ParaView

ParaView is an open-source, multi-platform data analysis and visualization application built on top of VTK, the Visualization Toolkit. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of petascale as well as on laptops for smaller data. ParaView is an application framework as well as a turn-key application. It has become an integral tool in many national laboratories, universities and industry. ParaView has also won several awards related to high performance computation.

This will be a very memory and computationally intensive application. Paraview largely consists of two sections, data processing and rendering. Data processing consists of reading and filtering the data, and rendering consists of visually displaying the data. MPI can be used to parallelize the data processing, with each process containing its own data flow on a piece of the total dataset. The more compute nodes we have the less memory bound the application will be, but we must remain conscious of the ghost cells and extra memory that can be required if a dataset is split too much. GPUs can be used for the rendering step (OpenGL) in either a client based, live, interactive session or data can be rendered straight to a video or other visual representation like a graph or comparison view. GPUs can also be used to accelerate some filters through an accelerated algorithms plugin, although not every filter can be accelerated. My guess for a dataset will be one which is temporal, large, and finely detailed. We would likely have to output a video through applying different filters which will be compared to a ground truth video at the competition. With this app a wide understanding of memory and filters must be known because of the wide possibility of input datasets and potential analyses used.

Data Flow Paradigm

SOURCE → FILTER(S) → SINK

An input file or programmatically created data set is referred to as a source
Intermediate algorithms are called filters

    Ex. clipping, slicing, generating contours from the data, computing derived quantities
    (see Filters section of write-up)

A visual is considered a sink

    Can be interactive or static (in the creation of a video)

Paraview supports large tile displays and even 3D head tracking immersive displays
Paraview contains a parallel rendering library called IceT which renders in pieces and joins together

IceT's algorithm is scalable and independent of the amount of data used

Program Interface

When installed and compiled, the program will consist of several executables:
paraview: creates a GUI
pvpython: displays a python shell, same ability as "paraview" but allows for scripting
pvbatch: python interpreter for batch processes, supports MPI
pvserver: for remote visualization, you can make other executables connect and execute remotely
This is what we will use for our cluster
pvdataserver and pvrenderserver: represent the data analysis and rendering parts of pvserver

Filter options

These are dataset specific!
Examples:
Shrink reduce by a common factor
Append: combine datasets
Clip: Extract a sub-set from the data-set
Inside out filter: invert function (can force dataset to be unstructured, resulting in more memory)
Slice: only view a D-1 space view of the dataset
Transform: used to rotate, translate, scale
(This is different that the rendering rotate, which does not change the actual data)
Reflect: Mathematical reflection
Warp by vector/scalar: Change the presentation of the object physically based on values of vectors or scalars associated with the data (ex. Hotter regions will appear bigger)
Glyph: Adds visual vectors to rendering
Stream tracer: Generates streams for vector fields
Probe: Display various data points/cells
Calculator: calculate new data points based on existing points
Gradient: Calculate gradients
Mesh quality: Used to describe the fitness of each data point
Accelerated Algorithms will accelerate some of these filters on a GPU
Make a plot of a variable value over a line

Data

Uses a VTK backend (Visualization Toolkit) for memory
Paraview stores data through something called a mesh, consisting of points and cells.
An example dataset will have a set of points connected (by graph theory definition) via proximity, which will then create a closed surface (see the squares below, each vertex is a point, the square itself is a surface). This surface is considered a cell. Attributes/data can be stored with respect to the points or the cell. Cell centered attributes are assumed to be constant across the cell and point attributes may vary. Interpolation functions are used for displaying point data attributes visually in spaces between the data points (see how red blends into blue below).

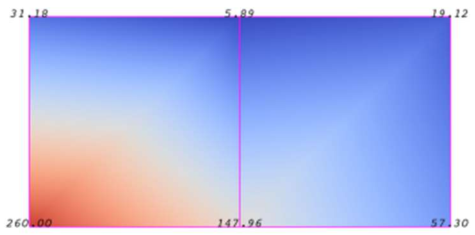


Figure 3.2: Point-centered attribute in a data array or field.

