, portable binary encoding (big/little endian byte order), random access, etc.
  3. VTKHDF formats (in development since 2022)
     - using HDF5 for actual storage
     - better serial and parallel I/O performance
     - supports hierarchical data structures
     - eventually will replace other VTK formats

# 1D/2D/3D data in VTK: 6 major discretization types

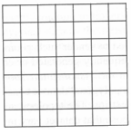
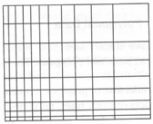
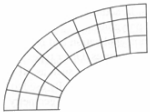

(a) Image Data

Image Data/Structured Points: `*.vti`, points on a regular rectangular lattice, scalars or vectors at each point


(b) Rectilinear Grid

Rectilinear Grid: `*.vtr`, same as Image Data, but spacing between points may vary, need to provide steps along the coordinate axes, not coordinates of each point


(c) Structured Grid

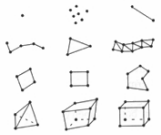
Structured Grid: `*.vts`, regular topology and irregular geometry, need to indicate coordinates of each point


(d) Unstructured Points

Particles/Unstructured Points: `*.particles`


(e) Polygonal Data

Polygonal Data: `*.vtp`, unstructured topology and geometry, point coordinates, 2D cells only (i.e. no polyhedra), suited for maps


(f) Unstructured Grid

Unstructured Grid: `*.vtu`, irregular in both topology and geometry, point coordinates, 2D/3D cells, suited for finite element analysis, structural design

# VTK 3D data: dataset attributes

A VTK file can store a number of datasets, each could be of one of the following types:

- Scalars: single valued, e.g. density, temperature, pressure
- Vectors: magnitude and direction, e.g. velocity
- Normals: direction vectors ($|\mathbf{n}| = 1$) used for shading
- LookupTable: each entry in the lookup table is a red-green-blue-alpha array (alpha is opacity: alpha=0 is transparent); if the file format is ASCII, the lookup table values must be float values in the range [0,1]
- TextureCoordinates: used for texture mapping
- Tensors: $3 \times 3$ real-valued symmetric tensors, e.g. stress tensor
- FieldData: array of data arrays

# Example: reading legacy VTK

Caution: Storing large datasets in ASCII is generally not a good idea – here, we check few small, text-based VTK files for instructional purposes

1. File: `data/volume.vtk`
   - simple example (Structured Points): $3 \times 4 \times 6$ dataset, one scalar field, one vector field
2. File: `data/density.vtk`
   - another simple example (Structured Grid): $2 \times 2 \times 2$ dataset, one scalar field
3. File: `data/cube.vtk`
   - more complex example (Polygonal Data): cube represented by six polygonal faces. A single-component scalar, normals, and field data are defined on all six faces (CELL_DATA). There are scalar data associated with the eight vertices (POINT_DATA). A lookup table of eight colours, associated with the point scalars, is also defined.

# Exercise: create your own 3D legacy VTK dataset

- Visualize a 3D "cylinder" function

$$f(x, y, z) = e^{-|r-0.4|}$$

where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$,
inside a unit cube ($x, y, z \in [0, 1]$)

➠ reproduce approximately the view on the right

- ASCII data in `data/cylinder.dat` (discretized on a $30^3$ Cartesian mesh)
- Add an appropriate header to create a VTK file using `data/volume.vtk` as template

# Writing XML VTK from C++ (and C, and Fortran, ...)

Let's look at larger datasets (tens of MB and up) – we should store them as binary

- A good option is to use XML VTK format with binary data and XML metadata, calling VTK library functions from C++ / Java / Python to write data
- Here is an example: `codes/SGrid.cpp` and `codes/Makefile`, generates the file `data/halfCylinder.vts`

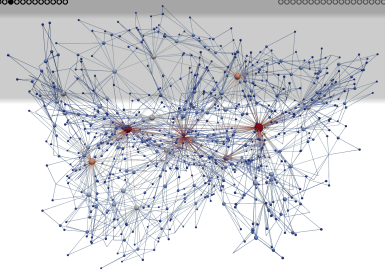
  This example shows how to create a Structured Grid, set grid coordinates, fill the grid with a scalar and a vector, and write it in XML VTK to a `*.vts` file.
- To run it, you need the VTK C++ library installed (either standalone or pulled from ParaView); check `codes/Makefile` to see the required library files

```
cd codes
make SGrid
(on Linux: export LD_LIBRARY_PATH=/path/to/vtk/lib:$LD_LIBRARY_PATH)
(on a Mac: export DYLD_LIBRARY_PATH=$HOME/Documents/local/vtk/lib)
./SGrid
```

- Many more examples included with the VTK source code or at `https://examples.vtk.org/site/Cxx`

# Writing XML VTK from Python

(And creating 3D graphs)



