Enhancing DEM Visualisation: ParaView ~~*Plugin*~~ Macro Development

J.P. Morrissey, ***University of Edinburgh***

CCC-ParaSolS Project

1st September 2026

Science and Technology Facilities Council

CoSeC
Computational Science Centre for Research Communities

IMPERIAL

THE UNIVERSITY of EDINBURGH
School of Engineering

Gran ED

Pre-Warning: Tutorial Datasets

- Three different datasets to work with:
  - Polydisperse spheres
  - Polydisperse multi-spheres with multiple shapes
  - Polydisperse polyhedrals with multiple shapes

- Available in both **XML** and **VTKHDF** formats

- Download from here:
  - https://zenodo.org/communities/parasols/records?q=&l=list&p=1&s=10&sort=newest

Pre-Warning: ParaView Macros

- Available at: https://git.ecdf.ed.ac.uk/parasols/paraview-macros

- Basic installation instruction and description of some settings that can be customised

- Clone or download now if you want to follow along

Visualisation Tools

- There is no single best option, and some relatively common ones are listed
- Open-source tools are combination of script and GUI-based visualisers
- Most support wide variety of common file formats
  - Many of the open-source tools are built upon the **VTK framework**

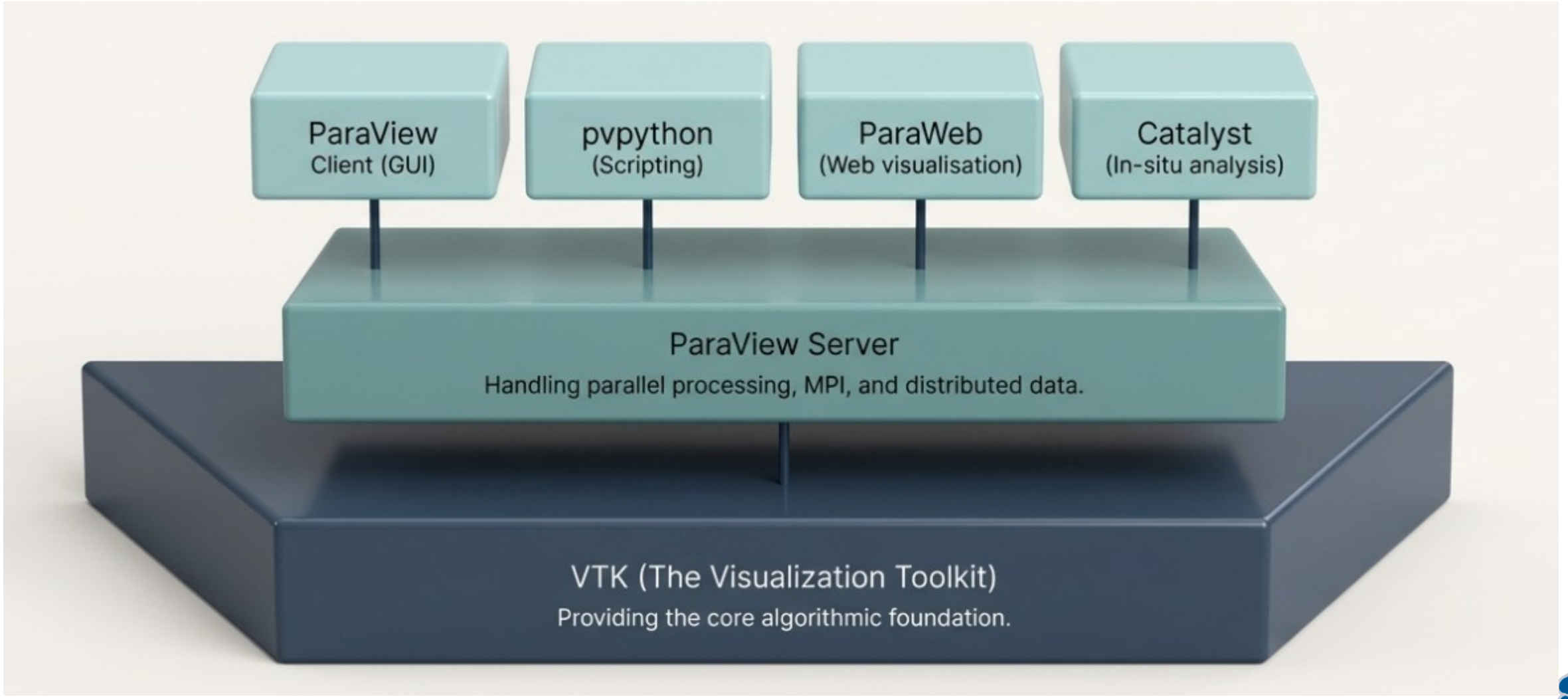
- **Commercial / Closed-source:**
  - Ensight
  - Ovito*
  - GiD
  - Tecplot
- **Open-source**
  - VisIt
  - ParaView
  - Ovito*
  - Blender
    - via BVTKNodes addon
  - Mayavi
  - PyVista
  - Vedo

  Python libraries
  - POV-Ray

*limited features in open-source version

The Visualisation Toolkit (VTK)

Scientific Visualisation Stack



What is the VTK Framework?

- The **Visualization Toolkit (VTK)** is an *"open-source, cross-platform system for 3D graphics, image processing, volume rendering, scientific visualization, and 2D plotting"*
- It supports a wide variety of visualization algorithms
- Takes advantage of both threaded and distributed memory parallel processing for speed and scalability
- VTK is designed to be **platform agnostic** and has bindings in many languages such as Python and Java

- Development originally began at GE in 1993 by Schroder, Martin & Lorensen
- Founded Kitware Inc. in 1998
- Strong support from Sandia and Los Alamos National Labs
- ParaView development started in 2000

kitware VTK

Schroeder, Will; Martin, Ken; Lorensen, Bill (2006), *The Visualization Toolkit* (4th ed.), Kitware, ISBN 978-1-930934-19-1

A Three-step Visualisation Process

1. Reading (Sources)

Importing external data sources (e.g. VTK files) or creating internal data sources (e.g. sphere glyph)

2. Filtering (Algorithms)

Extracting or deriving features from a data source (e.g. clipping, streamlines, etc.)

