Visualisation scientifique avec ParaView
1re partie

Pier-Luc St-Onge
et Alex Razoumov

Digital Research Alliance of Canada

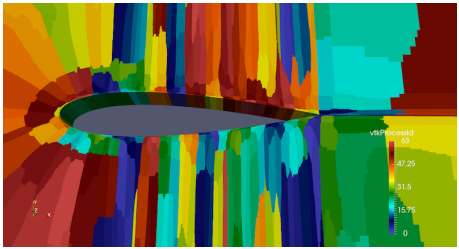
✔ Diapositives, données et codes : `https://folio.vastcloud.org/introparaview`

- le lien "main ZIP file" permet de télécharger `paraview.zip` (∼30 Mo)
- une fois le contenu extrait, on obtient `codes/`, `data/` et `slides1.pdf`

✔ Installez ParaView 6.0.x sur votre ordinateur à partir de `http://www.paraview.org/download`

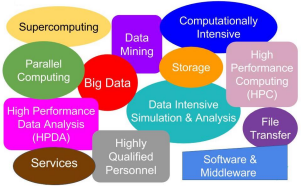
Plan de l'atelier

- Introduction à la visualisation scientifique : généralités, outils, graphiques vs visualisation multi-dimensionnelle (∼30 min.)
- Survol des outils génériques actuels de visualisation multi-dimensionnelle
- Architecture de ParaView
- Importer des données dans ParaView : données binaires, types de données VTK, NetCDF/HDF5, OpenFOAM
- Flux de travail de base : utilisation de filtres, création de pipelines et de champs vectoriels



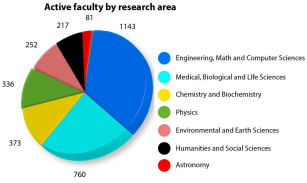
Qui sommes-nous?

https://docs.alliancecan.ca

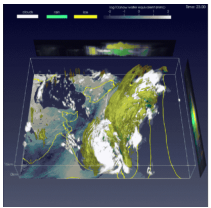


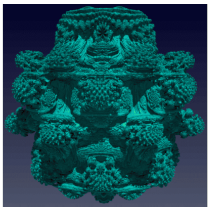
L'Alliance fournit gratuitement aux chercheuses et aux chercheurs une infrastructure et des services de calcul informatique de pointe (CIP) :

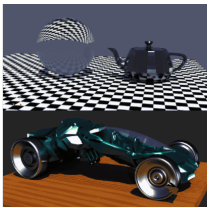
- SUPERORDINATEURS, infonuagique
- stockage, gestion et transfert de données
- soutien technique, formation, visualisation



- Équipe nationale de visualisation : `https://ccvis.netlify.app`
- Compétition Visualize This depuis 2016, compétition IEEE SciVis 2021 `https://ccvis.netlify.app/contests`







Visualisation scientifique (données définies dans l'espace)

- C'est le processus permettant l'analyse de données sous une FORME VISUELLE
  - Il est plus facile de comprendre des images que de grands ensembles de nombres.
  - Cela permet d'explorer interactivement des données, de déboguer une analyse et de communiquer avec les pairs.

CHAMP DE RECHERCHE	TYPE DE VISUALISATION
mécanique des fluides numérique	écoulements 2D/3D, densité, température, traceurs
climat, météorologie, océanographie	fluides dynamiques, nuages, chimie, etc.
astrophysique	fluides 2D/3D, données particulaires, champ de rayonnement $\leq$6D, champs magnétiques, champs gravitationnels
chimie quantique	fonctions d'onde
dynamique moléculaire (physique, chimie, biologie)	données particulaires/moléculaires
imagerie médicale	IRM, CT scans, ultrason
systèmes d'information géographique	altitude, rivières, villes, routes, strates, etc.
bioinformatique	réseaux, arbres, séquences
sciences humaines et sociales	données abstraites, ou l'un des types ci-dessus

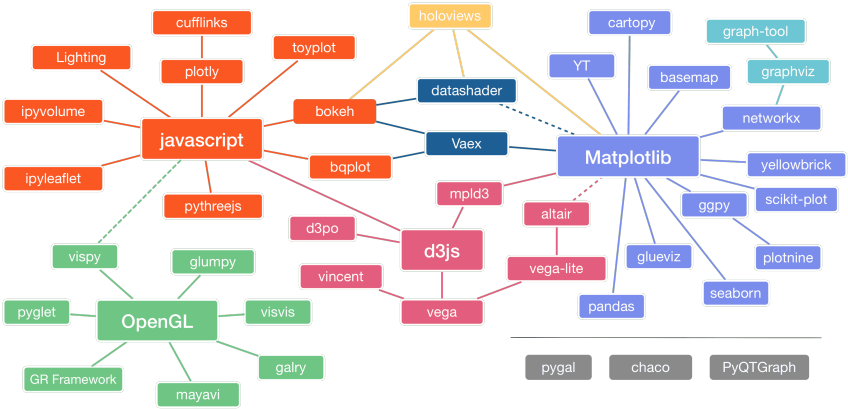
Graphiques 1D/2D vs visualisation multi-dimensionnelle

- **Graphiques 1D/2D** : données typiquement tabulaires en 1D
  - `gnuplot` et `pgplot` pour des graphiques simples
  - bibliothèques fortement recommandées en Python :
    - `https://matplotlib.org` (graphiques statiques)
    - `https://seaborn.pydata.org` (idéal pour données statistiques)
    - `https://plotly.com/python` (graphiques interactifs en HTML5)
    - `https://bokeh.org` (graphiques interactifs en HTML5)
    - `https://altair-viz.github.io` (graphiques interactifs, bibliothèque déclarative utilisant la grammaire de Vega-Lite)
    - `https://plotnine.org` (grammaire des graphiques pour Python)
    - `https://holoviews.org`
  - bibliothèque recommandée en R : `https://ggplot2.tidyverse.org` (basée sur la grammaire des graphiques)
- **Visualisation 2D/3D**
  - afficher des ensembles de données multi-dimensionnelles, c'est-à-dire des données sur des grilles structurées (uniformes et multi-résolution) ou non structurées (selon une certaine topologie en 2D/3D)
  - le rendu est souvent lourd sur CPU et/ou GPU

Graphiques 1D/2D vs visualisation multi-dimensionnelle

- Autant que possible, évitez les outils propriétaires, sauf s'il y a un réel avantage (probablement pas)
  - grande quantité d'$$ à l'achat initial
  - la license peut imposer des limitations sur l'endroit où l'outil peut être utilisé, sur quel type de machine ou de plateforme, etc.
  - la communauté d'utilisateurs est généralement plus petite que pour les outils à source ouvert, et il est plus difficile d'obtenir de l'aide
  - une fois que vous commencez à accumuler des scripts, vous vous retrouvez contraint d'utiliser ces outils et, par conséquent, de payer régulièrement de l'$$

Outils de visualisation Python selon le moteur de rendu

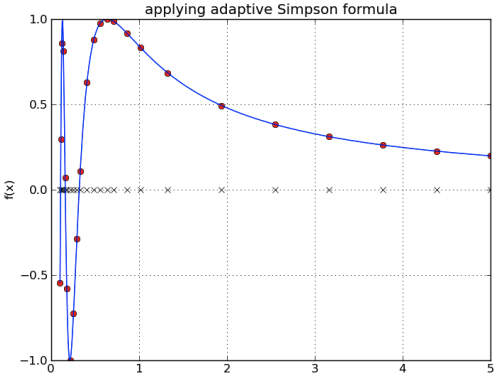


`https://github.com/rougier/python-visualization-landscape`, par Nicolas Rougier

Cette figure est fortement influencée par la création de graphiques 1D/2D ... montrant qu'une poignée d'outils 3D. Voir également `https://pyviz.org` pour une liste à jour des différents outils

