Ascent: Flyweight In Situ Visualization and Analysis for HPC Simulations

ECP Web Tutorial

Matt Larsen (LLNL), Cyrus Harrison (LLNL), Hank Childs (Univ of Oregon)

December 17th, 2020

LLNL-PRES-817787

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

Lawrence Livermore National Laboratory

Acknowledgements

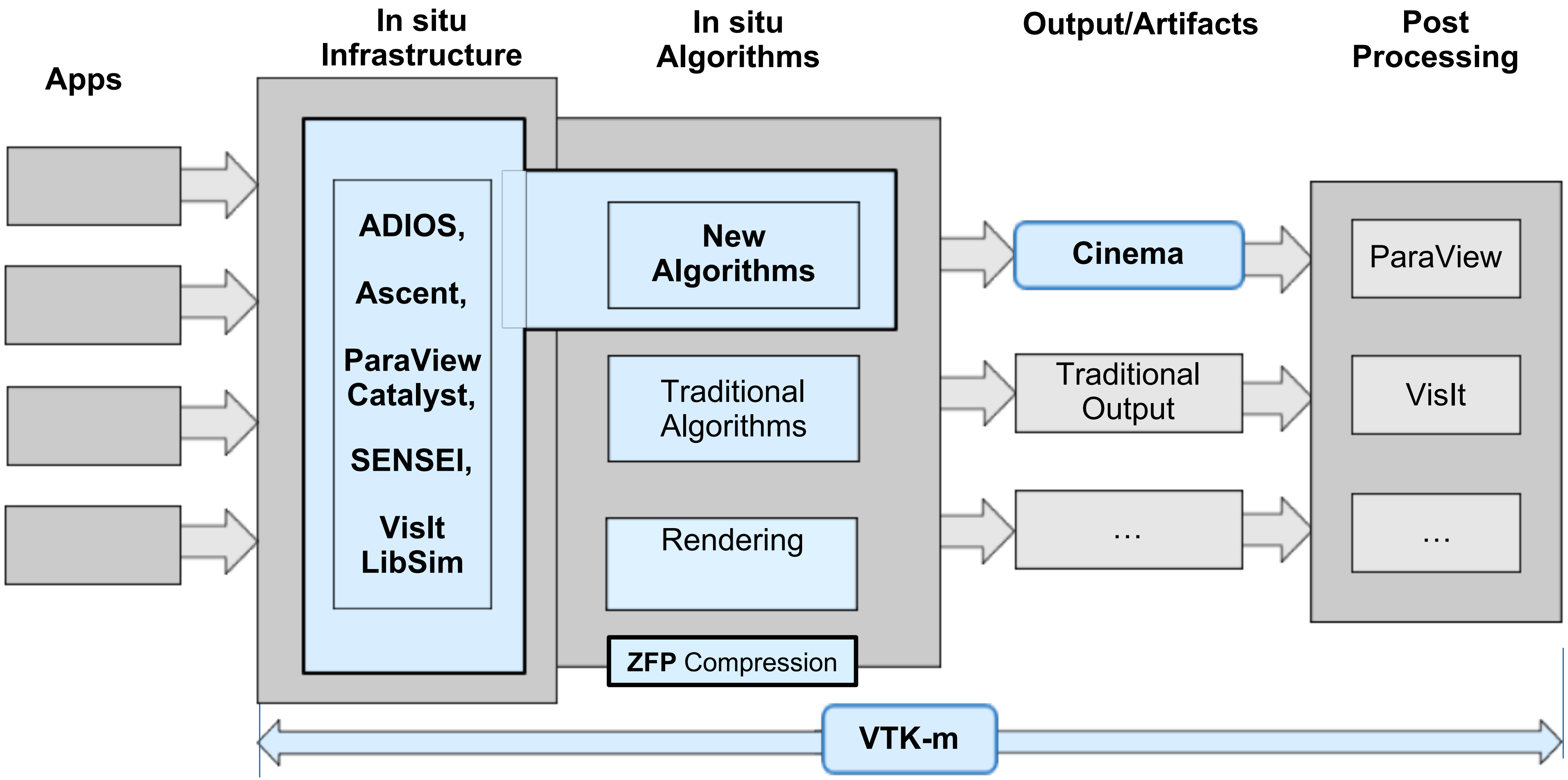
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

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.

What is In situ processing?