3. Rendering (Sinks/Views)

Converting the processed spatial data into viewable formats or images

The VTK Data Format

VTK Data Format

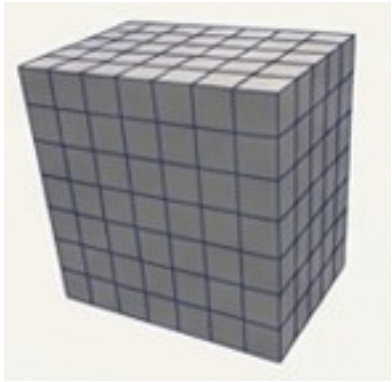
- **Structured Grids**: Regularly spaced grids where data is organised in a structured manner
  - **Image data:** A special case of structured grid where data is organized in a regular 3D array
  - **Rectilinear grid:** A structured grid where the spacing between points can vary along each axis
  - **Structured grid:** A general structured grid where points can be arranged in a regular pattern but may have varying spacing

- **Unstructured Grids**: Irregularly spaced grids that can represent complex geometries
  - **Unstructured grid:** A more general unstructured grid that can represent a wider variety of geometries, including tetrahedra, hexahedra, and other cell types
  - **Polydata:** An unstructured grid that represents geometric shapes, such as points, lines, and polygons

VTK Data Model – Structured Data

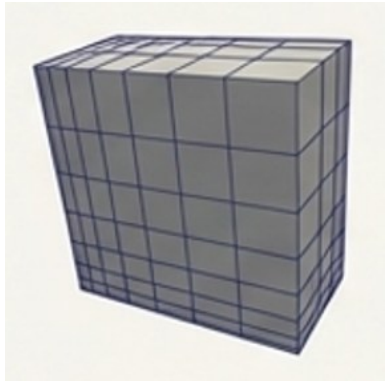
Uniform Rectilinear Grid* (Image Data)

- Storage: Both topology and geometry are defined implicitly
- Performance: Because everything is implicit, it requires the least amount of storage



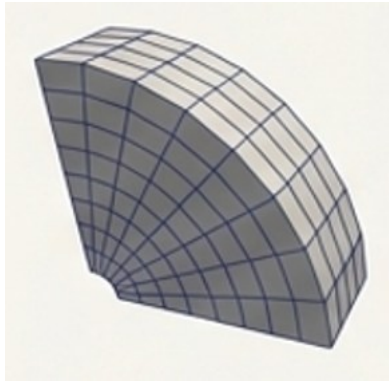
Rectilinear Grid

- Storage: Topology is defined implicitly, while point coordinates are semi-implicitly defined
- Performance: It offers significant memory savings over explicit types because the coordinate arrays are linear (*nx* + *ny* + *nz*) rather than volumetric



Curvilinear Grid (Structured Grid)

- Storage: Topology is implicit, but point coordinates are explicit
- Performance: While it preserves the compact memory footprint for topology, it requires more memory than rectilinear grids to store every point position explicitly



Topology refers to the connectivity, structure, and arrangement of data elements (cells and points), completely independent of their spatial location or geometry.

TK Data Model – Unstructured Data

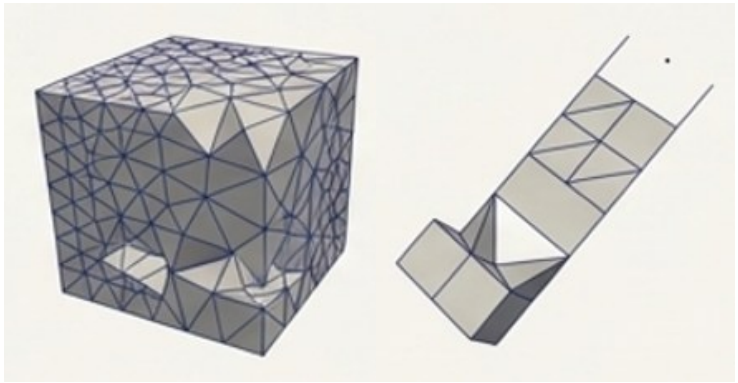
nstructured Grid*

The **most general and flexible** data type

t can represent any combination of 0D, 1D, 2D, and 3D cell types

Storage: Both topology and point coordinates are stored explicitly

Performance: Because all connectivity must be written out cell-by-cell, it has a much **higher memory footprint** than structured data



Polygonal Grid* (Poly Data)

- It consists **strictly** of 0D cells (vertices), 1D cells (lines), and 2D cells (polygons/strips)
- It cannot represent 3D volumetric cells like tetrahedra
- Storage: A ***specialised*** version of an unstructured grid designed for efficient rendering
- Note: Polygonal data represents the basic rendering primitives for ParaView; most other data types must be converted to poly data before they can be displayed in a 3D view

VTK Data Format

Field Data: Data associated with the current dataset – e.g. timestep, etc.

Point Data: Data associated with individual points (each mesh node) in the dataset

Cell Data: Data associated with cells (volumes) in the dataset, which can be used for visualizing interactions between particles

Attributes: Additional data associated with points or cells

– Typically, scalars, vectors, tensors, normals, texture coordiantes

ummary of File Formats

ta Type	Topology	Geometry	Memory Use	Flexibility	Extension	VTKHDF Support
age Data	Implicit	Implicit	**Lowest**	Least (Regular spacing)	*.vti*	✓
ctilinear	Implicit	Semi-Implicit	Low	Moderate (Variable axis spacing)	*.vtr*	**VTKHDF 2.5+**
rvilinear	Implicit	Explicit	Moderate	High (Arbitrary warping)	*.vts*	**VTKHDF 2.5+**
ly Data	Explicit	Explicit	High	High (Limited to 2D surfaces)	*.vtp*	✓
structured	Explicit	Explicit	**Highest**	Greatest (Full 3D volume)	*.vtu*	✓

low Best to Store DEM Data?

DEM data is unstructured in nature and **PolyData** offers the most efficient way of storing this data:

- Particle information can easily be stored as point data (0D)
- Contact information maps lines (1D - CellData)
- Geometry and particle shape templates can easily be stored as polys (2D - CellData)

More efficient than **UnstructuredGrid** as this requires an extra piece of topology information (cell type) in addition to the connectivity and offsets required to define each mesh

- Although it sounds trivial, it becomes significant if you are storing a million extra integers for no good reason!

Depending on the user/developer choice, particle information could be stored as either PointData or CellData

History of File Format

Original file format (*.vtk*) consisted of ascii files with a specific structure

Several limitations of this initial specification:

- Compression not supported
- Parallel writing not supported
- Parsing bottleneck

The `.vtk` format was considered deprecated from VTK 4.0 onwards which introduced the new XML-based format

- Removed some features of older spec
- Added compression support
- Added parallel file format (.pvti, .pvtu, etc,)

VTKHDF was introduced in **VTK 9.2.x** and is still undergoing development but will ultimately replace the XML format

- Better read/write performance
- Partial data loading

toring Scientific Datasets

Storing large datasets in ASCII is generally not a very good idea

- Takes up more space
- May suffer from precision errors

Generally, binary formats should be preferred

VTK XML files are valid XML files and can be parsed with any regular XML reader

- **Caveat**: VTK supports one special case where raw binary data can be added to the appended section creating an **invalid XML file**
  - This means specialist readers are required making the files less portable – these should be avoided
- Unfortunately, DEM codes like MFiX have chosen this approach
- Fortunately, ParaView has a reader that support this