Exemple Matplotlib : graphique 1D

Intégration adaptative de Simpson



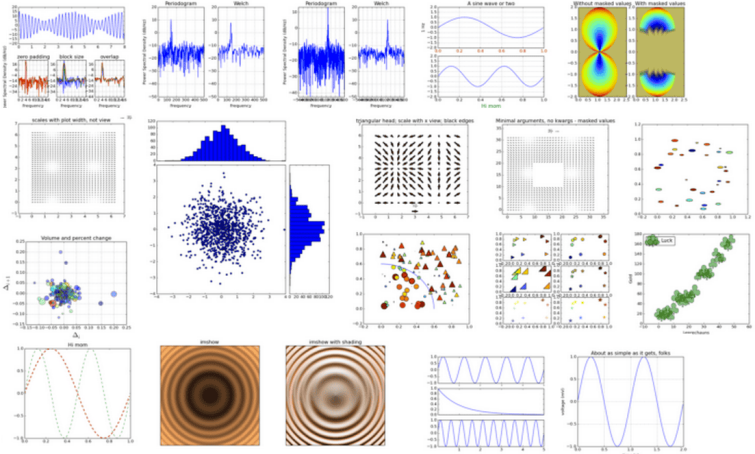
- Une fonction Python `simpsonAdaptive(function, a, b, tolerance)` calcule des coordonnées `(x, f(x))` et génère le graphique ci-dessus
- Code : `codes/adaptive.py`

Exemple Python Imaging Library (PIL) : graphique 2D

Détection de contours par différenciation numérique

- À partir d'une image PNG, calculer le gradient du canal bleu et représenter la norme des vecteurs selon une échelle de blanc à noir
- Code : `codes/fuji.py`

Galerie Matplotlib contenant des centaines d'exemples



- `https://matplotlib.org/stable/gallery` – cliquez un graphique pour obtenir son code source
- `https://github.com/rougier/matplotlib-tutorial` est vraiment un excellent tutoriel

Galerie Bokeh

- Projet à code source ouvert de Continuum Analytics
  `https://docs.bokeh.org/en/latest/docs/gallery.html`
- Produit des visualisations dynamiques en HTML5 (objet JSON) pour le Web

Galerie Plotly

- Projet à code source ouvert de Plot.ly `https://plot.ly/python`
- Produit des visualisations dynamiques en HTML5 pour le Web; peut aussi générer des formats statiques
- API pour Python (avec/sans Jupyter), R, JavaScript, MATLAB
- Tutoriel : `https://wgpages.netlify.app/plotly`

Outils de visualisation multi-dimensionnelle

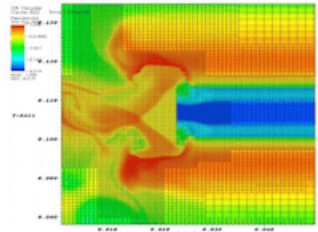
- Code source ouvert + multi-plateforme + usage général + doit avoir les caractéristiques suivantes :
  - pouvoir visualiser des champs scalaires et vectoriels
  - prise en charge : maillages structurés et non structurés en 2D et 3D, données particulaires, données polygonales, topologies irrégulières, ensembles de données AMR / multi-résolution
  - capacité à traiter de très grands ensembles de données (de Go à To), jusqu'à $10^{\sim 12}$ éléments
  - pouvoir utiliser la puissance des grappes de calcul (jusqu'à $10^3 - 10^5$ coeurs CPU par tâche)
  - manipulation interactive via l'interface graphique
  - automatisation via des scripts Python
  - prise en charge des formats de données les plus courants, lecture et écriture en parallèle

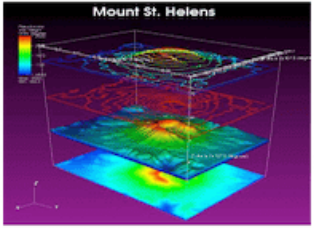
1. VisIt (dernière version : 3.4.2)
2. ParaView (dernière version : 6.0.1)

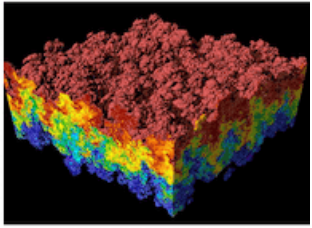
VisIt

https://visit-dav.github.io/visit-website

- Développé par l'Advanced Simulation and Computing Initiative (ASCI) du Department of Energy (DOE) pour visualiser les résultats de simulations à l'échelle du téraoctet
- Disponible sous forme de code source et de binaire pour Linux/Mac/Windows
- Plus de 80 fonctionnalités de visualisation (contour, maillage, coupe, volume, molécule, ...)
- Prend en charge plus de 110 formats de fichiers différents ; API pour C++, Python et Java
- Interactif et programmable en Python ; intégration complète avec la bibliothèque VTK
- Utilise MPI pour le parallélisme à mémoire distribuée sur les grappes de calcul





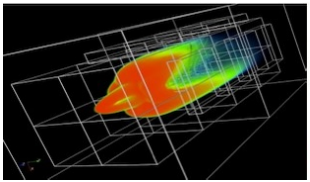


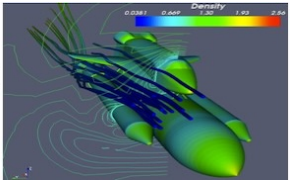
Lawrence Livermore National Laboratory

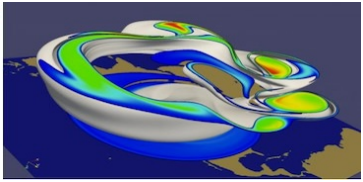
ParaView

`http://www.paraview.org` et `https://github.com/Kitware/ParaView`

- Collaboration lancée en 2000 entre Los Alamos NL et Kitware Inc., rejointe plus tard par Sandia NL et d'autres partenaires ; première version publique en 2002
- Disponible sous forme de code source et de binaire pour Linux/Mac/Windows
- Pour visualiser des données très volumineuses sur des machines à mémoire distribuée
- Utilise MPI pour le parallélisme à mémoire distribuée sur les grappes de calcul
- Interactif et programmable en Python ; client-serveur pour visualisations à distance
- ParaView est basé sur VTK (Visualization Toolkit)
  - pas le seul moteur de rendu scientifique basé sur VTK, par ex. VisIt, MayaVi (Python + numpy + scipy + VTK), et bien sûr, plusieurs outils de Kitware, outre ParaView, sont basés sur VTK.
  - VTK s'utilise en C++, Python et maintenant en JavaScript comme moteur de rendu







Pourquoi des ateliers sur ParaView?

- En ~2010, notre expert Alex a dû faire un choix
  - les binaires de ParaView et de VisIt étaient largement disponibles et en plein développement
  - les deux permettaient la visualisation client-serveur à distance et offraient une excellente scalabilité
  - cependant, ParaView et VisIt ont une interface très différente
- ParaView est fortement intégré avec VTK (mêmes développeurs), peut lire 130 formats en entrée
- Un certain nombre de projets complémentaires
  - **ParaViewWeb** est une bibliothèque JavaScript permettant de développer des applications Web communiquant avec un serveur ParaView distant ; elle peut reproduire l'intégralité du fonctionnement de ParaView dans un navigateur Web (WebGL + traitement distant)
  - **vtk.js** est une bibliothèque de rendu scientifique pour le Web (WebGL autonome)
  - **ParaView Glance** est une application Web permettant la visualisation scientifique 3D directement dans le navigateur.
  - **Catalyst** est une bibliothèque de *visualisation in situ* utilisable à même votre code de simulation ; intéraction via des scripts ParaView
  - **ParaView Cinema** permet la visualisation interactive d'images pré-rendues (rotation, panoramique, zoom, activation/désactivation de variables)
- Depuis, nous utilisons et enseignons aussi VisIt

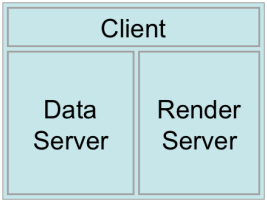
Architecture et interface graphique de ParaView

Architecture parallèle distribuée de ParaView

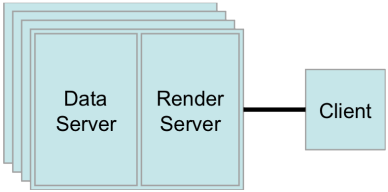
ParaView comprend trois composants logiques : ces unités peuvent être intégrées à la même application sur le même ordinateur, mais peuvent également s'exécuter sur des machines différentes :

- **Serveur de données** – L'unité responsable de la lecture, du filtrage et de l'écriture des données. Tous les objets du pipeline visibles dans l'explorateur de pipelines sont contenus dans le serveur de données. Ce serveur peut être exécuté en parallèle.
- **Serveur de rendu** – L'unité responsable du rendu. Le serveur de rendu peut également être parallèle.
- **Client** – L'unité responsable de la visualisation. Le client contrôle la création, l'exécution et la destruction des objets sur les serveurs, mais ne contient aucune donnée, ce qui permet aux serveurs de prendre toute la charge sans saturer le client. Si une interface graphique est présente, elle se trouve également du côté client. Le client est toujours une application séquentielle.