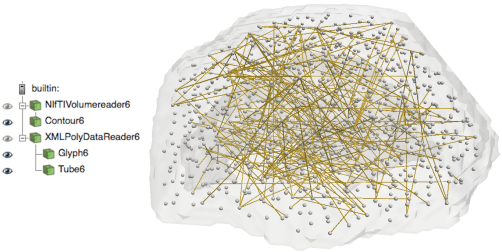
```
$ pip install vtk networkx [scipy]
$ python dgm.py 7
```

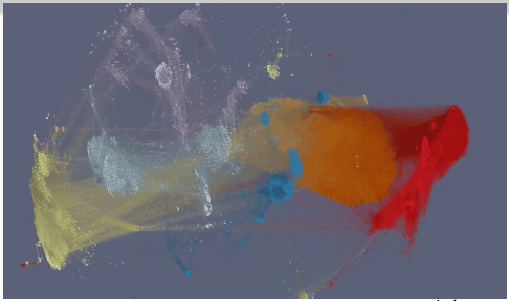
Check out `codes/{writeNodesEdges,dgm}.py`

```python
def writeObjects(nodeCoords,
                 edges = [],
                 scalar = [], name = '', power = 1,
                 scalar2 = [], name2 = '', power2 = 1,
                 nodeLabel = [],
                 method = 'vtkPolyData',
                 fileout = 'test'):
    """
    Store points and/or graphs as vtkPolyData or vtkUnstructuredGrid.
    Required argument:
    - nodeCoords is a list of node coordinates in the format [x,y,z]
    Optional arguments:
    - edges is a list of edges in the format [nodeID1,nodeID2]
    - scalar/scalar2 is the list of scalars for each node
    - name/name2 is the scalar's name
    - power/power2 = 1 for r-scalars, 0.333 for V-scalars
    - nodeLabel is a list of node labels
    - method = 'vtkPolyData' or 'vtkUnstructuredGrid'
    - fileout is the output file name (will be given .vtp or .vtu extension)
    """
```

```python
import networkx as nx
from writeNodesEdges import writeObjects
import sys

if len(sys.argv) == 1:
    print("Usage: python dgm.py <generationNumber>")
    sys.exit(1)

generation = int(sys.argv[1])
H = nx.dorogovtsev_goltsev_mendes_graph(generation)
pos = nx.spring_layout(H, dim=3)
xyz = [list(pos[i]) for i in pos] # list of [x,y,z]

print(nx.number_of_nodes(H), 'nodes and',
      nx.number_of_edges(H), 'edges')
degree = [d for i,d in H.degree(H.nodes())]
writeObjects(xyz, edges=H.edges(), scalar=degree,
        name='degree', power=0.333, fileout='network')
```

# Think of ParaView as a GUI front end to VTK classes



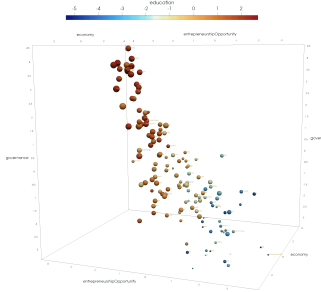



hidden/mutOnCtOrbits.mp4 on presenter's laptop

```
vtkPoints *points = vtkPoints::New();
for (i=0; i<1028; i++) points->InsertNextPoint(x[i], y[i], z[i]);
vtkCellArray *lines = vtkCellArray::New();
for (j=0; j<degree; j++) { // line from node to adjacent[j]
  lines->InsertNextCell(2);
  lines->InsertCellPoint(node);
  lines->InsertCellPoint(adjacent[j]); }
vtkPolyData* polyData = vtkPolyData::New();
polyData->SetPoints(points);  polyData->SetLines(lines);
vtkSmartPointer<vtkXMLPolyDataWriter> writer = vtkSmartPointer<vtkXMLPolyDataWriter>::New();
writer->SetFileName("output.vtp");  writer->SetInputData(polyData);
writer->Write();
```

# Demo: prosperity index – 3D scatter plot showing 5 attributes

- Data from the Legatum 2015 Prosperity Index `http://www.prosperity.com/#!/ranking` ⇒ saved in `legatum2015.csv`: 8 rankings for each country
- Complete and run the code `countries.py` – writes five attributes into `countries.vtp`
- Load `countries.vtp`, apply Glyph filter with Glyph Type = Sphere, Glyph Mode = All Points
- 3D position by (economy, entrepreneurshipOpportunity, governance)
- Colour by education, size by safetySecurity, turn on Axes Grid
- Optionally turn on labels for countries
  - press V to bring up Find Data dialogue
  - Data Producer = countries.vtp with ID $\geq 0$ (all points) and press Find Data
  - make `countries.vtp` visible in the pipeline browser
  - check Point Labels -> tag to display the label (and not another variable)
  - click on the gear icon (Edit Label Properties) and set opacity=0 and adjust the Point Label Font size
- On presenter's laptop:
  `paraview --state=countries.pvsm`




```
import pandas as pd
from writeNodesEdges import writeObjects
data = pd.read_csv('legatum2015.csv', sep=',')

>>> (fill in this part) <<<

writeObjects(xyz, fileout='countries',
             nodeLabel = list(data['country']),
             scalar = list(data['education']),
             name = 'education',
             scalar2 = list(data['safetySecurity']+6),
             name2 = 'safetySecurity', power2 = 1)
```