VTK XML Format – Binary Example


```
ion="1.0"?>
ype="PolyData" version="1.0" byte_order="LittleEndian" header_type="UInt64">
a>
NumberOfPoints="20" NumberOfVerts="20" NumberOfLines="0" NumberOfStrips="0" NumberOfPolys="0">
ntData Scalars="1_temp" Vectors="3_force">
taArray Name="1_temp" NumberOfComponents="1" type="Float64" format="binary">
      oAAAAAAAAABgvDpseejpPw6Ed0W3gNM/kuwL31fN5D8Cfx1wrQ/rP2zR+f3q/9g/EMEd+ts/qT+oNtlobme4P/r4fBrGK9o/vym3nbwm7z/pxw36IAHpPwi+7+7NQrY/XxLNdTO85z+FYpT5VQTiP4DgSYzY5us/lGHPhWe+6T8g20I4bRH
DataArray>
taArray Name="2_pressure" NumberOfComponents="1" type="Float64" format="binary">
      oAAAAAAAAADdZNbbgA7gP5ymrMbzOdg/oNUpDBsClT9G1D1TmLTvP7ysfVXNyMM/c0WAwoca7D8N/7iEKkPjP8EgdyZmTek/zpYDvu0j0D/tv+JkeBfvPziklr0hN7E/uK9EEl7p3T/gfLGtIEGhP9WISu7DHOE/mJ/qc2jNvT9kNELrlI/
DataArray>
taArray Name="3_force" NumberOfComponents="3" type="Float64" format="binary">
      4AEAAAAAAAAgpVswV62fPwpRIiBnMdo/QHzbaKFS7z+4L+cOJgfbP9BgBuQUDbc/aLn1Abbgxj9aHwIu+PvSP6kVQrIoQeY/8Eft6Toh0j8oII99u8/UPxBX2HJnmug/xuV6EkiB3j/joAjkKrntP57cNYM1YtU/ZyAShFwy7j/yZzS1kiD
DataArray>
ntData>
Data> </CellData>
nts>
taArray Name="Points" NumberOfComponents="3" type="Float64" format="binary">
      4AEAAAAAAACI/0qfWvPdP7DHS/9sm6s/1RCRUhkd7D88ThUFhVHlP/nLGxz0wOU/dffEocmS4D+SfxCuNTbnP3eMkwfTmeE/Nxz3Am7E7j980oYVrDjdP21fp2p+HOE/CJ/BDUNS1D8cSTIrID3FP9htFlsnpbM/MFbuZLUw6T+
/DataArray>
nts>
s>
taArray Name="connectivity" NumberOfComponents="1" type="Int32" format="binary">
      UAAAAAAAAAAAAAAAAQAAAAIAAAADAAAABAAAAAUAAAAGAAAABwAAAAgAAAAJAAAACgAAAAsAAAAMAAAADQAAAA4AAAAPAAAAEAAAABEAAAASAAAAEwAAAA== </DataArray>
taArray Name="offsets" NumberOfComponents="1" type="Int32" format="binary">
      UAAAAAAAAAABAAAAAgAAAAMAAAAEAAAABQAAAAYAAAAHAAAACAAAAAkAAAAKAAAACwAAAAwAAAANAAAADgAAAA8AAAAQAAAAEQAAABIAAAATAAAAFAAAAA== </DataArray>
rts>
es>
taArray Name="connectivity" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
taArray Name="offsets" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
nes>
ips>
taArray Name="connectivity" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
taArray Name="offsets" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
rips>
ys>
taArray Name="connectivity" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
taArray Name="offsets" NumberOfComponents="1" type="Float64" format="binary"> </DataArray>
ays>
e>
ta>
```

VTK XML Format – Multiblock File

VTM stands for **VTK MultiBlock**

It is an XML-based file format (`.vtm`) that describes a **static collection of datasets** (blocks), each of which can be a VTK file (e.g., `.vtp`, `.vtu`, etc.) or even another `.vtm` file

Used to group multiple datasets into a single logical dataset (multiblock dataset)

Does **not** encode time information

```
<?xml version="1.0"?>
<VTKFile type="vtkMultiBlockDataSet" version="1.0" byte_order="LittleEndian" header_type="UInt64">
  <vtkMultiBlockDataSet>
    <Block index="0" name="Geometries">
      <Piece index="0" name="Factory">
        <DataSet index="0" file="Factory_t_0.vtp"/>
      </Piece>
      <Piece index="1" name="Box">
        <DataSet index="0" file="Box_t_0.vtp"/>
      </Piece>
    </Block>
  </vtkMultiBlockDataSet>
</VTKFile>
```

ParaView Data File

PVD stands for **ParaView Data**

- It is an XML-based file format (.pvd) that acts as a **meta-file** to describe **a time series** of datasets
- Lists datasets (e.g., .vtp, .vtu, .vtm, etc.) associated with specific time steps

Used for **time-dependent simulations** to allow ParaView to load and animate data over time

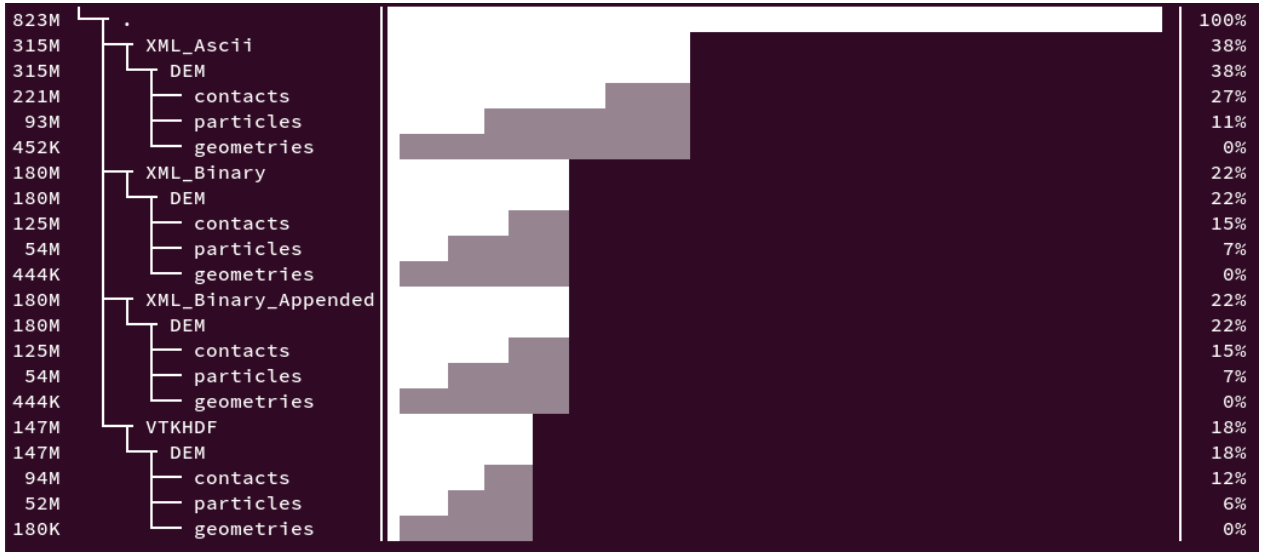
```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1">
  <Collection>
    <!-- timestep: float value for time | file: path to the data file -->
    <DataSet timestep="0.0" group="" part="0" file="data_0.vtp"/>
    <DataSet timestep="0.5" group="" part="0" file="data_1.vtp"/>
    <DataSet timestep="1.0" group="" part="0" file="data_2.vtp"/>
  </Collection>
</VTKFile>
```