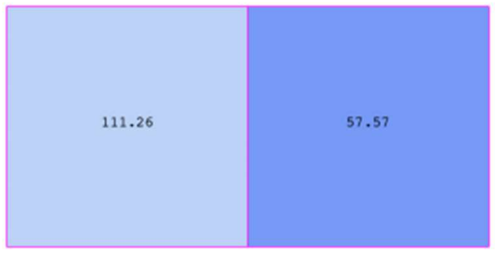
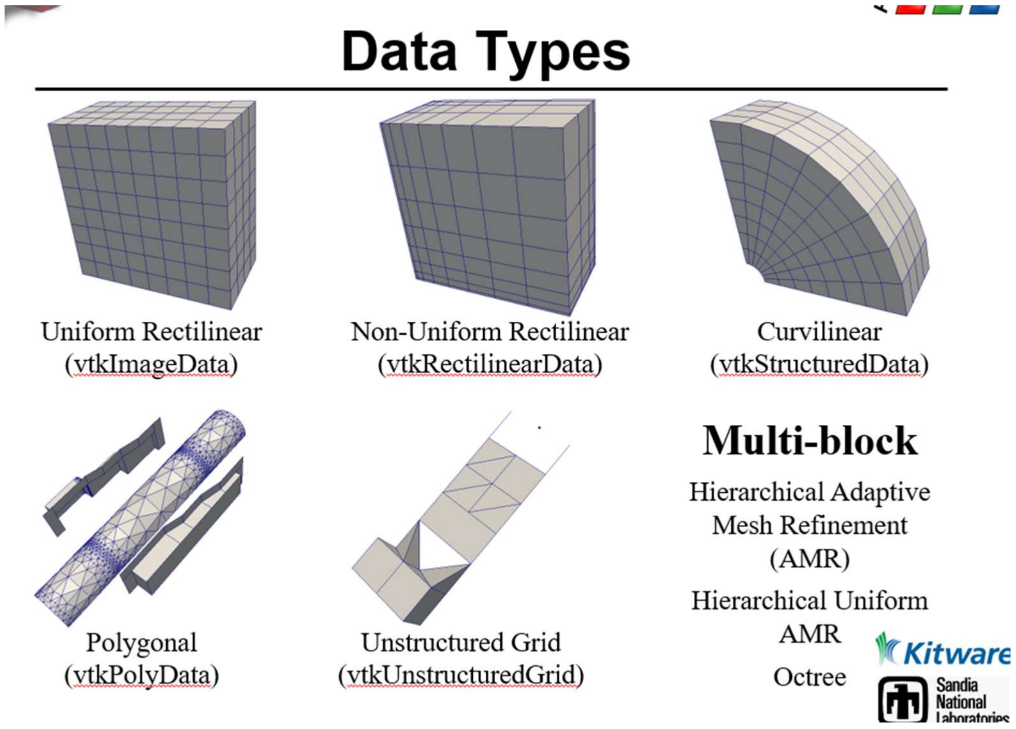


Figure 3.3: Cell-centered attribute.

As far as actual memory storage, data is contained in the data structure called a tuple. There is a tuple for each point and each cell corresponding to the dimensionality or feature space of the dataset.


Data Types
Uniform Rectilinear (vtkImageData)
Non-Uniform Rectilinear (vtkRectilinearData)
Curvilinear (vtkStructuredData)
Polygonal (vtkPolyData)
Unstructured Grid (vtkUnstructuredGrid)
Multi-block
Hierarchical Adaptive Mesh Refinement (AMR)
Hierarchical Uniform AMR
Octree
Kitware
Sandia National Laboratories

A coordinate system should always be used for data. An unstructured grid compared to a coordinate reference grid will use significantly more storage for identifying each point's position.

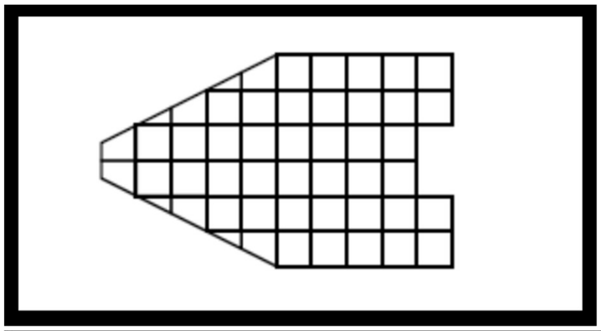
Parallelization

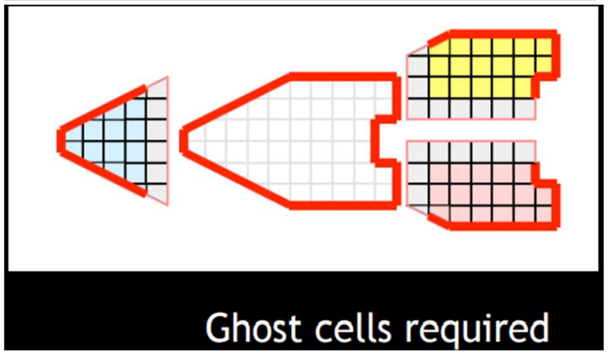
Data cutting is very important so that each process has access to all needed values

A split diagonally across a dataset may split a cell in half, and cell data will be lost/interpolation data will be incorrect

One must preserve the connectivity of data when separated using ghost cells

Ghost cell: potentially unnecessary neighbor cell added in set to ensure connectivity





Random tiny partitioning would result in many cell splits and therefore many ghost cells

→ inefficient calculations

Partitioning the data potentially transforms it into unstructured data

→ much higher memory usage

Parallelization will be a game of balancing data splitting for each process and the amount of ghost cells/extra memory and calculations created

D3 is a feature within Paraview used for load balancing and ghost cell adjusting in unstructured data

Other Cool Aspects of Paraview:

Web visualization: ParaViewWeb

Used to embed real time 3D rendering within a website

Tracing is used to track GUI instructions in Python scripts

Paraview uses automated wrapping. Basically

In-situ analysis: Catalyst

-In-situ analysis is the process of analyzing the data while simultaneously rendering because the data is so large it would be unreasonable to do both separately

-It is already common for simulations to discard most of what they compute in order to minimize time spent on I/O. As we enter the exascale age the problem of scarce I/O

capability continues to grow. Since storing data is no longer viable for many simulation applications, data analysis and visualization must now be performed *in situ* with the simulation to ensure that it is running smoothly and to fully understand the results that the simulation produces. Catalyst is a lightweight version of the ParaView server library that is designed to be directly embedded into parallel simulation codes to perform *in situ* analysis at run time.
-This sounds complicated and is very potentially something that they could throw at us at the competition

Origins of Paraview:
Started by Kitware Inc. and LANL through a federal grant