# Another option for writing XML VTK from Python

PyEVTK library https://github.com/paulo-herrera/PyEVTK

```
$ pip install pyevtk
```

- Many examples in https://github.com/paulo-herrera/PyEVTK/tree/master/evtk/examples

```
from pyevtk.hl import imageToVTK
from numpy import zeros
n = 30
data = zeros((n,n,n), dtype=float)
for i in range(n):
    x = ((i+0.5)/float(n)*2.-1.)*1.2
    for j in range(n):
        y = ((j+0.5)/float(n)*2.-1.)*1.2
        for k in range(n):
            z = ((k+0.5)/float(n)*2.-1.)*1.2
            data[i][j][k] = ((x*x+y*y-0.64)**2 + (z*z-1.)**2) * \
                            ((y*y+z*z-0.64)**2 + (x*x-1.)**2) * \
                            ((z*z+x*x-0.64)**2 + (y*y-1.)**2)
imageToVTK("decoCube", pointData={"scalar" : data})
```

# VTKHDF

- Extension `*.vtkhdf`
- In development since 2022; very much a work in progress
- Once complete, will replace all other VTK formats
- Using HDF5 for actual storage ⇒ all the bells and whistles of HDF5, including:
  - better serial and parallel I/O performance
  - support for hierarchical data structures
- Read/write VTKHDF files:
  1. using VTKHDF functions in VTK
     - at the moment, `vtk.vtkHDFWriter()` only supports `vtkPolyData` and `vtkUnstructuredGrid` (not all cell types)
     - write support for `vtkImageData`, `vtkOverlappingAMR`, `vtkMultiBlockDataSet`, `vtkPartitionedDataSetCollection`, `vtkPartitionedDataSet` should be coming soon
  2. using standard HDF5 tools ⇐ the file looks like an HDF5 file with specific formatting
     - e.g. `h5py` and compression with `hdf5plugin`
- On presenter's laptop:

```
paraview ~/training/paraviewWorkshop/vtkhdf/warpingSpheres.vtkhdf
```

# NetCDF and HDF5

- VTK is incredibly versatile format, can describe many different data types
- Very often in science one needs to simply store and visualize multi-dimensional arrays
- Problem: how do you store a $2000^3$ array of real numbers (30GB of data)?
  - ASCII – forget about it
  - raw binary – possible, but many problems
  - VTK – probably an overkill for simple arrays
- Scientific data formats come to rescue, two popular scientific data formats are NetCDF and HDF5
  - binary (of course!)
  - self-descriptive (include metadata)
  - portable (cross-platform): libraries for many OS's, universal datatypes, byte order in a word (little vs. big endian), etc.
  - support parallel I/O (through MPI-IO)
  - support compression

# NetCDF support in ParaView

- NetCDF is supported natively in ParaView
  - `codes/writeNetCDF.cpp` (Fortran version `codes/writeNetCDF.f90`) writes a $100^3$ volume with a doughnut shape at the centre in NetCDF

## C++ example

```
$ icc writeNetCDF.cpp -o writeNetCDF -I/path/to/netcdf/include \
      -L/path/to/netcdf/lib -lnetcdf_c -lnetcdf
$ ./writeNetCDF
```

## F90 example

```
$ ifort writeNetCDF.f90 -o writeNetCDF -I/path/to/netcdf/include \
        -L/path/to/netcdf/lib -lnetcdff -lnetcdf
$ ./writeNetCDF
```

- ParaView understands common NetCDF conventions, e.g., conventions for CF (Climate and Forecast) metadata (`http://cfconventions.org`): 2D or 3D datasets on a sphere, coordinate axes, fill-in values, etc.
  - example 1 on presenter's laptop: 2D dataset `hidden/ice.nc`
  - example 2: snapshot of a 3D dataset `hidden/temp1.png`
  - example 3: more polished 3D visualization `hidden/tempsalt.mp4`

# On the subject of spheres ...

- How about adding topography to our visualization?
- `https://www.earthmodels.org` used to have some good resources on this in the past ...
  - files on presenter's laptop in `~/training/paraviewWorkshop/topography`
- Option 1: load precomputed topography stored as Polygonal Data
  - `ETOPO_10min_Ice.vtp` provides full globe (both land and ocean) at 10 arcmin resolution
  - `ETOPO_10min_Ice_only-land.vtp` provides only land at 10'
- Option 2: map a bitmap to the globe, e.g. this 8192 × 4096 image `texture_land_ocean_ice_8192.png`
  1. create a high-resolution Sphere (from Sources)
  2. apply Texture-Map-to-Sphere filter, make sure to click Apply before you can see Miscellaneous:Texture
     - creates "texture coordinates"
     - we haven't studied filters yet
  3. in filter Properties: under Miscellaneous:Texture use the drop-down menu to load a PNG image, click Apply
  4. in Properties of the filter: uncheck Prevent Seam at the top, check Seamless U, Seamless V, again click Apply
  5. still colouring by Solid Color and viewing as Surface