Note: In order to store the correct timestep data for a file, .pvd files are required for legacy VTK files. As of VTK 8.2, XML files support the embedding of time metadata as a `FieldData` array of `TimeValue`

Size of Various File Formats

- The amount of space used differs according to the file format
  - Ascii XML files are the largest and larger than legacy vtk files due to additional xml metadata
  - Binary XML is significantly smaller
  - VTKHDF is the most space efficient
- Read and write performance varies across files formats but VTKHDF offers significant performance increases

File Type / Method	Write Speed	Compression Ratio	Notes
Legacy VTK (.vtk)	465 MB/s	0.88	Significant overhead
VTK XML, none	256 MB/s	0.70	Significant overhead
VTK XML, zlib	105 MB/s	2.52	Default compression
VTK XML, lz4	401 MB/s	1.47	
VTK XML, lzma	9.93 MB/s	3.10	
VTK HDF (.vtkhdf), lvl0	1733 MB/s	0.93	**No compression**
VTK HDF (.vtkhdf), lvl4	137 MB/s	2.37	Default compression



ParaView Introduction

Science and Technology Facilities Council

CoSeC

IMPERIAL

THE UNIVERSITY of EDINBURGH
School of Engineering

The ParaView Interface

riable
s

on

ces

ies

ation

Time Controls

3D Viewport Controls

Visualisation Layouts can be renders or plots

Additional ParaView Features

Science and Technology Facilities Council

CoSeC

IMPERIAL

THE UNIVERSITY of EDINBURGH
School of Engineering

Automating Workflows With Python

ParaView comes with its own Python interpreter pvpython

- This is an interactive python shell that leverages the VTK backend via interactive and reproducible scripts
- Ideal for mass automation
- Couple with pvbatch which is a self-contained MPI server

	pvpython	pvbatch	Catalyst
Execution Mode	Interactive serial	Parallel batch	In-situ Co-Processing
Primary Use Case	Scripting & Pipeline prototyping	Massive Post-Processing	Zero-I/O Run-time Analysis
Launch Mechanism	Pvpython script.py	Mpirun pvbatch	Compiled with Simulation
Disk I/O Dependency	High (reads raw data)	High (reads raw data)	Minimal/Zero (reads from RAM)

Automating Repetitive Tasks

- ParaView allows you to record your workflows using a trace tool and the macro tool
  - **Traces** are hardcoded to specific sources and filters
  - **Macros** are generalised using script functions such as GetActiveSource()

- Both can be helpful ways of reducing the amount of repetitive tasks

Developing DEM Visualisation *"Helpers"*

Science and Technology Facilities Council

CoSeC

IMPERIAL

THE UNIVERSITY of EDINBURGH
School of Engineering

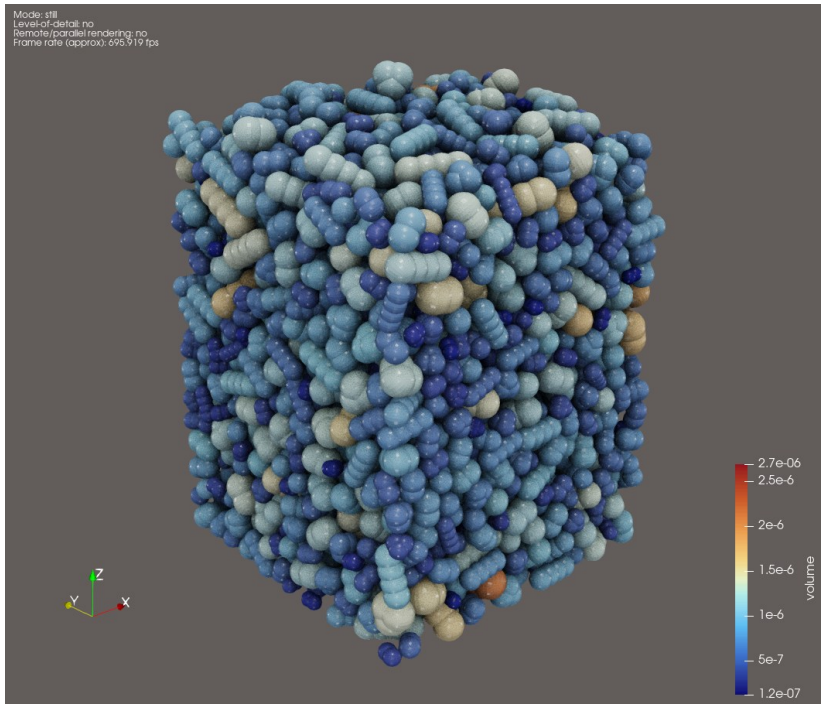
Defining a Simulation for ParaView

ParaView visualisation is dependent on the **visualisation pipeline** to:

- Apply various filters to manipulate the data such as:
  - Applying particle shapes if non-spherical in nature
  - Applying tube filters for visualising contacts
- Various data sources may be required to be loaded to display particles, contacts, geometries, etc
  - May be from separate files

It can be difficult for the new or average DEM user to get past the initial ParaView hurdle

- A *plugin* to automate this process was considered a useful task by the CCC-ParaSolS community
- Engaged **Kitware** to develop this *plugin* with 10 hours of consultancy time
- Several hours of ParaSolS development time to create datasets and refine macros



Plugin or Macro?

In ParaView there are multiple ways to achieve this:

– **Plugins** and **Macros**

A **Macro** is an automation script used to replay GUI actions

A **Plugin** is a compiled package used to fundamentally extend ParaView's underlying data processing, file rendering, or UI capabilities

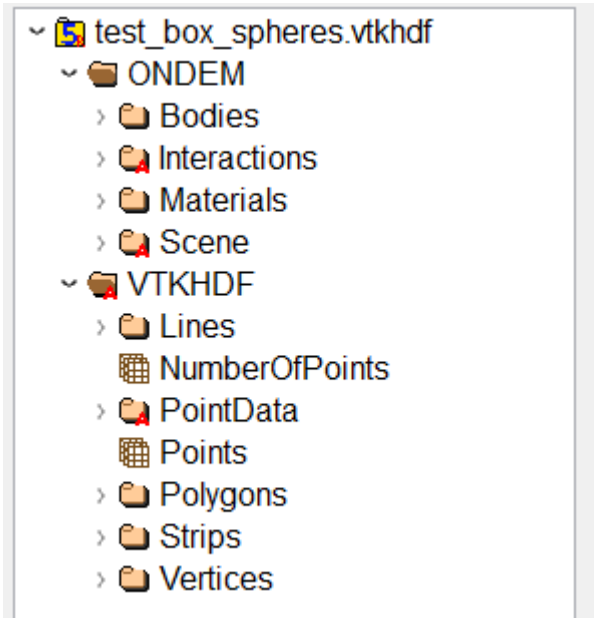
ure	Macro	Plugin
ary Purpose	Automating repetitive workflows	Adding brand-new capabilities
uage	Python (pvpython API)	C++ or Python (with XML wrapping)
pilation	None (interpreted at runtime)	Required (binary libraries or XML scripts)
tegration	Adds a simple clickable button to the toolbar	Adds complex menus, dock panels, and properties
ution Context	Runs entirely on the client application	Can run on both the Client and Server architectures
bilities	Can only run existing filters, change colourmaps, import files	Can add new readers, advanced math filters, integrate third part tools

upporting Open DEM file-format from ON-DEM

Update from WG3: https://on-dem.atlassian.net/wiki/spaces/Index/blog/2026/06/26/863666179/WG3+Meeting+in+Sofia+Expanding+the+Open+DEM+File+Format