Deux principaux modes d'utilisation



Mode autonome : les calculs et l'interface utilisateur s'exécutent sur la même machine



Mode client-serveur : `pvserver` sur un serveur multicœur ou une grappe de calcul

Avantages du rendu client-serveur à distance

- Le mode autonome a ses limites : bande-passante en lecture des données, mémoire système et puissance CPU / GPU d'une station de travail
- Par exemple, un ordinateur doté de 48 Go de mémoire pourrait traiter des modèles jusqu'à 2048^3 valeurs à virgule flottante à simple précision sur des grilles structurées stockées localement (faible latence)
- Un tel ordinateur ne pourrait traiter des données plus larges ou de plus haute résolution, des grilles plus complexes ou des données nécessitant des filtres complexes
- Un problème typique qui ne tiendrait pas sur un ordinateur de 48 Go et dont la lecture via SSHFS serait trop lente : simulation de l'écoulement d'air autour d'une aile sur une *grille non structurée* (*.vtu) avec 246×10^6 cellules ($\sim 627^3$), une seule variable prend déjà 25 Go – par contre, ce serait possible de visualiser interactivement à distance sur un serveur de 64 cœurs avec un processus `pvserver` prenant environ 120 Go de mémoire

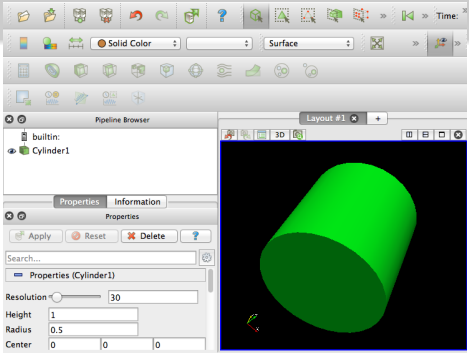
Démarrer ParaView

Aujourd'hui, nous allons tout faire avec le mode autonome de ParaView. Pour ce faire, démarrez ParaView sur votre ordinateur :

- Linux/Unix : entrez `paraview` à la ligne de commande
- MacOS : cliquez sur *ParaView* dans les Applications
- Windows : sélectionnez *ParaView* dans le menu Démarrer

L'interface graphique de ParaView devrait apparaître. En arrière-plan, un processus `pvserver` est démarré automatiquement pour vous.

Interface graphique



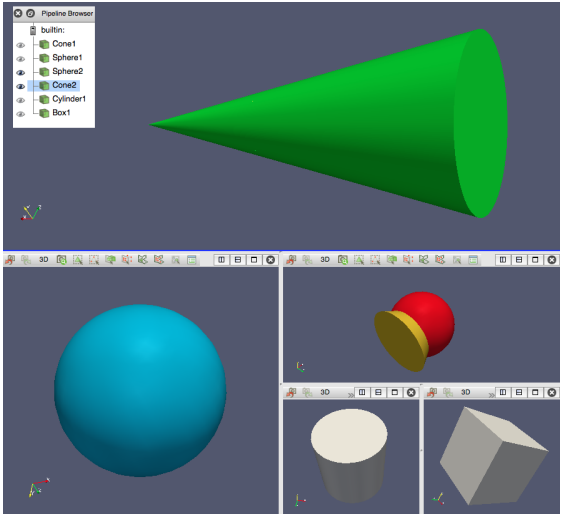
- **Pipeline Browser** : lecteurs de données, filtres de données, peut activer ou désactiver la visibilité de chaque objet
- **Inspecteur d'objets** : voir et modifier les paramètres de l'objet sélectionné dans le pipeline (via les onglets *Properties* et *Information*)
- **Fenêtre de visualisation** : affiche le résultat

1. Dans la barre d'outils, trouvez les boutons : "Connect", "Disconnect", "Toggle Color Legend Visibility", "Edit Color Map", "Rescale to Data Range"
2. Chargez un ensemble de données prédéfini : menu *Sources* → *Geometric Shapes* → *Cylinder*
3. Essayez de bouger le cylindre en appuyant sur le bouton gauche de la souris ; essayez aussi avec le bouton droit et le bouton central
4. Familiarisez-vous avec le menu contextuel ; changez la *Representation* du cylindre (par exemple, de *Surface* à *Wireframe*) ou changez sa couleur via *Edit Color*

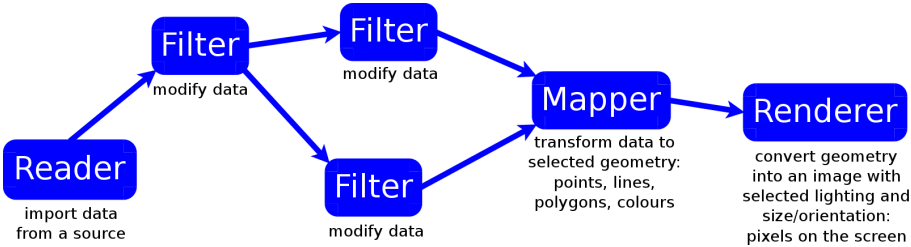
Fenêtres de visualisation

Reproduisez cette image :

- Ajoutez un objet du menu *Sources* (*cone*, *sphere*, *cylinder* ou *box*) et éditez ses propriétés
- Utilisez un des boutons en haut à droite de la fenêtre de visualisation pour diviser la vue
- Répétez avec l'objet ou les objets suivants
- Liez deux vues : bouton droit de souris sur une vue, sélectionner *Link Camera* et cliquer sur une autre vue



Flux de données dans VTK



- Les données passent à travers un Mapper qui sait comment les dessiner dans la scène de rendu via un Actor VTK
  - `mapper.setInputConnection(object.getOutputPort())`
- Un Actor est un objet OpenGL, soit un élément rendu
  - Il prend ses données du Mapper : `actor.setMapper(mapper)`
  - Et les envoie au Renderer : `renderer.addActor(actor)`
- Le Renderer peut contenir plusieurs Actors
- RendererWindow (à l'écran) peut contenir plusieurs Renderer

IMPORTER DES DONNÉES DANS PARAVIEW

Sources de données

- On peut générer des données avec un objet *Source*
- On peut aussi lire des données à partir d'un fichier

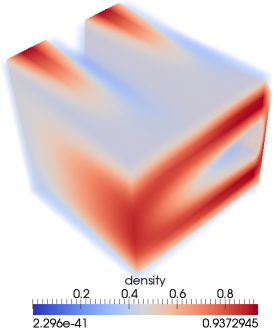
ADAPT Files (*.nc *.cdf *.elev *.ncd)
Adaptive cosmo files (*.cosmo)
ADIOS2 BP3 File (Corelmage) (*.bp)
ADIOS2 BP4 Directory (Corelmage) (*.bp)
ADIOS2 BP4 Metadata File (Corelmage) (md.idx)
AMR Enzo Files (*.boundary *.hierarchy)
AMR Flash Files (*.flash *.flash)
AMR Velodyne Files (*.xamr *.Xamr *.XAMR)
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)
CEAued Files (*.ued *.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)
Exodus1 (*.g *.gz *.ex2 *.ex2v2 *.exo *.gen *.par
ExtrudedVol Files (*.exvol)
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 *.ff)
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 *.ag *.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)
H5Nimrod Files (*.h5nimrod)
H5Part particle files (*.h5part)
HyperTreeGrid (*.htg)
HyperTreeGrid (partitioned) (*.phtg)
Image Files (*.pnm *.ppm *.sdt *.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 (*.m)
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 (*.prds *.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)
POINTTED 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 file (*.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 (Flugin) (*.H)
Silo Files (*.silo *.pdb *.silo.series *.pdb.series)
SLAC Mesh Files (*.ncdf *.nc)
SLAC Particle Files (*.ncdf *.netcdf)
Spheral Files (*.spheral *.sv)
Spy Plot History Files (*.hsth *hscr*)
SpyPlot CTH dataset (*.spt* 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 *.ttt)
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)
VTK 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 (*.xmdv)
XMol Molecule Files (*.xyz)
XYZ Files (*.xyz)