# HDF5 support in ParaView

- Now also supported via VTKHDF formats
- No native support for standalone HDF5, however, ParaView supports a container format XDMF (eXtensible Data Model and Format) which uses HDF5 for actual data
  - details at `http://www.xdmf.org`
  - XDMF = XML for **light** data + HDF5 for **heavy** data
    - data type (float, integer, etc.), precision, rank, and dimensions completely described in the XML layer (as well as in HDF5)
    - the actual values in HDF5, potentially can be enormous
  - single XML wrapper can reference multiple HDF5 files (e.g. written by each cluster node)
  - don't need HDF5 libraries to perform simple operations
  - C++ API is provided to read/write XDMF data
  - can be used from Python, Tcl, Java, Fortran through C++ calls
  - in Fortran can generate XDMF files with HDF5 calls + plain text for the XML wrapper `http://www.xdmf.org/index.php/Write_from_Fortran`
- Support for several file formats produced by third-party software, which themselves are built on HDF5

# Recap of input file formats

- Raw binary data

1. VTK legacy format (*.vtk) with ASCII data for small datasets
   - Structured Points
   - Structured Grid
   - Polygonal Data
2. VTK XML formats for large datasets: most versatile, use from C++ and Python
   - Structured Grid (*.vts)
   - other formats can be written using the respective class, e.g. vtkPolyData, vtkRectilinearGrid, vtkStructuredGrid, vtkUnstructuredGrid
3. VTKHDF formats write support coming soon

- HDF5 files via XDMF and VTKHDF, **native NetCDF support**
- Hundreds of 3rd-party file formats understood natively by ParaView

# WORKING WITH PARAVIEW: FILTERS

# Filters

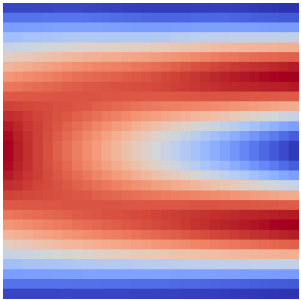
- Many interesting features of a dataset cannot be determined by simply looking at its surface: much of the useful information lies inside, or emerges from a combination of variables or their derivatives
- Sometimes a desired view is not available for a given data type, e.g.
  - a 2D dataset $f(x, y)$ will be displayed as a 2D dataset even in 3D (try loading `data/2d000.vtk`), but we might want to see it in 3D by displaying the elevation $z = f(x, y)$
  - the volumetric view is available only for certain VTK datasets, including ImageData, StructuredPoints and UnstructuredGrid with connectivity
- **Filters** process your data to generate, extract, or derive additional features
- Filter connections form a **visualization pipeline**
- Last time I checked, ParaView provided 146 built-in filters, and you can extend this set by creating custom filters either in Python (using the Programmable Filter) or in C++ as a shared-library ParaView plugin

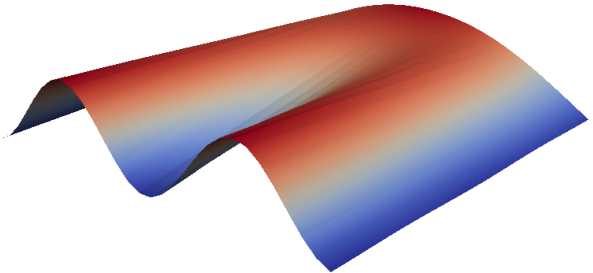
➠ Check out "Filters" in the menu; some are found in the toolbar

➠ Edit your filter's properties in Properties

# Simple filter to visualize a 2D dataset in 3D

- Load the file `data/2d000.vtk` that samples the 2D function $f(x, y) = (1 - y)\sin(\pi x) + y\sin^2(2\pi x)$, where $x, y \in [0, 1]$, on a $30^2$ grid
- Highlight the dataset in the pipeline browser and apply the WarpByScalar filter
- Change to 3D view, edit the offset factor to reproduce the 3D view below