Summary:

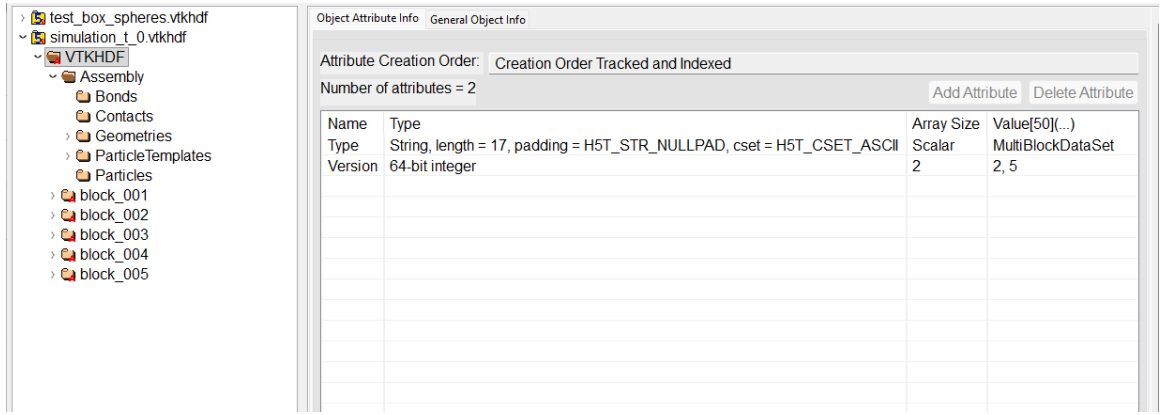
- File format contains two partitions: one for the simulation restart data and metadata and another for visualisation data
- VTKHDF readers only read data in the VTKHDF group
- To allow for a single data file per timestep VTKHDF partition needs to be in the format of a nested **MultiBlockDataSet**



What does this require from the Open file format?

A defined, consistent structure for how ParaView sees the data in the file

- The MultiBlockDataSet structure should contain the required subgroups for particle, contacts, geometry data
  - Note: ONDEM format may choose to use different naming conventions
- A group for storing particle templates to be visualised would also be required



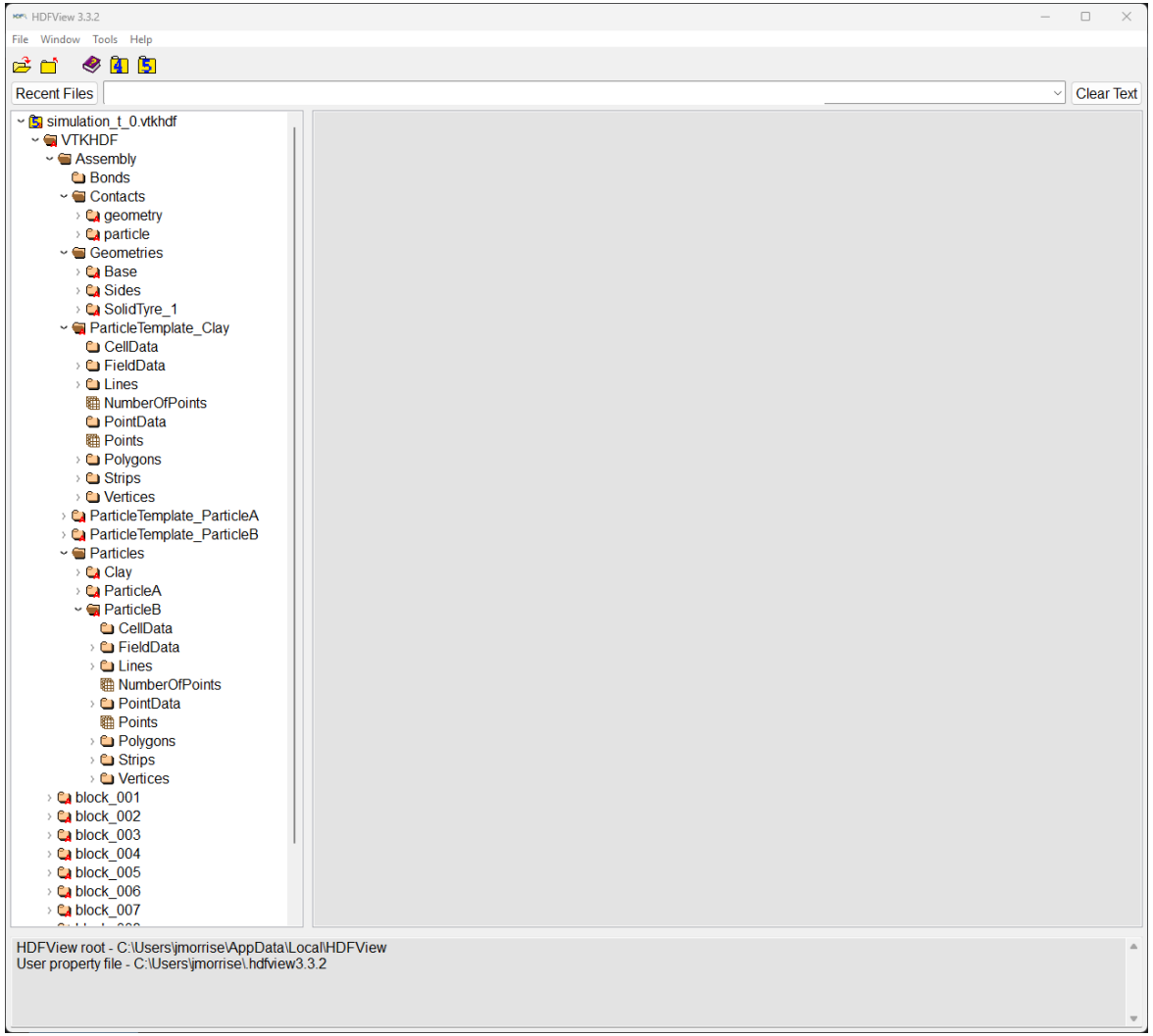
By defining a **MultiBlockDataSet** as the entry point for visualisation support can be extended to XML datasets as well

Proposed Structure

- Single file per timestep
  - ONDEM Group
    - Restart data
    - Only needs to be written on restart points
      - T=0
      - T=-1
  - VTKHDF Group
    - FieldData with TIMEVALUE attribute to store the current time
    - Assembly Group
      - Links
    - VTK block data
- Assembly (*a.k.a simulation*)
  - `particle_templates`
  - `particles`
    - Could be by type subgrouping (optional)
  - `contacts`
    - particle_particle
    - particle_wall
  - `geometries`
    - Individual geometries
    - Should static and dynamic geometries be handled differently with separate groups?
    - Could link to group in previous file by composing as virtual dataset
  - `bonds`
    - Optional subgroup
    - Easier visualisation when working with bonded materials

