Visualization and Analysis of HPC Simulation Data with VisIt

ATPEC 2021
Monday August 9th, 2021

Cyrus Harrison (cyrush@llnl.gov)
Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.
Outline

- VisIt Project Introduction (~30 min)

- Hands-on: (~45 min)
  - Guided tour of VisIt
  - Visualization of an Aneurysm (Blood Flow) Simulation
Tutorial Resources

- **VisIt 3.2.1 or 3.2.0**
  - [https://github.com/visit-dav/visit/releases](https://github.com/visit-dav/visit/releases)

- **Tutorial Materials**

- **How to get in touch**
  - GitHub: [https://github.com/visit-dav/visit](https://github.com/visit-dav/visit)
  - GitHub Discussions: [https://github.com/visit-dav/visit/discussions](https://github.com/visit-dav/visit/discussions)
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
https://visit.llnl.gov/ ▼

VisIt is an Open Source, interactive, scalable, visualization, animation and analysis tool. From Unix, Windows or Mac workstations, users can interactively ...

VisIt Gallery · Key Features of VisIt · VisIt Home · Archive of VisIt Releases

You've visited this page many times.
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
- Quantitative Analysis
- Visual Debugging
- Comparative Analysis
- Presentation Graphics
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.

Project Started | LLNL ASC users transitioned to VisIt | 2005 R&D 100 | DOE SciDAC: VACET Funded | Transition to Public NERSC SW repo | Visit 2.0 Release | R&D Collaborations | ECP ALPINE Funded | Visit 3.0 Release
--- | --- | --- | --- | --- | --- | --- | --- | ---
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://github.com/visit-dav/visit/releases/)
  - 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://github.com/visit-dav/visit/releases/)
  - Fledgling support for building via spack (https://github.com/spack/spack)
VisIt supports more than 110 file formats

“How do I get my data into VisIt?”

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

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

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

- Experiment with the *visit_writer* utility:

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.](image)

![Algorithms research: How to efficiently calculate particle paths in parallel.](image)

![Algorithms research: Reconstructing material interfaces for visualization](image)

![Methods research: How to incorporate statistics into visualization.](image)
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 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

**GitHub**

VisIt 3.2 (April 2021) includes Python 3 support

- We ported our legacy docs to Sphinx, now hosted on Read the Docs

**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)
VisIt employs a parallelized client-server architecture.
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
  - ALPINE (ParaView/VisIt)
    - http://alpine.dsscale.org
  - ParaView Catalyst
    - http://www.paraview.org/in-situ
  - VisIt LibSim
    - https://visit.llnl.gov
  - SENSEI in situ
    - http://www.sensei-insitu.org
  - Ascent
    - https://github.com/Alpine-DAV/ascent

VTK-m
http://m.vtk.org

DIY
https://github.com/diatomic/diy
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)

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

### Diagram

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

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

3. **Plot**
   - Make a plot of the result
     (Example: Pseudocolor plot of $speed$)
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

Resources:
- Main website: http://www.llnl.gov/visit
- Github: https://github.com/visit-dav/visit
- GitHub Discussions: https://github.com/visit-dav/visit/discussions
- Wiki: http://www.visitusers.org
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

Resources:

- Main website: http://www.llnl.gov/visit
- Github: https://github.com/visit-dav/visit
- GitHub Discussions: https://github.com/visit-dav/visit/discussions
- Wiki: http://www.visitusers.org
Additional Slides
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.
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 a AMR hierarchy
Resources

Presenter Contact Info:

- Cyrus Harrison: cyrush@llnl.gov

Resources:

- Main website: http://www.llnl.gov/visit
- Github: https://github.com/visit-dav/visit
- GitHub Discussions: https://github.com/visit-dav/visit/discussions
- Wiki: http://www.visitusers.org
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.