Exemple : lire des données brutes (binaires)

Visualisez la fonction suivante pour $x, y, z \in [0, 1]$ sur une grille $16 \times 16 \times 16$:

$$f(x, y, z) = (1 - z)\left[(1 - y)\sin(\pi x) + y\sin^2(2\pi x)\right] + z\left[(1 - x)\sin(\pi y) + x\sin^2(2\pi y)\right]$$

1. Ouvrez le fichier `data/simpleData.raw`
   - Lorsque demandé, sélectionnez IMAGE READER
2. Spécifiez ensuite les propriétés des données :
   - *Data Scalar Type* : `float`
   - *Data Byte Order* : `LittleEndian`
   - *Scalar Array Name* : `density`
   - *File Dimensionality* : `3`
   - *Data Extent* : `0` à `15` pour chaque dimension (`1` à `16` chargerait les données incorrectement)
3. Modifiez la vue : *Outline*, *Points*, *Wireframe*, *Volume*
4. Selon la vue, on peut éditer l'échelle de couleurs
5. Essayez *File* → *Save Data...* et sauvegardez avec le type ParaView (`*.pvd`), supprimez l'objet et chargez le `*.pvd` – ce fichier contient toutes les propriétés des données



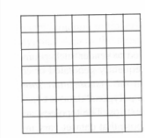
VTK = Visualization Toolkit

- Système logiciel libre pour l'infographie 3D, le traitement d'images et la visualisation
- Interfaces pour C++, Tcl, Java et Python
- ParaView est basé sur VTK ⇒ prend en charge tous les formats de fichiers VTK standard
- Les différents formats de fichiers VTK :
  1. Anciens formats sériels (*.vtk) : lignes d'entête en ASCII + données ASCII/binaires
  2. Formats XML : balises XML + données ASCII/binaires/compressées
     - plus récent, largement préféré aux anciens formats VTK
     - prend en charge la lecture et l'écriture en parallèle, la compression, l'encodage binaire portable (*big/little endian*), les accès aléatoires, etc.
  3. Formats VTKHDF (en développement depuis 2022)

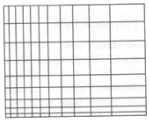
`https://docs.vtk.org/en/latest/vtk_file_formats/index.html`

Données 3D VTK : principaux types de discrétisation (1/2)

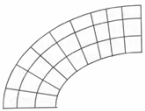
- **Image/Points structurés** : `*.vti`, points sur une grille rectangulaire régulière, des scalaires ou des vecteurs à chaque point


(a) Image Data

- **Grille rectiligne** : `*.vtr`, comme le précédent, mais l'espacement entre les points peut varier ; il faut fournir les intervalles le long des axes de coordonnées et non les coordonnées de chaque point


(b) Rectilinear Grid

- **Grille structurée** : `*.vts`, topologie régulière, mais géométrie irrégulière ; il faut indiquer les coordonnées de chaque point


(c) Structured Grid

Données 3D VTK : principaux types de discrétisation (2/2)

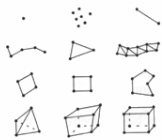
- **Particles/Points non structurés** : `*.particles`


(d) Unstructured Points

- **Données polygonales** : `*.vtp`, topologie et géométrie non structurées, coordonnées de points, cellules 2D uniquement (c'est-à-dire pas de polyèdres), adapté aux cartes


(e) Polygonal Data

- **Grille non structurée** : `*.vtu`, topologie et géométrie irrégulières, coordonnées de points, cellules 2D/3D, adaptée à l'analyse par éléments finis et à la conception de structures


(f) Unstructured Grid

Données 3D VTK : attributs des ensembles de données

Un fichier VTK peut stocker plusieurs ensembles de données, chacun pouvant être de l'un des types suivants :

- Scalaires : des valeurs individuelles de densité, de température, de pression, etc.
- Vecteurs : direction et magnitude, par exemple des vecteurs de vitesse
- Normales : vecteurs de direction ($|\mathbf{n}| = 1$) utilisés pour l'ombrage
- Tenseurs : tenseurs symétriques de 3×3 valeurs réelles, par exemple des tenseurs de stress mécanique
- Table de correspondance : chaque entrée de la table est un tuple rouge-vert-bleu-alpha (où alpha=1 est opaque et alpha=0 est transparent) ; si le format du fichier est en ASCII, les valeurs de la table de correspondance doivent être des nombres à virgule flottante dans la plage [0,1]
- Coordonnées de texture : utilisées pour le mappage de textures
- Données de champs (`FieldData`) : tableaux de tableaux de données

Exemples : lire un fichier VTK de l'ancien format

Attention : Stocker de grands ensembles de données en ASCII n'est pas une bonne idée ; nous allons ici examiner des fichiers VTK textuels à des fins pédagogiques

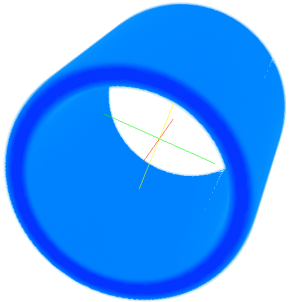
1. Exemple de points structurés : `data/volume.vtk`
   - Grille régulière de $3 \times 4 \times 6$ points, un champ scalaire (`density`) et un champ vectoriel (`velocity`)
2. Exemple de grille structurée : `data/density.vtk`
   - Grille de $2 \times 2 \times 2$ points, un seul champ scalaire
3. Exemple plus complexe (données de polygone) : `data/cube.vtk`
   - Un cube est représenté par six faces polygonales
   - Une composante scalaire, une normale et des données de champs sont définies pour chacune des six faces (CELL_DATA)
   - Des données scalaires sont associées aux huit sommets (POINT_DATA). Une table de correspondance de huit couleurs, associées aux données scalaires, est également définie.

Exercice : créer un ensemble de données VTK 3D

- Visualisez la fonction “cylindre” 3D

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

dans le domaine $x, y, z \in [0, 1]$ avec $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$,
et reproduisez la vue ci-contre



Instructions :

- Créez un fichier `*.vtk` avec les 30^3 valeurs du fichier `data/cylinder.dat`
- Ajoutez et ajustez une entête ASCII en s’inspirant de `data/volume.vtk`
- Visualisez les données en mode *Volume*
- Éditez le *Color Map* : cliquez sur *Choose Preset* et sélectionnez *Warm to Cool* dans *All* ; ensuite, abaissez la courbe d’opacité à 0 vis-à-vis le bleu pâle

Créer un fichier VTK XML via un code C++ (ou autre)

Pour des données plus volumineuses (Mo, Go), on favorisera un format binaire

- Une bonne option serait le format XML avec des données binaires et des métadonnées XML, et d'écrire ces données via une bibliothèque VTK depuis C++, Java ou Python
- Par exemple, le programme `codes/SGrid.cpp`, compilé via le `codes/Makefile`, génère le fichier `data/halfCylinder.vts`

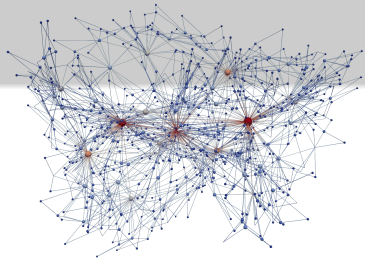
  Cet exemple montre comment créer une grille structurée, définir ses coordonnées, la remplir avec des scalaires et des vecteurs, et l'écrire en XML dans un fichier `*.vts`.
- Pour l'exécuter, vous devez installer la bibliothèque VTK C++ (de manière autonome ou via ParaView) ; regardez dans `codes/Makefile` pour voir les fichiers requis

```
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
```