Defining a Structure within the VTKHDF

- The VTKHDF group contains two types of data
  - The raw VTK blocks
    - Polydata, UnstrucutredGrid or ImageData
    - Flat structure, blocks can have any names
  - The **Assembly** which is where the MultiBlockDataSet structure is written
    - This links to all the written VTK blocks in the file
    - Needs to track order
  - The Assembly effectively creates the user API for the file



Defining a Structure with the VTK XML

- The XML file format is somewhat different
  - Each individual block is a single XML file
- A VTK MultiBlock file (*.vtm*) creates an index to all files that make up the multiblock
- Can easily be added for existing exports
- Does not support file metadata

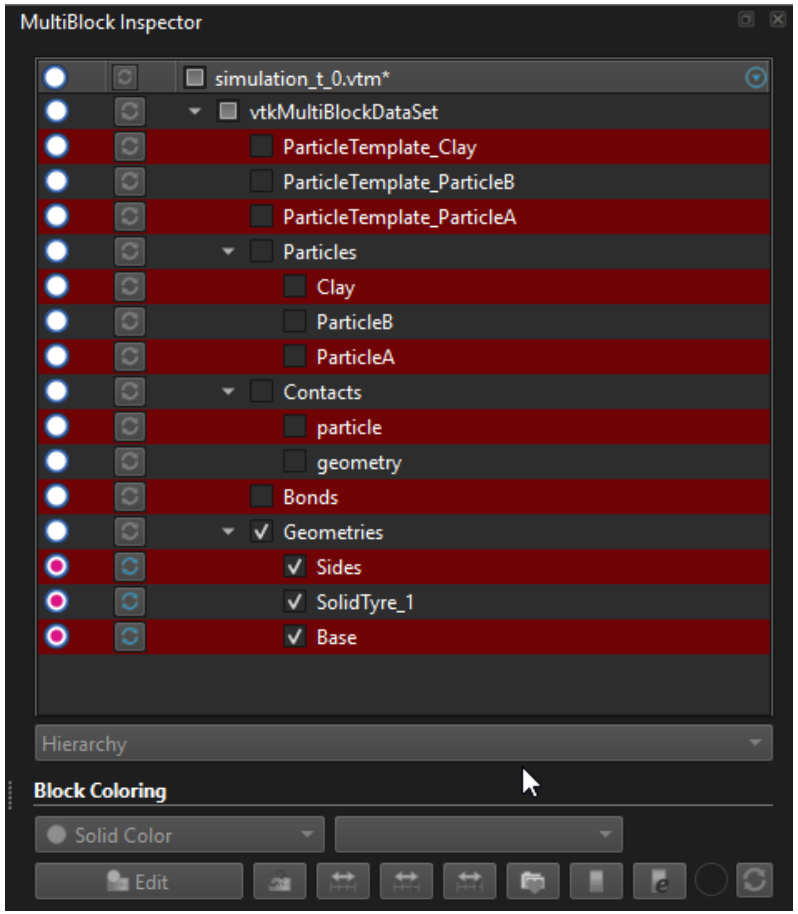
```
<?xml version="1.0"?>
<VTKFile type="vtkMultiBlockDataSet" version="1.0" byte_order="LittleEndian" header_type="UInt64">
  <vtkMultiBlockDataSet>
    <DataSet index="0" name="ParticleTemplate_Clay" file="particle_templates/ParticleTemplate_Clay_t_0.vtp"/>
    <DataSet index="1" name="ParticleTemplate_ParticleB" file="particle_templates/ParticleTemplate_ParticleB_t_0.vtp"/>
    <DataSet index="2" name="ParticleTemplate_ParticleA" file="particle_templates/ParticleTemplate_ParticleA_t_0.vtp"/>
    <Block index="3" name="Particles">
      <Piece index="0" name="Clay">
        <DataSet index="0" file="particles/Clay_t_0.vtp"/>
      </Piece>
      <Piece index="1" name="ParticleB">
        <DataSet index="0" file="particles/ParticleB_t_0.vtp"/>
      </Piece>
      <Piece index="2" name="ParticleA">
        <DataSet index="0" file="particles/ParticleA_t_0.vtp"/>
      </Piece>
    </Block>
    <Block index="4" name="Contacts">
      <Piece index="0" name="particle">
        <DataSet index="0" file="contacts/contacts_pp_particle_t_0.vtp"/>
      </Piece>
      <Piece index="1" name="geometry">
        <DataSet index="0" file="contacts/contacts_pg_geometry_t_0.vtp"/>
      </Piece>
    </Block>
    <Block index="5" name="Bonds">
    </Block>
    <Block index="6" name="Geometries">
      <DataSet index="0" name="SolidTyre_1" file="geometries/SolidTyre_1_t_0.vtp"/>
      <DataSet index="1" name="Sides" file="geometries/Sides_t_0.vtp"/>
      <DataSet index="2" name="Base" file="geometries/Base_t_0.vtp"/>
    </Block>
  </vtkMultiBlockDataSet>
</VTKFile>
```