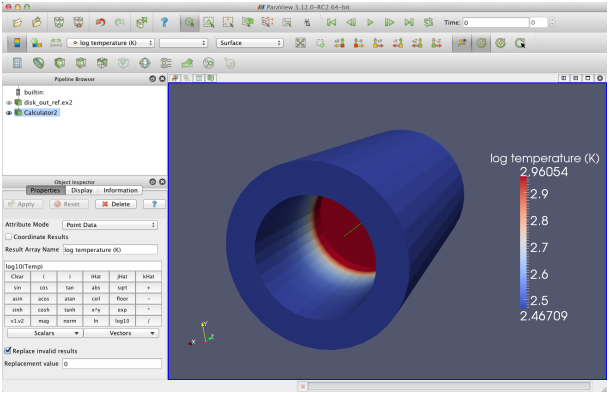


# Find these filters in the toolbar

- **Calculator** evaluates a user-defined expression on a per-point or per-cell basis
- **Contour** extracts user-defined points, isocontours/isosurfaces from a scalar field
- **Clip** removes all geometry on one side of a user-defined plane
- **Slice** intersects the geometry with a plane; the effect is similar to clipping except that all that remains is the geometry where the plane is located
- **Threshold** extracts cells that lie within a specified range of a scalar field
- **Glyph** places a glyph on each point in a mesh; glyphs may be oriented by a vector and scaled by a vector or scalar
- **Stream Tracer** seeds a vector field with points and then traces those seed points through the steady state vector field

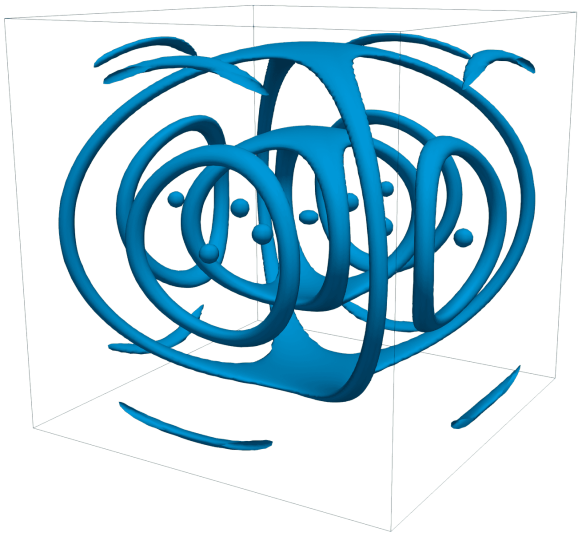
# Calculator

- Load temperature, velocity, pressure from `data/disk_out_ref.ex2`, try visualizing individual variables: Pres, Temp, V
- Click on "Toggle Colour Legend Visibility" to see the temperature range
- Now apply **Calculator** filter to display log10(Temp)
  - also try to create Pres/Temp, mag(V)
  - dropdown menus "Scalars" and "Vectors" will help you enter variables
  - the "?" button is surprisingly useful
- You can change visibility of each object in the pipeline browser by clicking on the eyeball icon next to it



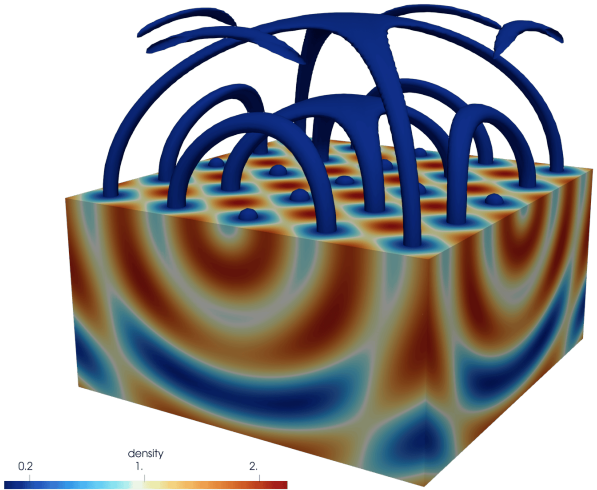
# Exercise: recreate this visualization

- Data source: `data/sineEnvelope.nc`



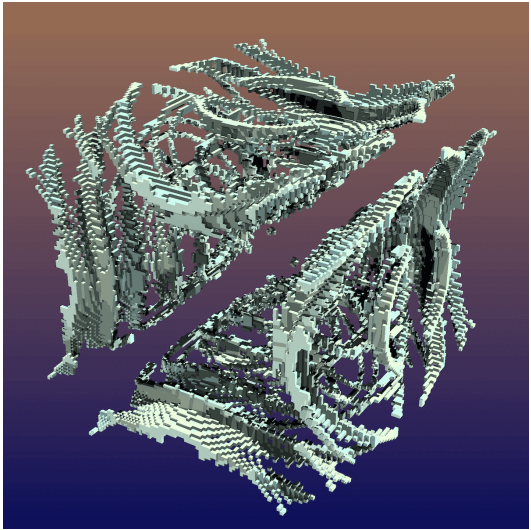
# Exercise: recreate this visualization

- Data source: `data/sineEnvelope.nc`
- More complex visualization pipeline: two filters