- D'autres exemples sont annexés au code source de VTK ou disponibles en ligne : `https://examples.vtk.org/site/Cxx/`

Créer un fichier VTK XML via Python

Et créer des graphiques 3D



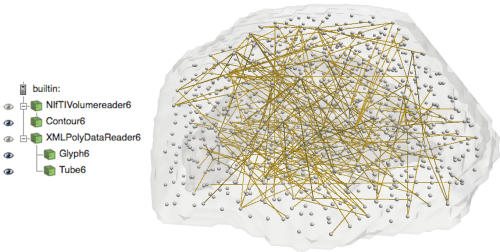
```
$ pip install vtk networkx [scipy]
$ python dgm.py 7
```

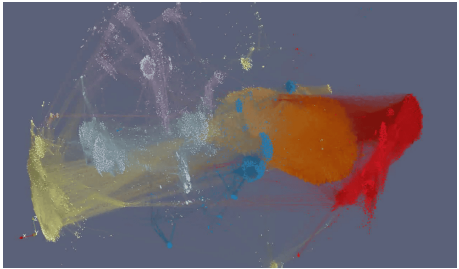
Voir : `codes/{writeNodesEdges,dgm}.py`

```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')
```

Considérer ParaView comme une vue sur les classes VTK



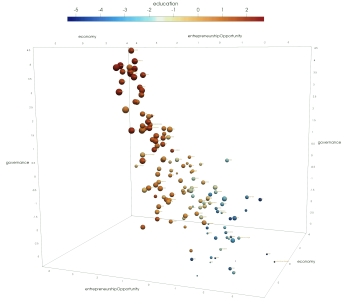


hidden/mutOnCtOrbits.mp4 en démo

```
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();
```

Exercice : Prosperity Index – graphique 3D montrant 5 attributs (1/2)

- Données tirées du Legatum 2015 Prosperity Index (`https://www.prosperity.com/`)
  ⇒ Pour chaque pays, on a huit valeurs de classement sauvegardées dans `data/legatum2015.csv`
  - `economy`
  - `entrepreneurshipOpportunity`
  - `governance`
  - `education`
  - `health`
  - `safetySecurity`
  - `personalFreedom`
  - `socialCapital`

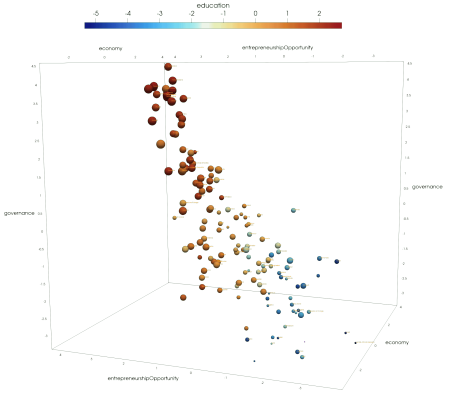


- Complétez et exécutez le code `codes/countries.py` de sorte à écrire cinq attributs dans `data/countries.vtp`
  - Positions `[x,y,z]` selon les valeurs des colonnes `economy`, `entrepreneurshipOpportunity` et `governance`
  - Champ scalaire selon les valeurs de `education`
  - Champ scalaire 2 selon les valeurs de `safetySecurity`

```
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)
```

Exercice : Prosperity Index – graphique 3D montrant 5 attributs (2/2)

- Chargez le fichier `countries.vtp`
- Ajoutez un filtre : *Filters* → *Common* → *Glyph*
  - Sous *Glyph Source*, sélectionnez *Glyph Type* = `Sphere`
  - Sous *Masking*, sélectionnez *Glyph Mode* = `All Points`
- Configurez l'objet Glyph :
  - Sous *Coloring*, choisissez le champ `education`
  - Sous *Scale*, sélectionnez *Scale Array* = `safetySecurity` et ajustez le *Scale Factor* autour de 0.05
  - Cherchez et activez le *Axes Grid*. Cliquez sur *Edit* et configurez le nom des axes
- En option, faites afficher le nom des pays
  - Appuyez sur la touche V pour faire afficher le panneau *Find Data*
  - Sélectionnez *Data Producer* = `countries.vtp`, ajustez le critère de sélection [`ID`] [`is >=`] `0` (donc tout) et appuyez sur le bouton *Find Data*
  - Sous *Selection Labels*, sélectionnez *Point Labels* = `tag` (uniquement)
  - Cliquez sur *Edit Label Properties*, réduisez l'opacité à 0 et ajustez le *Point Label Font* à une taille de 10



Autre option pour écrire des fichiers VTK XML via Python

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

```
$ pip install pyevtk
```