MultiBlockDataSet in ParaView

A defined, consistent structure for how ParaView sees the data in the file

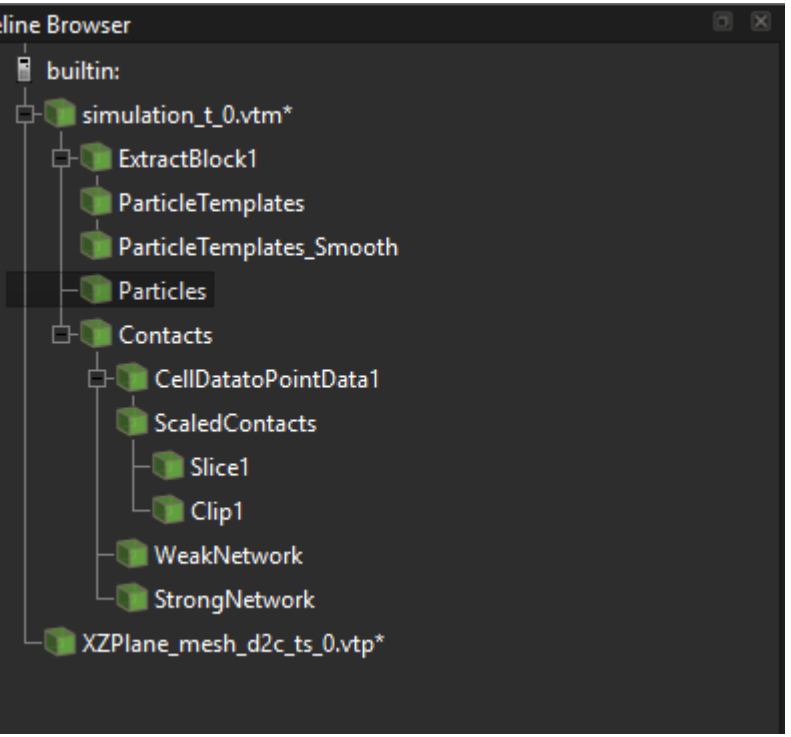
– The **MultiBlockdataSet** structure

```
version="1.0"?>
ile type="vtkMultiBlockDataSet" version="1.0" byte_order="LittleEndian" header_type="UInt64">
<MultiBlockDataSet>
DataSet index="0" name="ParticleTemplate_Clay" file="particle_templates/ParticleTemplate_Clay_t_0.vtp"/>
DataSet index="1" name="ParticleTemplate_ParticleB" file="particle_templates/ParticleTemplate_ParticleB_t_0.vtp"/>
DataSet index="2" name="ParticleTemplate_ParticleA" file="particle_templates/ParticleTemplate_ParticleA_t_0.vtp"/>
Block index="3" name="Particles">
 <Piece index="0" name="Clay">
  <DataSet index="0" file="particles/Clay_t_0.vtp"/>
 </Piece>
 <Piece index="1" name="ParticleB">
  <DataSet index="0" file="particles/ParticleB_t_0.vtp"/>
 </Piece>
 <Piece index="2" name="ParticleA">
  <DataSet index="0" file="particles/ParticleA_t_0.vtp"/>
 </Piece>
/Block>
Block index="4" name="Contacts">
 <Piece index="0" name="particle">
  <DataSet index="0" file="contacts/contacts_pp_particle_t_0.vtp"/>
 </Piece>
 <Piece index="1" name="geometry">
  <DataSet index="0" file="contacts/contacts_pg_geometry_t_0.vtp"/>
 </Piece>
/Block>
Block index="5" name="Bonds">
/Block>
Block index="6" name="Geometries">
 <DataSet index="0" name="SolidTyre_1" file="geometries/SolidTyre_1_t_0.vtp"/>
 <DataSet index="1" name="Sides" file="geometries/Sides_t_0.vtp"/>
 <DataSet index="2" name="Base" file="geometries/Base_t_0.vtp"/>
/Block>
tkMultiBlockDataSet>
File>
```

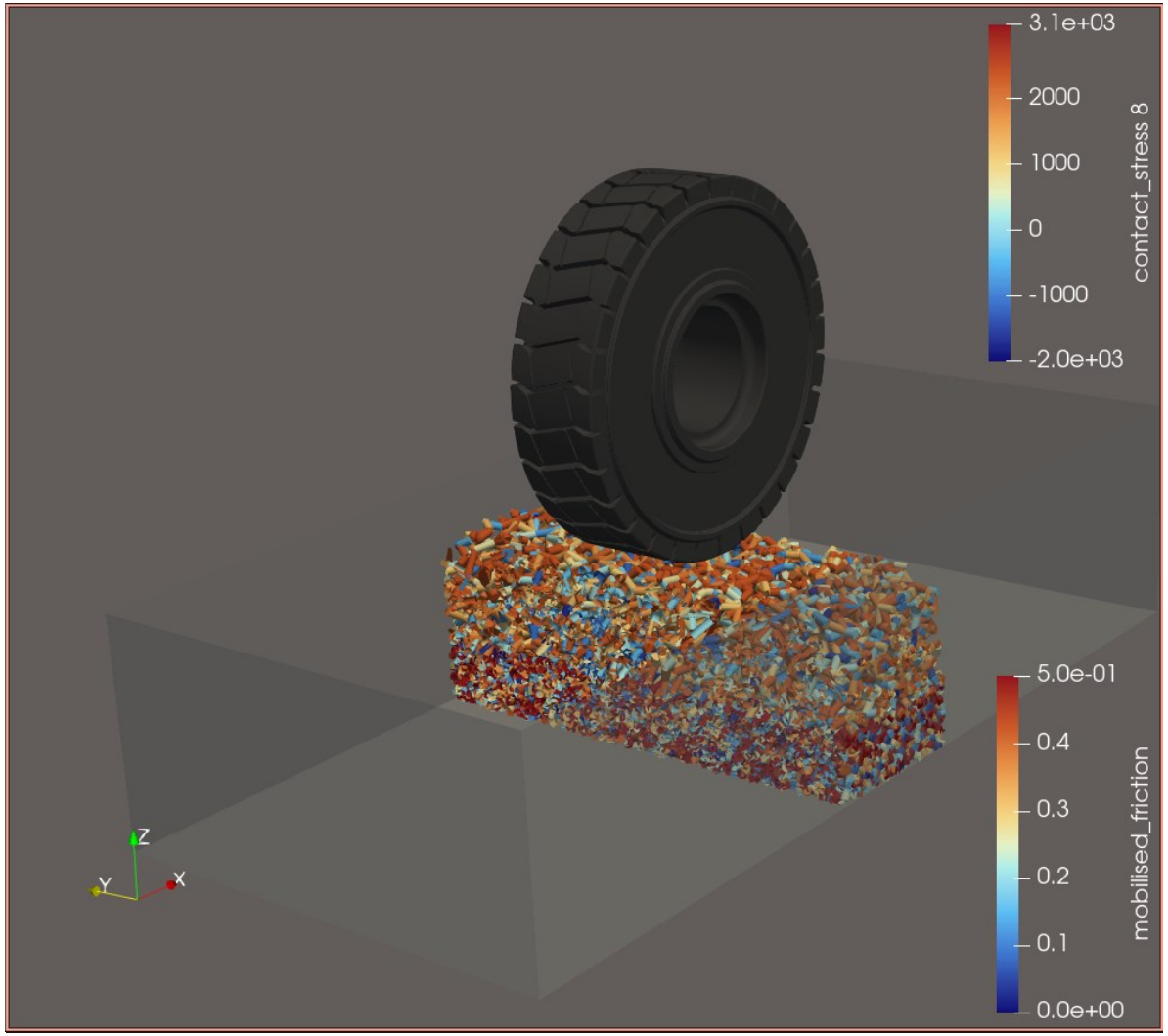


low to get from Multiblock to Visualisation?

Answer: Pipeline and filters



Question: how do I know what filters to use?



low to simplify this process?

Develop a simple plugin *or* macro to automate this process

To be able to do this need to have a **predefined** MultiBlockDataSet structure that is provided

- Will not work without this enforced layout

Will load particle, contact and geometry data from file(s)

- Extract blocks if required
- For non-spherical particles a particle glyph needs to be automatically loaded for each particle type
- For contacts, scaling by force or other values requires converting the contact data to point data to use the “vary_by_*” options in ParaView
- Apply as 3D Glyph from pipeline
- Does not support ray tracing, only PBR

olving the Static Geometry Problem

XML multiblock files were just a link to a single VTK file which is used to create a multiblock (.vtm)

- This could be nested
- Allow linking to a constant file every timestep – no need to write large geometries every timestep

This can be replicated using the virtual datasets functionality of hdf5 files

- Some data would be written out to static vtkhdf files which are linked by the main file
- Could also be used for templates
- Allows one point of loading into ParaView
- Can also replicate the PVD file behaviour by creating an index to all timesteps
  - Not necessary as ParaView has a timeseries reader based on filenames

xample Datasets

There are multiple ways of representing data so the following datasets have been created for testing the new macros

- Polydisperse spheres
  - No particle template required but requires a radius or diameter
  - A single sphere template* is provided and a scale field is required
- Multisphere particles
  - All particle templates* are required
- Polyhedral particles
  - All particle templates* are required

Datasets are provided as both **XML** and **VTKHDF** format

- **What is a template?**
  - For polyhedral particles this is in the form of a triangulated surface mesh
  - Multispheres/Spheres have two possible methods
    - Triangulated surface mesh
    - List of sphere positions and radii

utorial Datasets: Download one now!