- Defined:
  - Process data while it is generated
  - Couple visualization and analysis routines with the simulation code (avoiding file system I/O)
- Pros:
  - No or greatly reduced I/O vs post-hoc processing
  - Can access all the data
  - Computational power readily available
- Cons:
  - Must know what you want to look for a priori
  - Increasing complexity
  - Constraints (memory, network)

Ascent is a part of a broader coordinated visualization and data analysis ecosystem



In situ processing works in various ways

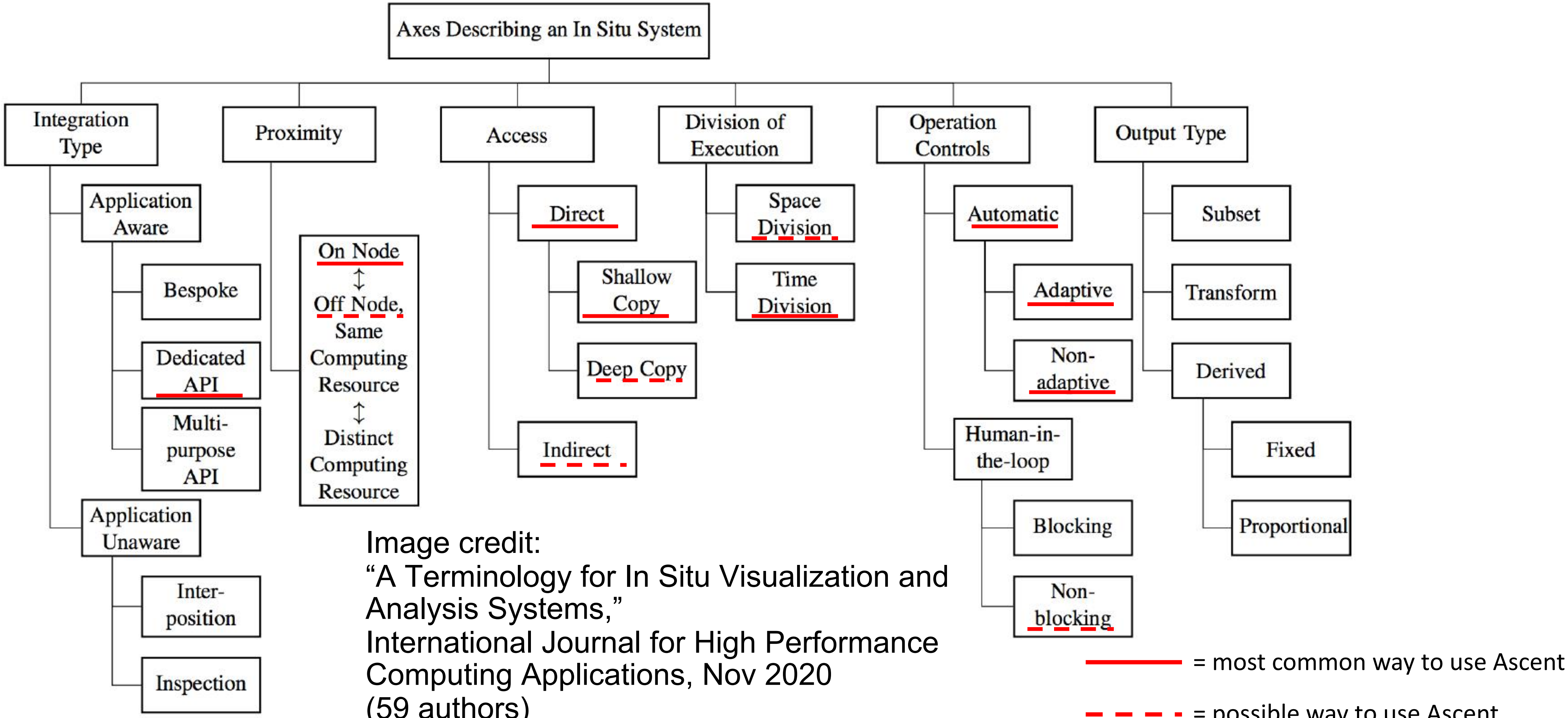
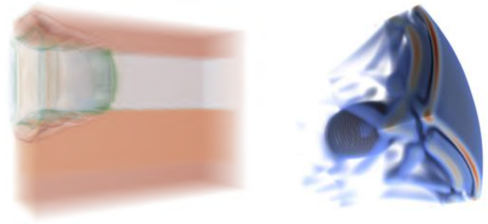


Image credit:
"A Terminology for In Situ Visualization and Analysis Systems,"
International Journal for High Performance Computing Applications, Nov 2020
(59 authors)