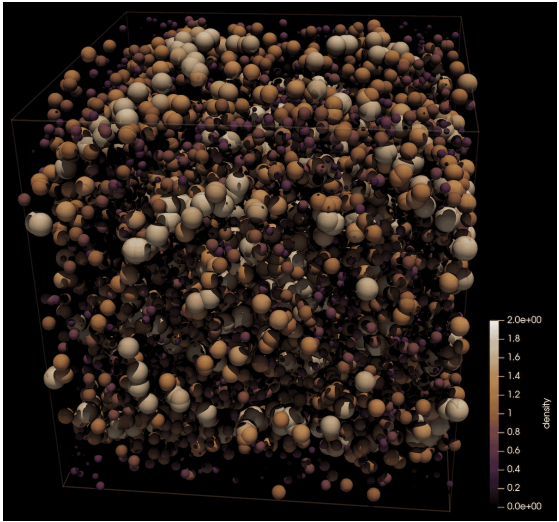
# Exercise: recreate this visualization

- Data source: `data/sineEnvelope.nc`
- Show only points in the range $\rho \in [0.8, 1.0]$
- Enable gradient background with custom colours
- Turn on ray tracing and shadows
- Play with View | Light Inspector



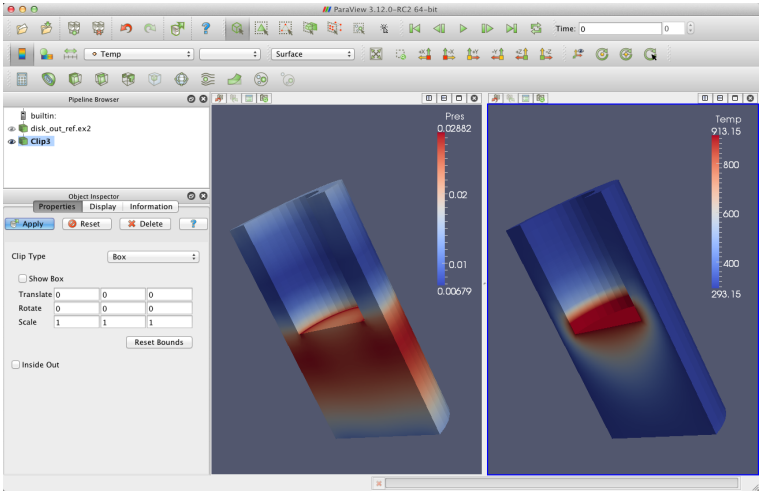
# Exercise: recreate this visualization

- Data source: `data/sineEnvelope.nc`
- Find the right filter
  - do colour and size of the spheres reflect the data?
- Turn on ray tracing and shadows
- Play with the background colour and the colourmap



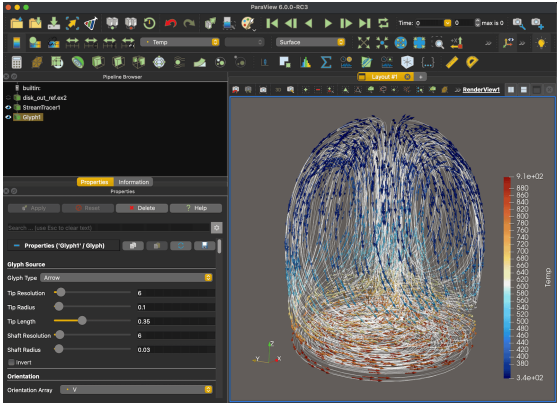
# Exercise: recreate this visualization

- Data source: `data/disk_out_ref.ex2`



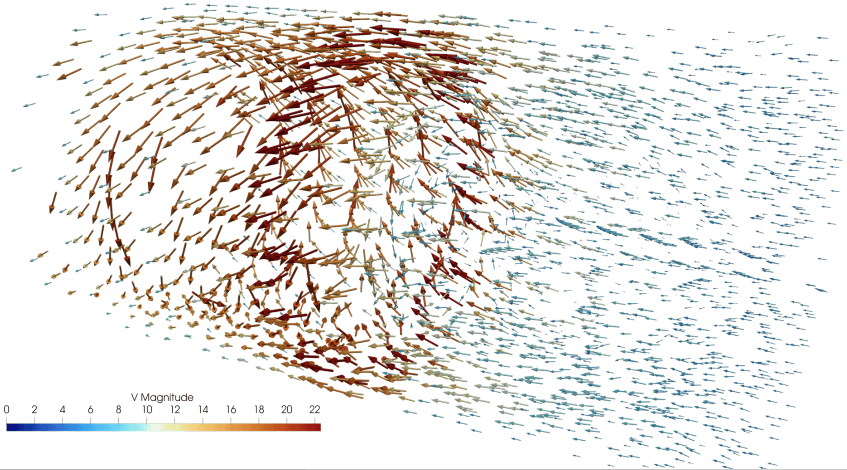
# Vector visualization: streamlines and glyphs