Three different datasets to work with:

- Polydisperse spheres
- Polydisperse multi-spheres with multiple shapes
- Polydisperse polyhedrals with multiple shapes

Available in both **XML** and **VTKHDF** formats

Download from here:

- https://zenodo.org/communities/parasols/records?q=&l=list&p=1&s=10&sort=newest

Requirements

PolyData or **UnstrucutredGrid**

- PolyData is sufficient for DEM data, no need for extra complexity of cell types
- ParaView technically doesn't care which type is in a multiblock – can mix

Required data

- Spheres require a `radius` or `diameter` field
- A `quaternions` filed is required for non-spherical particle orientations and glyph display
- A `scale` field is required for glyph scaling
- A `glyph_id` field is required for glyph table mechanism
  - This is not necessarily the particle type ID, but the index in which the templates for the particle data are written to the MultiBlockDataSet (Blocks have an index)
  - Can be the same

ParaView Macros

Three macros developed with assistance from Kitware

Additional helper macros for splitting views and re-apply glyphs

Available at: https://git.ecdf.ed.ac.uk/parasols/paraview-macros

Basic installation instruction and description of some settings that can be customised

Clone or download now if you want to follow along

Visualise Geometries

- Extract all geometries in the simulation dataset to the Geometries block
- Sets the representation to the specified type (surface by default)
- Set the opacity to default value (default 0.5)

arameter	Description
EFAULT_REPRESENTATION	Default geometry representation to be used in ParView (e.g. "Surface", "Surface With Edges", "Wireframe", "Outline"). Default: "Surface"
EFAULT_OPACITY	Level of opacity to be applied to all geometries. Default: 0.5

Visualise Particles Macro

Extracts the Particles block (for all timesteps) and ParticleTemplates block (from the first timestep)

- If the template is a list of sphere positions and radii, the multi-sphere glyphs are created
  - Sphere resolution is controlled by `SPHERE_RESOLUTION` variable
- Colours the particles - either a solid colour or by a variable
  - If an array is specified it finds the first timesteps that particles exists and set the colour range based on that data

You should always **choose the most appropriate ranges for your data** by manually specifying a range of interest or by rescaling across all timesteps to find the absolute max and min values

Visualise Particles Macro

Extracts the Particles block And ParticleTemplates block from the first timestep

Parameter	Description
SPHERE_RESOLUTION	The phi/theta resolution that controls the number of facets on the sphere glyph. Default: 24
COLOR_ARRAY	Specifies how the particles are coloured. Accepts None or the string name of variable field in the dataset. Default: "volume"
COLOR_MAP	String name of the colour map preset used when COLOR_ARRAY is set (e.g. "Viridis", "Inferno", "Plasma", "Magma", "Cividis", "Fast", "Turbo", "Cool to Warm", "Cool to Warm (Extended)", "Rainbow Desaturated"). Must match a preset name ParaView recognises; run print(ListColorPresetNames()) in the shell to list options. Default: "Viridis"
SOLID_COLOR	If no COLOR_ARRAY is specified, a single solid colour will be used for all particles. RGB colour format. Default: [1,1,0]

Visualise Contacts

Extract contacts Block from simulation dataset

Apply tube filter to visualise contacts

- Read a specified timestep to get some force data
- Size the tube based on the forces and ratio of forces
- Calculate the mean value and create a Weak and Strong network for visualisation
- Set colour maps for weak and strong based on force ranges

You should always **choose the most appropriate ranges for your data** by manually specifying a range of interest or by rescaling across all timesteps to find the absolute max and min values

Visualise Contacts

ameter	Description
GNITUDE_RANGE_TIME_STEP	Timestep to read for setting contact information values. Default: 2
GNITUDE_RANGE_TIME_STEP_LAST	Bool flag as to whether the last timestep should be used for setting contact variables. If False, the specified timestep is used. Default: True
E_NUM_SIDES	Number of faces used in rendering the tube. More faces will require more memory but give a more "round" tube". Default: 12
TICLE_SIZE_FACTOR	Maximum contact tube size as a ratio of the average particle size. To prevent the largest contact forces being rendered much larger than particles. Default: 0.5
IUS_FACTOR_RATIO_CAP	Limit on the size ratio of tube diameters for largests and smallest forces. Will attempt to use the real force ratio but will use this cap if that ratio is very large. Sets the Tube Radius value. Default 25.
Y_RADIUS_BY_	Specifies how the tubes are scaled. Can be any valid ParaView Option ("By Vector Norm", "By Vector", "By Scalar", "By Absolute Scalar"). Default: "By Vector Norm"
N_AGGREGATION_LEVEL	Controls how the Weak/Strong threshold is computed with regards to to particle and geometry contacts. Default: "top_level". Options: - "global": one mean across all contacts combined (particle + geometry) -"top_level": one mean per top-level block (e.g. separate means for "particle" and "geometry", combining their sub-blocks) -"leaf": one mean per leaf block (e.g. separate means for each of A-A, A-B, B-B, A-top, A-bottom, B-top, B-bottom)
WORK_COLOR_MAP	Default colour map to use on the strong and weak contact networks. Default: "Inferno"
ERT_COLOR_MAP	Bool to specify inversion of selected colour map. Default: True

Additional Helpers

Macro to re-apply particle glyphs for any further filtered particle data

- Filters usually just return the point data so *"re-glyphing"* would be required in most cases

Clone view macro

- Creates a cloned view in a new layout tab with all display setting preserved

Split and Clone Macros

- Split view either vertically or horizontally and clone with all display settings preserved
- Can be applied recursively

Demonstration

Download datasets and scripts if you want to follow along!

New Macros

New macros could be developed to do many other things such as plotting or further complex pipelines

Please contributes any useful ones you have wither on the forum or the repository

Repository:

- ParaSolS repo?
- Maintainers who want to be involved, let me know

ummary

Any Macro/Plugin for visualisation requires a concrete multiblock structure to be defined

- Proposal made & prototype created

Example datasets to cover both file formats and 5 possible use cases created

Macros for visualising particles, contacts and geometries developed

- Additional helper macros also developed

Questions?

ParaSolS
Particulate Solids Simulations

parasols@ed.ac.uk

https://www.ccc-parasols.ed.ac.uk/

Discourse https://parasols.discourse.group/

https://www.linkedin.com/company/ccc-parasols/