Visualization and Analysis of HPC Simulation Data with VisIt

ATPESC 2020
Monday August 3rd, 2020

Cyrus Harrison (cyrush@llnl.gov)
Outline

- VisIt Project Introduction (~30 min)

- Hands-on: (~1 hour)
  - Guided tour of VisIt
  - Visualization of an Aneurysm (Blood Flow) Simulation
Tutorial Resources

- VisIt 3.1.2
  - https://github.com/visit-dav/visit/releases

- Tutorial Materials

- How to get in touch
  - GitHub: https://github.com/visit-dav/visit
  - Email List: visitusers@ornl.gov
  - Cyrus: cyrush@llnl.gov
Tutorial Data Acknowledgements

Aneurysm Simulation Dataset
Simulated using the LifeV (http://www.lifev.org/) finite element solver.
Available thanks to:
— Gilles Fourestey and Jean Favre
  Swiss National Supercomputing Centre (http://www.cscs.ch/)

Potential Flow Simulation Dataset
Simple tutorial simulation built using MFEM (https://mfem.org/)
Available thanks to:
— Aaron Fisher and Mark Miller, LLNL
VisIt Project Introduction
**visit**

/ˈvizɪt/

**verb**

1. go to see and spend time with (someone) socially.
   "I came to visit my grandmother"
   **synonyms**: call on, call in on, pay a call on, pay a visit to, pay someone a call, pay someone a visit, go to see, come to see, look in on;  

2. inflict (something harmful or unpleasant) on someone.
   "the mockery **visited upon** him by his schoolmates"
   **synonyms**: happen to, overtake, befall, come upon, fall upon, hit, strike
   "it is hard to imagine a greater psychological cruelty visited on a child"

**noun**

1. an act of going or coming to see a person or place socially, as a tourist, or for some other purpose.
   "a visit to the doctor"
   **synonyms**: social call, call
   "after reading the play she paid a visit to the poet"
VisIt is an open source, turnkey application for data analysis and visualization of mesh-based data

- Production end-user tool supporting scientific and engineering applications.
- Provides an infrastructure for parallel post-processing that scales from desktops to massive HPC clusters.
- Source released under a BSD style license.

Pseudocolor plot of Density
(27 billion element dataset)
VisIt supports a wide range of use cases

Data Exploration

Comparative Analysis

Quantitative Analysis

Presentation Graphics

Visual Debugging
VisIt provides a wide range of plotting features for simulation data across many scientific domains.

- Streamlines / Pathlines
- Vector / Tensor Glyphs
- Pseudocolor Rendering
- Volume Rendering
- Molecular Visualization
- Parallel Coordinates
VisIt is a vibrant project with many participants

- The VisIt project started in 2000 to support LLNL’s large scale ASC physics codes.
- The project grew beyond LLNL and ASC with development from DOE SciDAC and other efforts.
- VisIt is now supported by multiple organizations:
  - LLNL, LBNL, ORNL, Univ of Oregon, Univ of Utah, Intelligent Light, ...
- Over 100 person years of effort, 1.5+ million lines of code.
VisIt provides a flexible data model, suitable for many application domains

- **Mesh Types**
  - Point, Curve, 2D/3D Rectilinear, Curvilinear, Unstructured
  - Domain Decomposed, AMR
  - Time Varying
  - Primarily linear element support, limited quadratic element support

- **Field Types**
  - Scalar, Vector, Tensor, Material Volume Fractions, Species
The VisIt team releases binaries for several platforms and a script that automates the build process

“How do I obtain VisIt?”

- **Use an existing build:**
  - For your Laptop or Workstation:
    - Binaries for Windows, OSX, and Linux (RHEL + Ubuntu):
      (https://wci.llnl.gov/simulation/computer-codes/visit/executables)
  - VisIt on ALCF’s Cooley:
    - https://www.alcf.anl.gov/user-guides/visit-cooley
  - Several other HPC centers have VisIt installed

- **Build VisIt yourself:**
  - “build_visit” is a script that automates the process of building VisIt and its third-party dependencies. (also at: https://wci.llnl.gov/simulation/computer-codes/visit/executables)
  - Fledgling support for building via spack (https://github.com/spack/spack)
“How do I get my data into VisIt?”

- The *PlainText* database reader can read simple text files (CSV, etc)

- Experiment with the *visit_writer* utility:

- Write to a commonly used format:
  - *VTK, Silo, Xdmf, PVTK*

- We are ramping up support for Mesh-based data in Conduit Blueprint:

VisIt’s infrastructure provides a flexible platform for custom workflows

- **C++ Plugin Architecture**
  - Custom File formats, Plots, Operators
  - Interface for custom GUIs in Python, C++ and Java

- **Python Interfaces**
  - Python scripting and batch processing
  - Data analysis via Python Expressions and Queries

- **In-Situ Coupling**
  - VisIt’s *Libsim* library allows simulation codes to link in VisIt’s engine for in situ visualization
VisIt is used as a platform to deploy visualization research