1. Load velocity, Temp from `data/disk_out_ref.ex2`
2. Add the **Stream Tracer** filter, set Seed Type = Point Cloud, Radius = 3, play with Number Of Points, Maximum Streamline Length
3. Optionally add shading and depth cues to the streamlines: Filters → Alphabetical → **Tube**
4. Add glyphs to streamlines to show the orientation and magnitude:
   - select StreamTracer in the pipeline browser
   - add the **Glyph** filter to StreamTracer
   - set Glyph Type = Arrow, Orientation Array = V, Scale Array = None
   - colour glyphs with "Temp"



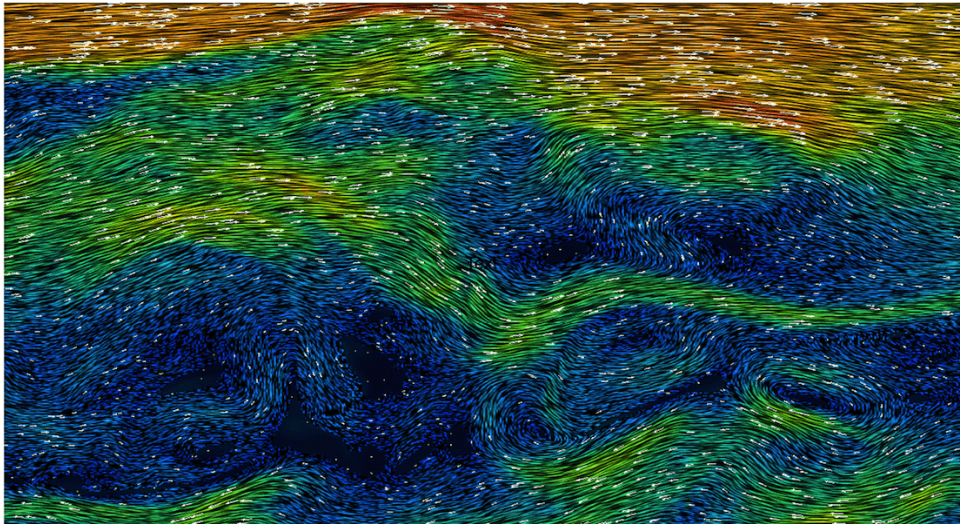
# Exercise: now fill the entire volume with vectors

- Load `data/disk_out_ref.ex2` and display the velocity field as arrows, colouring them by the velocity magnitude

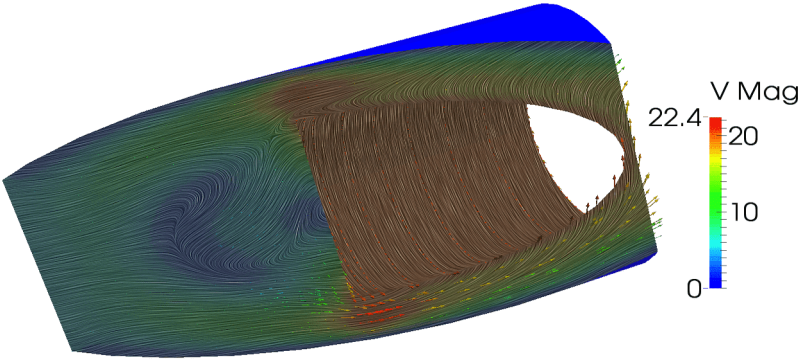


# Line Integral Convolution representation

Details at `http://www.paraview.org/Wiki/ParaView/Line_Integral_Convolution`



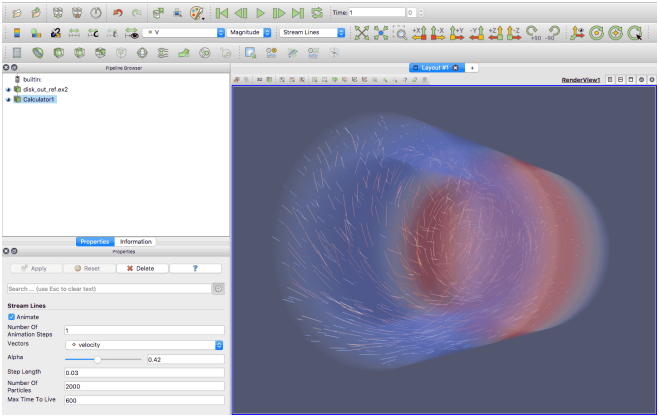
# Line Integral Convolution in ParaView




- From Tools | Manage Plugins load *Surface LIC plugin*
- Load `data/disk_out_ref.ex2` or `data/halfCylinder.vts`
- Apply a filter to see its interior (required step for `data/halfCylinder.vts`)
- Switch to *Surface LIC* representation in the drop-down menu
- In properties make sure to select the velocity variable for Surface LIC
- Play with the number of steps and individual step sizes, adjust colour

