Visualizing BOUT++ data with VisIt

BOUT++ Workshop 2018
August 17, 2018

Eric Brugger
Overview

- Overview of VisIt (10 min)
- BOUT data and VisIt (5 min)
- Tutorial (45 min)
VisIt is an open source, richly featured, turnkey application for scientific data analysis and vis

- Utilizes a plugin architecture for extensibility
  - Rendering methods
  - Data manipulation
  - Data readers
- Utilizes a client/server architecture
- Provides a scalable/parallel server
- Multiple interfaces
  - A graphical user interface
  - A python scripting interface
  - A java library interface
- Multi-platform support – Unix, Windows, Mac
Major use cases

Data Exploration

Comparative Analysis

Visual Debugging

Presentations

Quantitative Analysis
VisIt employs a parallelized client-server architecture.
Getting your data into VisIt

- VisIt requires BOUT++ data to be stored as a NETCDF file
- The data must be stored with the appropriate conventions to be recognized as a BOUT++ file
  - If you use the incorrect conventions then VisIt will treat the file as a generic NETCDF file and you will only be able to render the data as multi-dimensional arrays.
  - When you store it using the correct conventions VisIt will automatically transform it into your real geometry
The format of the solution file

Grid file

Solution files

Necessary variables

Must be in t, x, y, z order

Name of grid file

Lawrence Livermore National Laboratory
The format of the grid file

```c
edge83\{brugger\}43: ncdump -h D3D_144382.02510_x516y64_psi080to120mer.nc
netcdf D3D_144382.02510_x516y64_psi080to120mer {
    dimensions:
    y = 64;
    x = 516;
    x2 = 3;
    x3 = 2;
    variables:
    short nx;
    short ny;
    int 1xseps1;
    short ixseps2;
    int jyseps1_1;
    int jyseps1_2;
    int jyseps2_1;
    int jyseps2_2;
    int ny_inner;
    float dx(x, y);
    float dy(x, y);
    float ShiftAngle(x);
    float zShift(x, y);
    float pol_angle(x, y);
    double ShiftTorsion(x, y);
    float Rxy(x, y);
    float Zxy(x, y);
    float Bpxy(x, y);
    float Btxy(x, y);
    float Bxy(x, y);
    float hthe(x, y);
}
```
When things go wrong

- Use “ncdump -h field_file” to check that the required variables are present
- Use “ncdump -v gridname field_file” to check that the gridname is correct
- Use “ncdump -h grid_file” to check that the required variables are present
- Use “visit -debug 4” and look at the resulting “vlog” files for a message that may indicate the problem.
VisIt automatically detects the standard grid configurations (circular, 1X, 2X)

Circular grid  1 X grid  2 X grid
VisIt adaptively refines the grid in the x and y directions based on zShift

Large jump in zShift leads to a lot of refinement

zShift  Original grid  Refined grid
VisIt will break the grid into several structured pieces

Circular grid

1 X grid

2 X grid
VisIt creates a single block in the radial direction
VisIt creates divides the blocks into 180 zones radially and resamples the variable based on zshift.
This is a different approach than in previous releases of VisIt

- Doesn’t create any large aspect zones, which caused rendering artifacts that had to be worked around

- Uses less memory so that it is more interactive
  - Previous version required running in parallel with 16 – 32 cores for good performance
  - Current version runs well with 4 cores and even does well in serial
Tutorial information

- The tutorial assumes some familiarity with VisIt.
  - The following tutorial goes over the basics of VisIt.

- Link to the tutorial content
Manuals & Other Documentation

- Users Manual

- Python Interface
  - http://visit.ilight.com/svn/visit/trunk/releases/2.10.0/VisItPyth onManual.pdf

- Several additional introductory tutorials
How to get help

- VisIt Users Mailing List
  - Address: visit-users@elist.ornl.gov
  - Info: https://elist.ornl.gov/mailman/listinfo/visit-users
  - Archive: https://elist.ornl.gov/pipermail/visit-users/

- VisIt Users Wiki
  - http://www.visitusers.org