- **DOE Research Collaborations**

- **Research Focus Areas**
  - Light weight In Situ Processing
  - Node Level Parallelism
  - Distributed Memory Parallel Algorithms

**Scaling research:** Scaling to 10Ks of cores and trillions of cells.

**Algorithms research:** How to efficiently calculate particle paths in parallel.

**Methods research:** How to incorporate statistics into visualization.
VisIt 3.0 (April 2019) included major updates to our software development process

- We migrated our source repo from *svn* at NERSC to *git* on GitHub and our issue tracking from an ORNL Redmine instance to GitHub
  - [https://github.com/visit-dav/visit](https://github.com/visit-dav/visit)

- We ported our legacy docs to Sphinx, now hosted on Read the Docs
VisIt uses MPI for distributed-memory parallelism on HPC clusters

Full Dataset
(27 billion total elements)

3072 sub-grids
(each 192x129x256 cells)

We are enhancing VisIt’s pipeline infrastructure to support threaded processing and many-core architectures
VisIt employs a parallelized client-server architecture

Client Computer

- VisIt Viewer
- VisIt GUI
- VisIt CLI
- Python Clients
- Java Clients

Parallel HPC Cluster

- VisIt Engine
- Data Plugin
- Data

- VisIt Engine
- Data Plugin
- Data

MPI

(Files or Simulation)
VisIt automatically switches to a scalable rendering mode when plotting large data sets on HPC clusters.

In addition to scalable surface rendering, VisIt also provides scalable volume rendering.
DOE’s visualization community is collaborating to create open source tools ready for Exascale simulations

Addressing node-level parallelism

- VTK-m is an effort to provide a toolkit of visualization algorithms that leverage emerging node-level HPC architectures
- We are also exploring using VTK-m and DIY to share more distributed-memory infrastructure across projects

Addressing I/O gaps with in-situ

- There are several efforts focused on in-situ infrastructure and algorithms

  * [ParaView/VisIt](http://www.paraview.org/in-situ)
  * [ALPINE](http://alpine.dsscale.org)
  * [VisIt LibSim](https://visit.llnl.gov)
  * [DIY](https://github.com/diatomic/diy)
  * [Ascent](https://github.com/Alpine-DAV/ascent)
  * [Sensei in situ](http://www.sensei-insitu.org)
The VisIt team is investing in Conduit and Ascent to create next generation in situ infrastructure.

**Conduit**
Intuitive APIs for in-memory data description and exchange

[http://software.llnl.gov/conduit](http://software.llnl.gov/conduit)

**Ascent**
Flyweight in-situ visualization and analysis for HPC simulations

[http://ascent-dav.org](http://ascent-dav.org)
Conduit provides intuitive APIs for in-memory data description and exchange

- Provides an intuitive API for in-memory data description
  - Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs

- Provides common conventions for exchanging complex data
  - Shared conventions for passing complex data (*e.g.*, *Simulation Meshes*) enable modular interfaces across software libraries and simulation applications

- Provides easy to use I/O interfaces for moving and storing data
  - Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)

Conventions for sharing in-memory mesh data

http://software.llnl.gov/conduit
http://github.com/llnl/conduit

Website and GitHub Repo
Ascent is an easy to use flyweight in situ visualization and analysis library for HPC simulations

- Easy to use in-memory visualization and analysis
  - Use cases: *Making Pictures, Transforming Data*, and *Capturing Data*
  - Young effort, yet already supports most common visualization operations
  - Provides a simple infrastructure to integrate custom analysis
  - Provides C++, C, Python, and Fortran APIs

- Uses a flyweight design targeted at next-generation HPC platforms
  - Efficient distributed-memory (MPI) and many-core (CUDA or OpenMP) execution
    - Demonstrated scaling: In situ filtering and ray tracing across **16,384 GPUs** on LLNL's Sierra Cluster
  - Has lower memory requirements than current tools
  - Requires less dependencies than current tools (ex: no OpenGL)
    - Builds with Spack [https://spack.io/](https://spack.io/)

Visualizations created using Ascent

Extracts supported by Ascent

[http://ascent-dav.org](http://ascent-dav.org)

[https://github.com/Alpine-DAV/ascent](https://github.com/Alpine-DAV/ascent)

Website and GitHub Repo
VisIt’s Visualization Building Blocks
VisIt’s interface is built around five core abstractions

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data
Examples of VisIt Pipelines

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data

- **Database**: Open a database, which reads from a file (Example: Open file1.hdf5)
- **Plot**: Make a plot of a field in the database (Example: Pseudocolor plot of density)
Examples of VisIt Pipelines

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data

**Database**
- Open a database, which reads from a file (Example: Open file1.hdf5)

**Operator**
- Apply an operator to transform the data (Example: Slice operator)

**Plot**
- Make a plot of the result (Example: Pseudocolor plot of density)
Examples of VisIt Pipelines

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data

**Database**
- Open a database, which reads from a file (Example: Open file1.hdf5)

**Operator 1**
- Apply an operator to transform the data (Example: Slice operator)

**Operator 2**
- Apply a second operator to transform the data (Example: Elevate operator)

**Plot**
- Make a plot of the result (Example: Pseudocolor plot of density)
Examples of VisIt Pipelines

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data

**Database**: Open a database, which reads from a file (Example: Open file1.hdf5)

**Expression**: Create derived quantity from existing fields (Example: \( \text{speed} = \text{magnitude}(\text{velocity}) \))