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/

Ascent



Visualizations created using Ascent

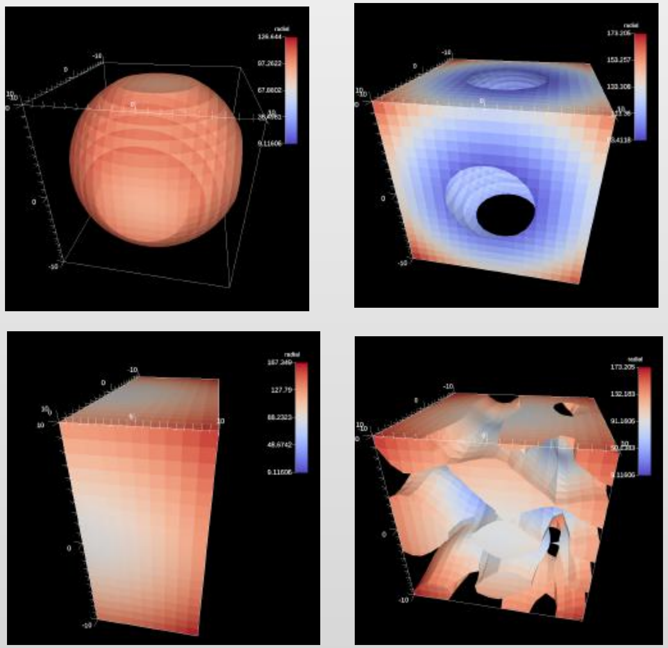
python HDF CINEMA

Extracts supported by Ascent

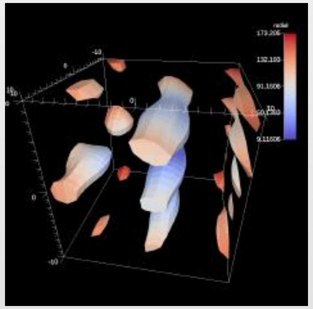
http://ascent-dav.org
https://github.com/Alpine-DAV/ascent

Website and GitHub Repo

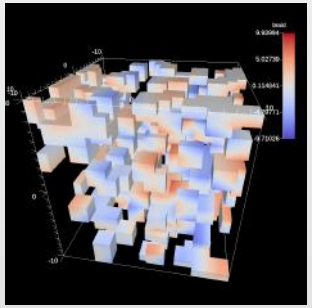
Ascent is ready for common visualization use cases



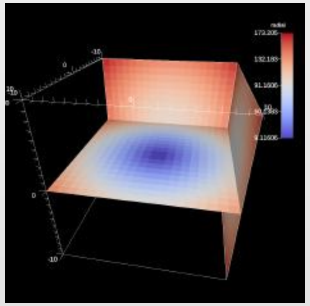
Clips



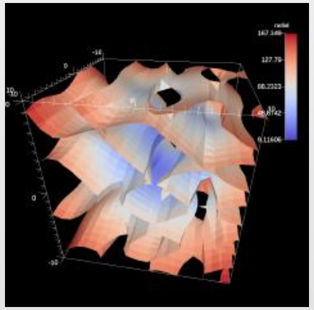
Iso-Volume



Threshold

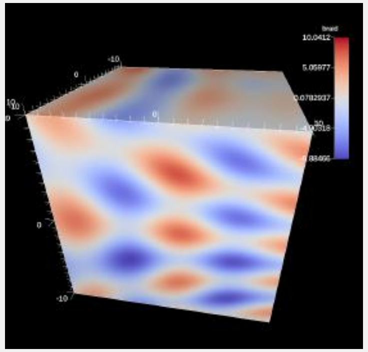


Slice

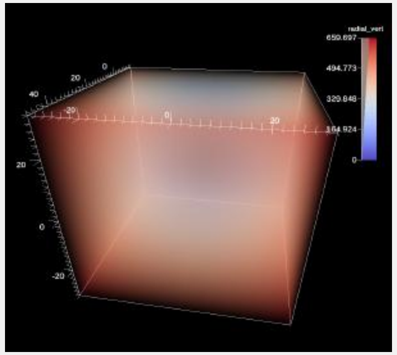


Contour

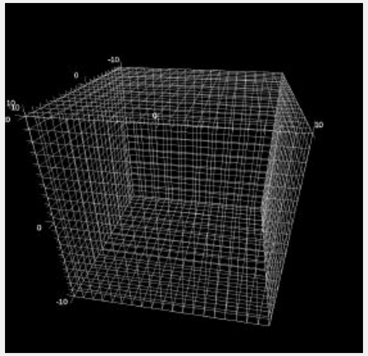
Rendering



Pseudocolor



Volume



Mesh

Ascent development is supported by the ECP ALPINE S&T project and LLNL's WSC program

ECP ALPINE (2.3.4.12)

Scope & Intent	R&D Themes	Delivery Process	Target ECP Users	Support Model
Deliver in situ visualization and analysis algorithms and infrastructure.	1) Automated in situ massive data reduction **algorithms** 2) Portable, scalable, performant **infrastructure**	Regular releases of software and documentation, open access to production software from GitHub	All ECP applications. Focused delivery for co-design centers applications.	Ongoing developer support. Dedicated email, issue tracking portals, comprehensive web-based documentation, regular tutorials.

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.