- Plusieurs exemples dans ce répertoire :
  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})
```

À propos des formats NetCDF et HDF5

- Le format VTK est incroyablement polyvalent et peut décrire de nombreux types de données
- Or, très souvent en science, on a uniquement besoin de stocker et de visualiser des tableaux multidimensionnels
- Problème : comment stocker un tableau de 2000^3 nombres réels ($\sim$32 Go)?
  - En ASCII – oubliez-ça : trop de caractères par nombre, sinon perte de précision
  - En binaire brut – possible, mais avec plusieurs problèmes de portabilité
  - Un des formats VTK – probablement exagéré pour des tableaux simples
- Les formats de données scientifiques viennent à la rescousse ; parmi les plus populaires, on trouve NetCDF et HDF5
  - binaire (bien sûr!)
  - auto-descriptif (comprend des métadonnées)
  - portable (multiplateforme) : bibliothèques pour de nombreux systèmes d'exploitation, types de données universels, gestion de l'ordre des octets (petit-boutiste ou gros-boutiste), etc.
  - permet des lectures et des écritures en parallèle (via MPI-IO)
  - permet la compression des données

Prise en charge de NetCDF dans ParaView

- Le format NetCDF est pris en charge nativement dans ParaView
  - Le programme `codes/writeNetCDF.cpp` (ou `codes/writeNetCDF.f90` en Fortran) écrit un volume de 100^3 valeurs de densité (formant un beigne au centre) dans un fichier NetCDF

Exemple C++

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

Exemple F90

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

- ParaView comprend les conventions NetCDF courantes, par exemple celles pour les métadonnées du climat et de la météo (`https://cfconventions.org/`) : données 2D ou 3D sur une sphère, axes de coordonnées, valeurs de remplissage, etc.
  - example 1 en démonstration : données en 2D `hidden/ice.nc`
  - example 2 : données en 3D `hidden/temp1.png`
  - example 3 : visualisation 3D plus élaborée `hidden/tempsalt.mp4`

À propos des sphères

Et si on superposait la topographie à notre visualisation?

- `https://www.earthmodels.org` avait de bons exemples dans le passé...
- Option 1 : charger la topographie précalculée stockée sous forme de données polygonales
- Option 2 : *mapper* une image sur le globe. Les différentes étapes sont :
  1. créer un objet *Sphere* à haute résolution
  2. ajouter un filtre *Texture-Map-to-Sphere* et cliquer sur *Apply* pour faire apparaître la propriété *Miscellaneous:Texture*
     - cette propriété permet de créer des *texture coordinates*
     - note : nous verrons les filtres plus en détail à la prochaine section
  3. sous *Miscellaneous:Texture*, utiliser le menu déroulant pour charger une image PNG et cliquer sur *Apply*
  4. vers le haut des propriétés, décocher *Prevent Seam*, cocher Seamless U et Seamless V, et cliquer sur *Apply*
  5. avoir une coloration par `Solid Color` et visualiser en mode `Surface`

Prise en charge de HDF5 dans ParaView

- Il n'y a aucune prise en charge **native** de HDF5. Cependant, ParaView prend en charge le format conteneur XDMF (eXtensible Data Model and Format) qui utilise HDF5 pour stocker les données – pour référence, voir : `https://www.xdmf.org`
- XDMF = XML pour des données de **petite taille** + HDF5 pour des données **massives**
  - le type des données (entier, virgule flottante, etc.), la précision, le nombre de dimensions et les dimensions sont décrits entièrement dans la couche XML (et aussi HDF5)
  - les données, potentiellement énormes, sont encodées en HDF5
- Un fichier XML peut référencer plusieurs fichiers HDF5 (ex.: un fichier par processus)
- Pas besoin de bibliothèques HDF5 pour effectuer des opérations simples
- Une API C++ est fournie pour lire/écrire des données XDMF
- Peut être utilisé depuis Python, Tcl, Java, Fortran via des appels C++
- En Fortran, on peut générer des fichiers XDMF via des appels HDF5 + texte brut pour le conteneur XML : `https://www.xdmf.org/index.php/Xdmf3_Fortran_API.html`
- Prise en charge également de plusieurs formats de fichiers générés par des logiciels tiers qui utilisent eux-mêmes HDF5 sous-jacent

Récapitulatif des formats de fichiers en entrée

- Données binaires brutes
- Anciens formats VTK (`*.vtk`), avec les données écrites en ASCII, pour les petits ensembles de données
  - Points structurés (image)
  - Grille structurée
  - Données polygonales
- Des formats VTK XML pour les grands ensembles de données : ce sont les plus polyvalents, utilisables en C++ et en Python
  - Grille structurée (`*.vts`)
  - D'autres discrétisations peuvent être sauvegardées en utilisant la classe correspondante, par exemple : `vtkPolyData`, `vtkRectilinearGrid`, `vtkStructuredGrid`, `vtkUnstructuredGrid`, etc.
- NetCDF est pris en charge nativement et le format HDF5, via XDMF
- ParaView prend en charge nativement de nombreux autres formats

ENRICHIR LA VISUALISATION AVEC DES FILTRES

Introduction aux filtres

Plusieurs caractéristiques intéressantes d'un ensemble de données ne peuvent être observées en surface : une grande quantité d'informations utiles se trouve plutôt à l'intérieur, ou peut être extraite d'une combinaison de variables.

Parfois, une vue souhaitée n'est pas disponible pour un certain type de données, par exemple :