# Stream Lines representation (live drawing)

Details at `http://bit.ly/2NFNcvQ`




- Enable StreamLinesRepresentation plugin, then load the dataset
- Use Stream Lines representation, in properties increase Step Length
- Spin your visualization

# 3D data as columns for quick input

- `data/tabulatedPoints.txt` contains 100 random points, with each line storing *x*, *y*, *z*, *scalar*
- Load in ParaView, apply the **Table To Points** filter, making sure to edit the fields (X/Y/Z Columns)
- Apply the **Glyph** filter to view points as spheres, colour them by *scalar*
- No implied topology here!
- You can optionally pass the points through the **Delaunay 3D** filter, followed by **Extract Edges**, followed by **Tube**

---

- `data/tabulatedGrid.txt` contains 1000 points representing a $10^3$ Cartesian mesh, with each line storing *x*, *y*, *z*, *scalar* of a point
- Load in ParaView, apply the **Table To Structured Grid** filter, making sure to edit the fields (Whole Extent 0 to 9 in each dimension, X/Y/Z Columns)
- The data must have some implied topology for this filter to work!

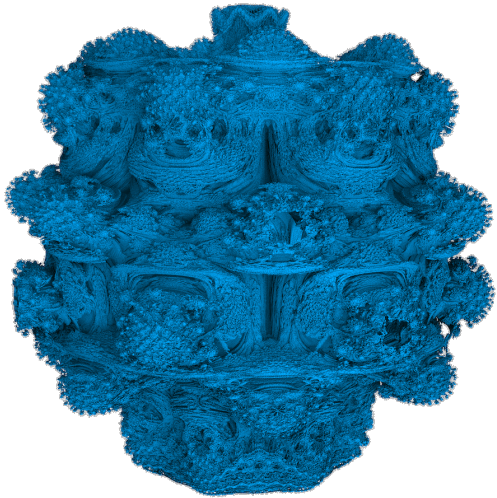
Not recommended for large datasets: waste of disk storage and bandwidth!

➢ tabulatedPoints.txt is 6231 bytes vs. 1600 bytes in single-precision binary

➢ tabulatedGrid.txt is 20,013 bytes vs. 4000 bytes in single-precision binary

# Exercise: something more complex

- Power-8 Mandelbulb is a 3D fractal
- Data file at two resolutions:
  - `mandelbulb300.nc` at $300^3$ for lower-memory machines
  - with 16GB+ memory, you can try loading `mandelbulb800.nc` at $800^3$
- Let's do this:
  1. check the grid size and the file size: do these agree?
  2. try slices, volumetric rendering, isosurfaces
  3. try to recreate the picture on the right (at a lower resolution): pay attention to the lights and shadows



# Exercise: 3D optimization

`data/stvol.nc` contains a discretized scaled variant of the 3D Styblinski-Tang function inside a unit cube ($x_i \in [0, 1]$), built with `codes/optimization.c`

$$f(x_1, x_2, x_3) = \frac{1}{2} \sum_{i=1}^{3} (\xi_i^4 - 16\xi_i^2 + 5\xi_i), \text{ where } \xi_i \equiv 8(x_i - 0.5)$$

Let's answer the following questions:

1. What is the size of the grid? Does it agree with the size of the file?
2. Find the approximate location of the *global minimum* of $f(x_1, x_2, x_3)$ using visual techniques (slices, isosurfaces, thresholds, volume renderings, etc.)

- Note: you can find the exact coordinates of the global minimum by using Filters -> Statistics -> **Descriptive Statistics**, clicking Apply, and sorting points in order of increasing f(x,y,z)

# Word of caution

- Many visualization filters (e.g. Clip, Slice) transform stuctured grid data into unstructured data
- For some filters, memory usage and CPU load can grow very quickly, e.g. clipping a $400^3$ dataset to 150 million cells can take $\sim$ 1 hour on a single CPU $\Rightarrow$ might want to run in distributed mode

# Python Calculator filter

https://www.paraview.org/Wiki/Python_Calculator

- To calculate vorticity, pick a fector field, enter `curl(V)`, call it vorticity
- Supported functions on arrays: abs(), cross(), curl(), det(), dot(), eigenvalue(), eigenvector(), global_mean(), global_max(), global_min(), gradient(), inverse(), laplacian(), ln(), log10(), max(), min(), mean(), mag(), norm(), strain(), trace(), vorticity()

# More filter functionality

- Can merge several existing filters into a *custom filter*
  `https://cfdengine.com/newsletter/104` (mini-tutorial)
  **Tools | Create Custom Filter** and edit its input, output and properties

- Can script filters in Python
  `http://www.paraview.org/Wiki/Python_Programmable_Filter`
  **Filters | Alphabetical | Programmable Filter**
  (more on general scripting in Part 2)

- Can write new filters as plugins, compile them as shared libraries with the same version of ParaView they are expected to be deployed with
  `https://www.paraview.org/paraview-docs/nightly/cxx/PluginHowto.html`