**Plot**: Make a plot of the result (Example: Pseudocolor plot of \( \text{speed} \))
Examples of VisIt Pipelines

- **Databases**: Read data
  - Open a database, which reads from a file (Example: Open file1.hdf5)

- **Plots**: Render data
  - Make a plot of a field in the database (Example: Pseudocolor plot of density)

- **Operators**: Manipulate data
  - Extract quantitative information (Example: integrate density to find mass)

- **Expressions**: Generate derived quantities

- **Queries**: Summarize data
Examples of VisIt Pipelines

- **Databases**: Read data
- **Plots**: Render data
- **Operators**: Manipulate data
- **Expressions**: Generate derived quantities
- **Queries**: Summarize data

**Database**
- Open a database, which reads from a file (Example: Open file1.hdf5)

**Expression**
- Create derived quantity from existing fields (Example: \( \text{speed} = \text{magnitude}(\text{velocity}) \))

**Operator 1**
- Apply an operator to transform the data (Example: Slice operator)

**Operator 2**
- Apply a second operator to transform the data (Example: Elevate operator)

**Plot**
- Plot a field (Example: Pseudocolor plot of \( \text{speed} \))

**Query**
- Extract quantitative information (Example: Maximum speed over cross-section)
Resources

Presenter Contact Info:
- Cyrus Harrison: cyrush@llnl.gov

User Resources:
- Main website: http://www.llnl.gov/visit
- Wiki: http://www.visitusers.org
- Email: visitusers@ornl.gov

Developer Resources:
- Email: visit-developers@ornl.gov
- Github: https://github.com/visit-dav/visit
Aneurysm Simulation Exploration

Remote Usage Tips

Python Scripting Basics

Connected Components

Additional Hands-on Materials

- **Potential Flow Simulation Exploration**

- **Water Flow Simulation Exploration**

- **Volume Rendering**

- **Movie Making**

- **Advanced Movie Making**
Resources

Presenter Contact Info:
- Cyrus Harrison: cyrush@llnl.gov

User Resources:
- Main website: http://www.llnl.gov/visit
- Wiki: http://www.visitusers.org
- Email: visitusers@ornl.gov

Developer Resources:
- Email: visit-developers@ornl.gov
- Github: https://github.com/visit-dav/visit
Visualization Techniques for Mesh-based Simulations
Pseudocolor rendering maps scalar fields to a range of colors

Pseudocolor rendering of Elevation

Pseudocolor rendering of Density
Volume Rendering cast rays though data and applies transfer functions to produce an image.
Isosurfacing (Contouring) extracts surfaces of that represent level sets of field values
Particle advection is the foundation of several flow visualization techniques

- $S(t) = \text{position of particle at time } t$
- $S(t_0) = p_0$
  - $t_0$: initial time
  - $p_0$: initial position
- $S'(t) = v(t, S(t))$
  - $v(t, p)$: velocity at time $t$ and position $p$
  - $S'(t)$: derivative of the integral curve at time $t$

This is an ordinary differential equation.
Streamline and Pathline computation are built on particle advection

- **Streamlines** – Instantaneous paths
- **Pathlines** – Time dependent paths
Meshes discretize continuous space

- Simulations use a wide range of mesh types, defined in terms of:
  - A set of coordinates ("nodes" / "points" / "vertices")
  - A collection of "zones" / "cells" / "elements" on the coordinate set

VisIt uses the “Zone” and “Node” nomenclature throughout its interface.
Mesh fields are variables associated with the mesh that hold simulation state

- Field values are associated with the zones or nodes of a mesh
  - Nodal: Linearly interpolated between the nodes of a zone
  - Zonal: Piecewise Constant across a zone

- Field values for each zone or node can be scalar, or multi-valued (vectors, tensors, etc.)
Material volume fractions are used to capture sub-zonal interfaces

- Multi-material simulations use volume/area fractions to capture disjoint spatial regions at a sub-grid level.

- These fractions can be used as input to high-quality sub-grid material interface reconstruction algorithms.
Species are used to capture sub-zonal weightings

- Species describe sub-grid variable composition
  - Example: Material “Air” is made of species “N2”, “O2”, “Ar”, “CO2”, etc.

- Species are used for weighting, not to indicate sub-zonal interfaces.
  - They are typically used to capture fractions of “atomically mixed” values.
Simulation meshes may be composed of smaller mesh “blocks” or “domains”.

Domains are partitioned across MPI tasks for processing.

Domain decomposed meshes enable scalable parallel visualization and analysis algorithms
Adaptive Mesh Refinement (AMR) refines meshes into patches that capture details across length scales

- Mesh domains are associated with patches and levels
- Patches are nested to form an AMR hierarchy