- un ensemble de données 2D $f(x, y)$ sera affiché comme une surface 2D, même en 3D (essayez de charger `data/2d000.vtk`), mais nous pourrions vouloir visualiser l'élévation $z = f(x, y)$.
- vue volumétrique – non disponible pour toutes les discrétisations VTK, mais disponible, entre autres, pour les points structurés et les grilles non structurées avec connectivité fournie

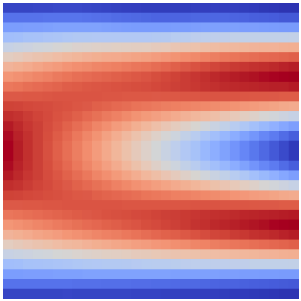
Les **filtres** sont des unités fonctionnelles qui traitent les données pour générer, extraire ou dériver des caractéristiques supplémentaires. Les connexions entre les filtres forment un **pipeline de visualisation**.

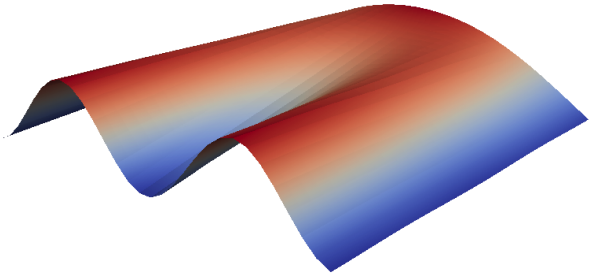
ParaView fournit déjà plus de 140 filtres et on peut en ajouter via des scripts Python

- Explorez le menu *Filters* ; certains filtres se trouvent aussi dans la barre d'outils
- L'édition des propriétés d'un filtre se fait dans le panneau *Properties*

Exercice – visualiser des données 2D en 3D

- Chargez le fichier `data/2d000.vtk` qui échantillonne la fonction 2D $f(x, y) = (1 - y)\sin(\pi x) + y\sin^2(2\pi x)$ pour $x, y \in [0, 1]$ sur une grille 30×30
- Sélectionnez les données dans le *Pipeline Browser*, ajoutez le filtre *Miscellaneous* $\rightarrow$ *Warp By Scalar* et activez la vue *3D* dans la fenêtre de rendu
- Ajustez le *Scale Factor* à 0.3 pour reproduire la vue 3D ci-dessous



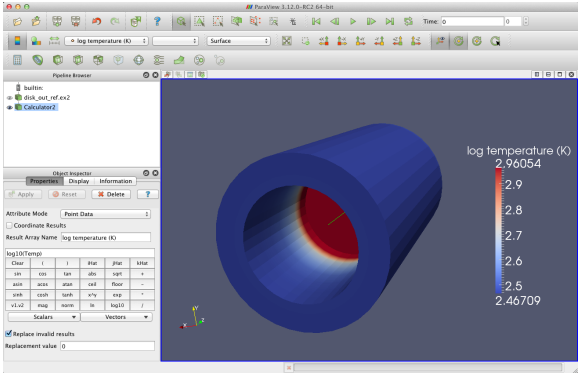


Principaux filtres dans la barre d'outils

- Un **Calculator** évalue une expression mathématique pour chaque point ou chaque cellule
- Un **Contour** extrait des points, des isocontours ou des isosurfaces à partir d'un champ scalaire
- Un **Clip** enlève toute la partie de la visualisation d'un côté d'un plan dans l'espace 3D
- Un **Slice** intersecte la visualisation avec un plan ; l'effet est similaire au *Clip*, excepté qu'il ne reste que la géométrie à l'intersection du plan et de la visualisation
- Un **Threshold** extrait les cellules se trouvant dans un certain intervalle du champ scalaire
- Un **Glyph** place des *glyphes* à chaque point d'un maillage ; les glyphes peuvent être orientés selon un champ vectoriel et agrandis selon un champ vectoriel ou scalaire
- Un **Stream Tracer** initialise un champ vectoriel avec des points, puis suit ces points initiaux à travers le champ vectoriel en régime permanent

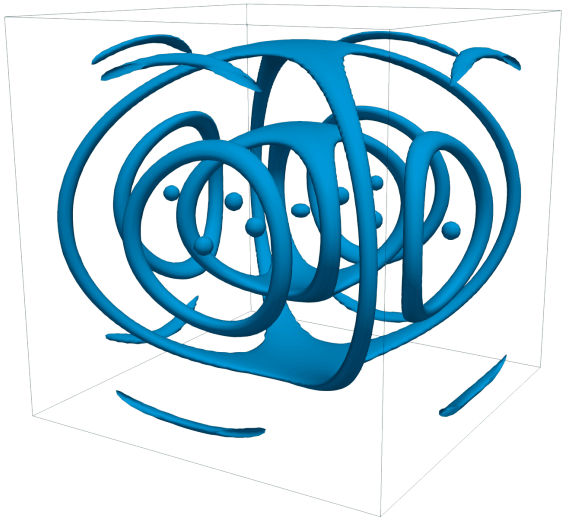
Calculator – Le filtre calculateur

- Du fichier `data/disk_out_ref.ex2`, chargez les variables `Pres`, `Temp` et `V`
- Visualisez la surface et coloriez selon chacune de ces trois variables ; avec l'outil *Toggle Colour Legend Visibility*, affichez et observez l'échelle des valeurs
- Ajoutez ensuite un filtre **Calculator** avec la formule `log10(Temp)`
  - les menus déroulants *Scalars* et *Vectors* aident à entrer les variables
  - mettre un nom unique : `LogTemp`
  - le bouton *? Help* est étonnamment utile
- Créez deux autres calculateurs :
  - `Pres_Temp` – formule : `Pres/Temp`
  - `Mag_V` – formule : `mag(V)`
- Dans le *Pipeline Browser*, modifiez la visibilité de chaque objet en cliquant sur l'icône en forme d'œil à côté de celui-ci



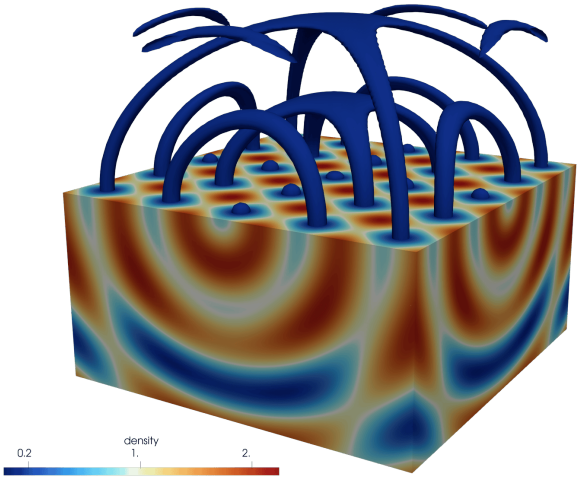
Exercice : recréez cette visualisation

- Données à charger : `data/sineEnvelope.nc`
- Indices :
  - filtre de type *Contour*
  - isosurface à 0.16
  - échelle de couleurs de 0.15 à 0.20



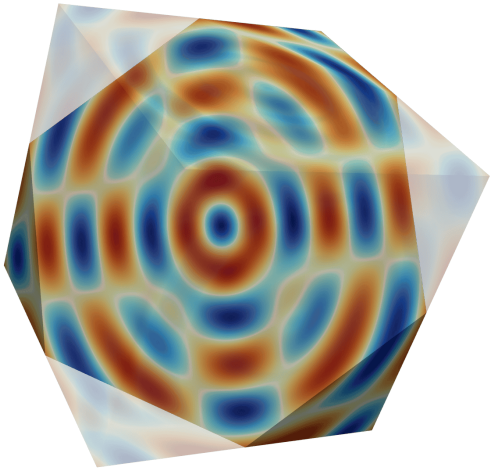
Exercice : recréez cette visualisation

- Données à charger : `data/sineEnvelope.nc`
- Indices :
  - pipeline de visualisation à deux filtres
  - filtre de type *Clip*
  - l'échelle de couleurs *density*, partagée par les deux calculateurs, s'ajuste automatiquement
  - *Show Plane*



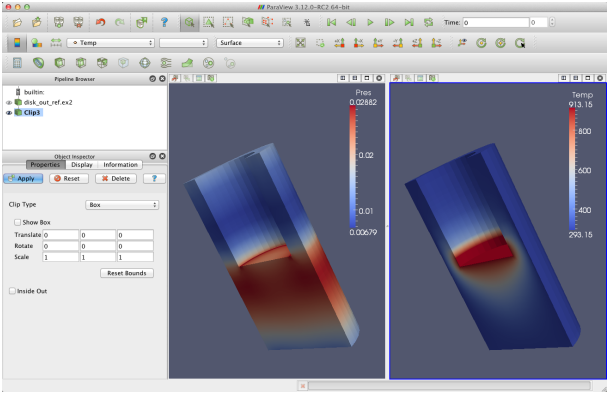
Exercice : recréez cette visualisation

- Données à charger : `data/sineEnvelope.nc`
- Indices :
  - surface avec la propriété d'opacité à 0.25
  - normale (1, 1, 1)



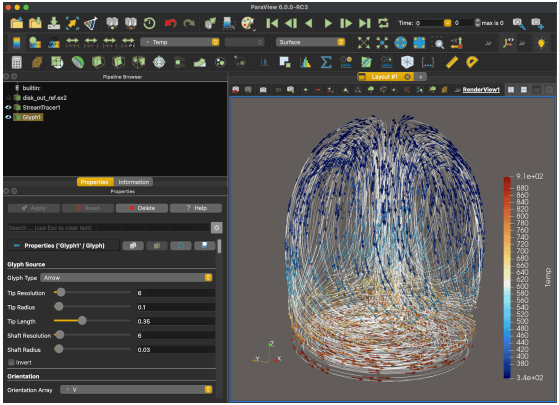
Exercice : recréez cette visualisation

- Données à charger : `data/disk_out_ref.ex2`
  - `Pres` et `Temp` seulement
- Indices :
  - compléter une visualisation à la fois
  - coloration selon `Pres` à gauche
  - coloration selon `Temp` à droite
  - lier les caméras



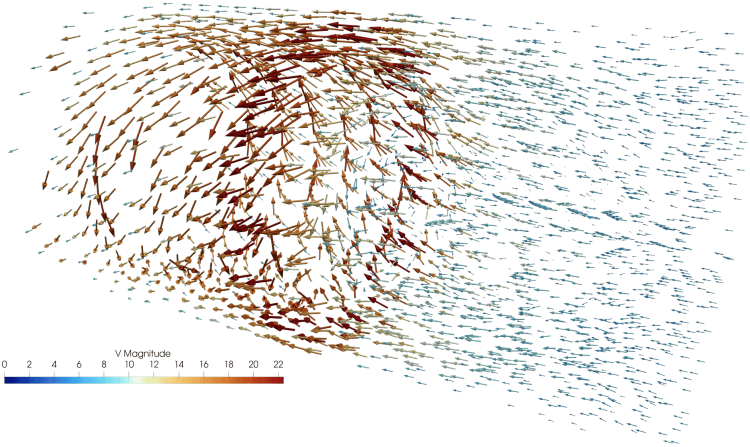
Visualisation vectorielle : lignes de flux et glyphes

1. Chargez `Temp` et la vélocité `V` du fichier `data/disk_out_ref.ex2`
2. Ajoutez un filtre *Stream Tracer*, configurez le *Seed Type* = `Point Cloud` et le *Radius* = `3` pour la sphère. Ensuite :
   - Essayez différents *Number Of Points* et différents *Maximum Streamline Length*
3. (Optionnel) transformez les lignes de flux en tubes en ajoutant : *Filters* → Miscellaneous → *Tube* (*Radius* = `0.03`)
4. Ajoutez des glyphes aux lignes de flux pour indiquer l'orientation et l'amplitude :
   - sélectionnez le *StreamTracer** dans le *Pipeline Browser*
   - ajoutez-lui un filtre *Glyph* de *Type* = `Arrow`, avec *Orientation Array* = `V`, *Scale Array* = `No scale array` et *Scale Factor* = `0.5`
   - Coloriez les glyphes selon `Temp`

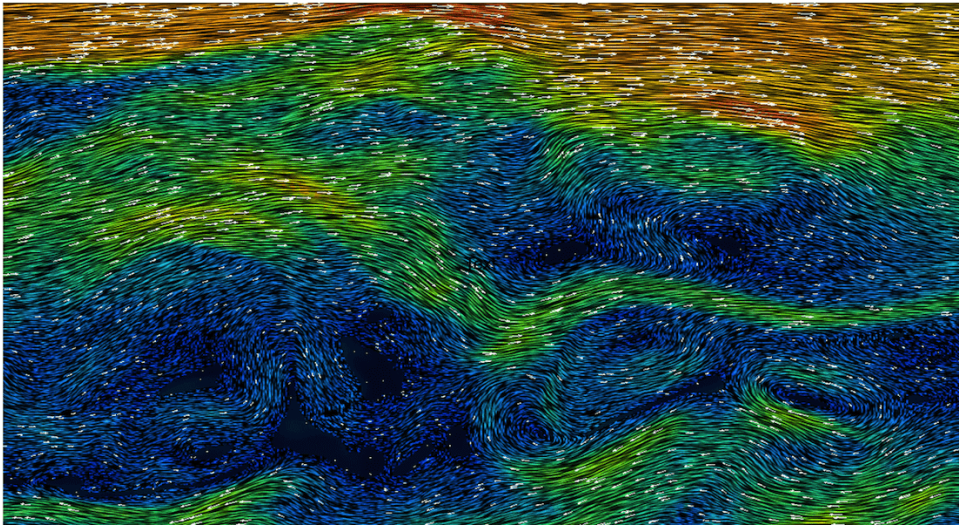


Exercice : remplir le volume entier de vecteurs

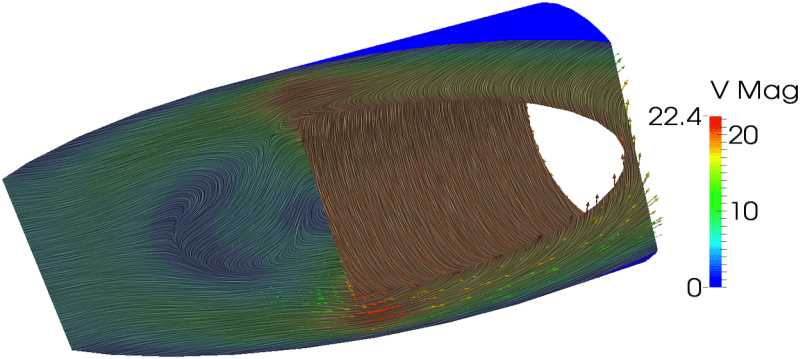
Chargez `V` de `data/disk_out_ref.ex2` et affichez le champ de vélocité avec des flèches orientées selon `V`, mais dimensionnées et colorées selon la norme de `V`



Représentation Line Integral Convolution (ou *Surface LIC*)



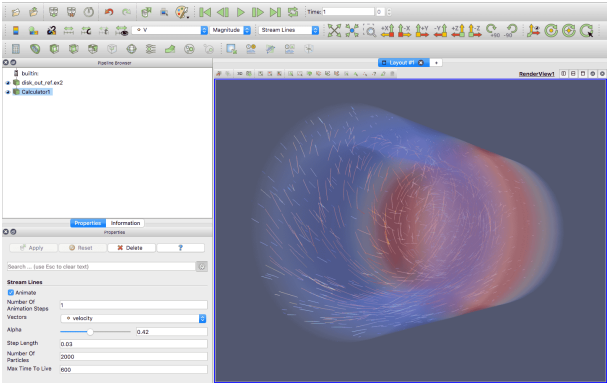
Utiliser le *Line Integral Convolution* dans ParaView



- Chargez `V` de `data/disk_out_ref.ex2`
- Ajoutez un filtre de type *Clip* et configurez sa normale à $(1, 0, 0.4)$
- Sélectionnez la représentation *Surface LIC* et changez l'échelle de couleurs (*Rainbow Uniform*)
- Dans les propriétés du *Clip*, sous *SurfaceLIC: Integrator*, vérifiez que `V` est sélectionné
- Jouez ensuite avec les valeurs de *Number Of Steps* et de *Step Size*

Représentation Stream Lines (tracé en temps réel)

Détails à la page `http://bit.ly/2NFNcvQ`



- Dans *Tools* → *Manage Plugins*, activez le *StreamLinesRepresentation*
- De `data/disk_out_ref.ex2`, chargez `Pres`, `Temp` et `V` ; affichez `Pres` en *Surface*, *opacité* = `0.25`
- Ajoutez un *Calculator* avec la formule `V` et utilisez la représentation *Stream Lines* (affichez la source)
- Dans les propriétés du *Calculator**, coloriez selon `Temp` et doublez le Step Length

Format simple : données 3D sous forme de colonnes

- `data/tabulatedPoints.txt` contient 100 points aléatoires, chaque ligne ayant les valeurs de `x,y,z,scalar`
- Chargez les données CSV et ajoutez un filtre *Table To Points* en assignant les champs *X/Y/Z Column*
- Ajoutez un filtre *Glyph* pour voir des sphères à la place des points, puis coloriez selon `scalar`
- Note : il n'y a aucune topologie implicite. Vous pouvez optionnellement passer les points via un filtre *Delaunay 3D*, suivi d'un filtre *Extract Edges* et ensuite d'un *Tube*

- `data/tabulatedGrid.txt` contient 1000 points sur un maillage cartésien $10 \times 10 \times 10$, chaque ligne ayant les valeurs de `x,y,z,scalar`
- Chargez les données CSV et ajoutez un filtre *Table To Structured Grid* en assignant les champs *Whole Extent* de `0` à `9` pour chaque dimension et les champs *X/Y/Z Column*
- Les données doivent présenter une topologie implicite pour que ce filtre fonctionne!

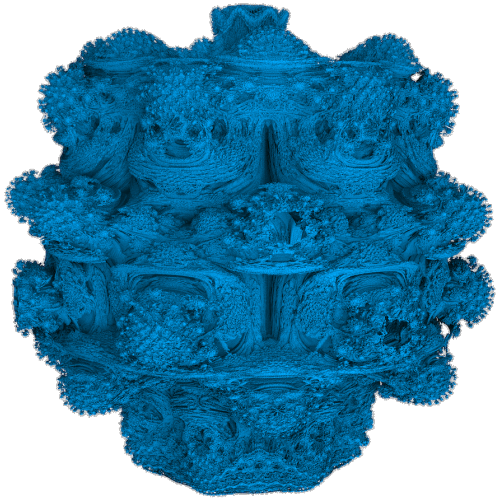
Format déconseillé pour les grands ensembles de données : gaspillage d'espace disque et de bande passante!

➢ `tabulatedPoints.txt` est 6231 octets vs 1600 octets en données binaires à simple précision

➢ `tabulatedGrid.txt` est 20 013 octets vs 4000 octets en données binaires à simple précision

Exercice : quelque chose de plus complexe

- Un Mandelbulb de puissance 8 est une fractale 3D
- Chargez le fichier `mandelbulb300.nc` ($300 \times 300 \times 300$ valeurs)
  - Si votre ordinateur a assez de mémoire, vous pouvez essayez de charger `mandelbulb800.nc` (800^3 valeurs)
- Ensuite :
  1. comparez la taille de la grille avec la taille du fichier ; est-ce proportionnel?
  2. essayez les représentations *Slice* et *Volume*, ainsi qu'un filtre *Contour* pour visualiser des isosurfaces
  3. essayez de recréer la figure ci-contre (à basse résolution) ; faites attention aux normales et à l'éclairage



Exercice : optimisation 3D

https://fr.wikipedia.org/wiki/Fonction_de_test_pour_l'optimisation

Le fichier `data/stvol.nc` contient une variante discrétisée et agrandie de la fonction 3D de Styblinski-Tang, construite avec `codes/optimization.c`

$$f(x_1, x_2, x_3) = \frac{1}{2}\sum_{i=1}^{3}(\xi_i^4 - 16\xi_i^2 + 5\xi_i), \text{ où } \xi_i \equiv 8(x_i - 0.5) \text{ et } x_i \in [0, 1]$$

Analysons cette fonction à partir du fichier de données :

1. Quelle est la taille de la grille? Est-ce que cela correspond à la taille du fichier?
2. Trouvez l'emplacement approximatif du **minimum global** de $f(x_1, x_2, x_3)$ à l'aide de techniques visuelles (tranches, isosurfaces, seuils, rendu volumique, etc.)

- Note : vous pouvez trouver les coordonnées exactes du minimum global en utilisant *Filters* → *Data Array* → *Statistics* → *Descriptive Statistics* et en triant les valeurs de *f(x,y,z)* de `stvol.nc`

Mise en garde

- De nombreux filtres de visualisation transforment les données structurées (en grille) en données non structurées. Par exemple : *Clip*, *Slice*, etc.
- L'empreinte mémoire et la charge du processeur peuvent augmenter très rapidement. Par exemple, couper 400^3 valeurs à 150 millions de cellules peut prendre $\sim$ 1 heure sur un seul cœur CPU ; il devient préférable de travailler en mode distribué

Le filtre *Python Calculator*

- *Filters* → *Programmable* → *Python Calculator*
- Exemple : pour calculer la vorticité, choisissez un champ vectoriel, entrez `curl(V)` et appelez-le `vorticity`
- Fonctions prises en charge pour les tableaux : `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()`

Fonctionnalités de filtrage avancées

- Il est possible de fusionner plusieurs filtres existants en un filtre personnalisé : `https://cfdengine.com/newsletter/104` (mini tutoriel)
  Tools → *Create Custom Filter*
  On y édite ses entrées, ses sorties et ses propriétés

- Il est aussi possible de créer des filtres en Python :
  `https://docs.paraview.org/en/latest/ReferenceManual/pythonProgrammableFilter.html`
  Filters → *Programmable* → *Programmable Filter*
  (à voir dans la partie 2 – Scripter des visualisations via ParaView)

- Enfin, il est possible de créer des filtres sous forme de plugiciel, de les compiler en tant que bibliothèques avec la même version de ParaView que celle avec laquelle ils seront déployés :
  `https://www.paraview.org/paraview-docs/nightly/cxx/PluginHowto.html`