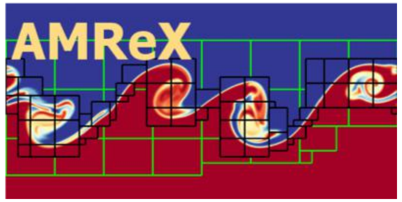
Ascent is one of the infrastructure thrusts for ECP ALPINE and a key part of LLNL WSC's in situ strategy

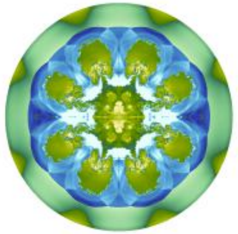
We are working to integrate and deploy Ascent with HPC simulation codes (ECP and beyond)

INT-755 INT-323

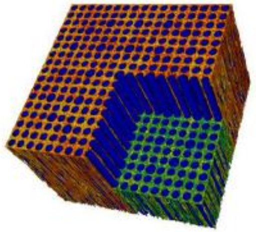
CEED
EXASCALE DISCRETIZATIONS

INT-311

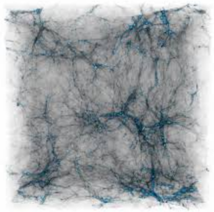




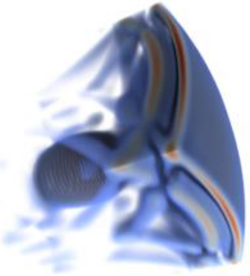
MARBL
INT-595



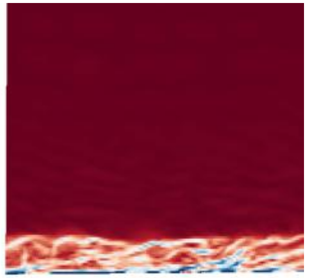
NekRS
INT-398



Nyx
INT-179

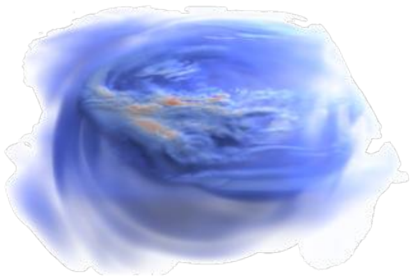


WarpX
INT-825



AMRWind
INT-1350

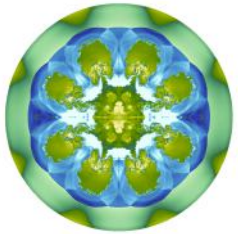
Pele
INT-133

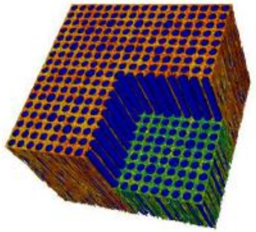


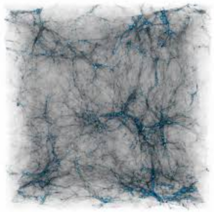
SW4
INT-190

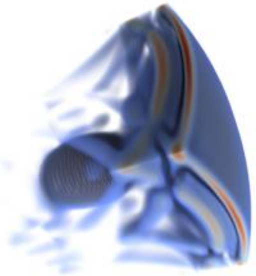
Ascent

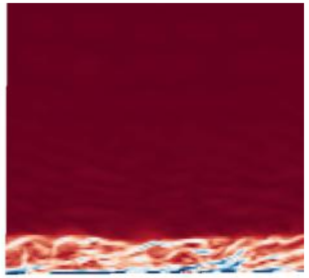
Ascent connects applications to visualization and analysis capabilities


MARBL

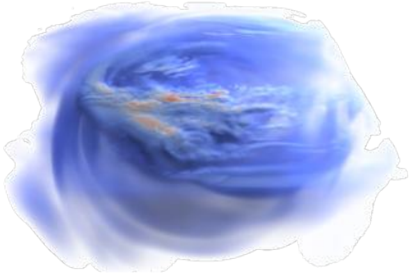

NekRS


Nyx


WarpX


AMRWind

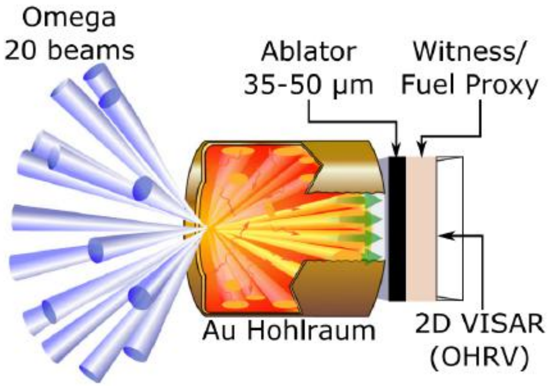

Pele


SW4

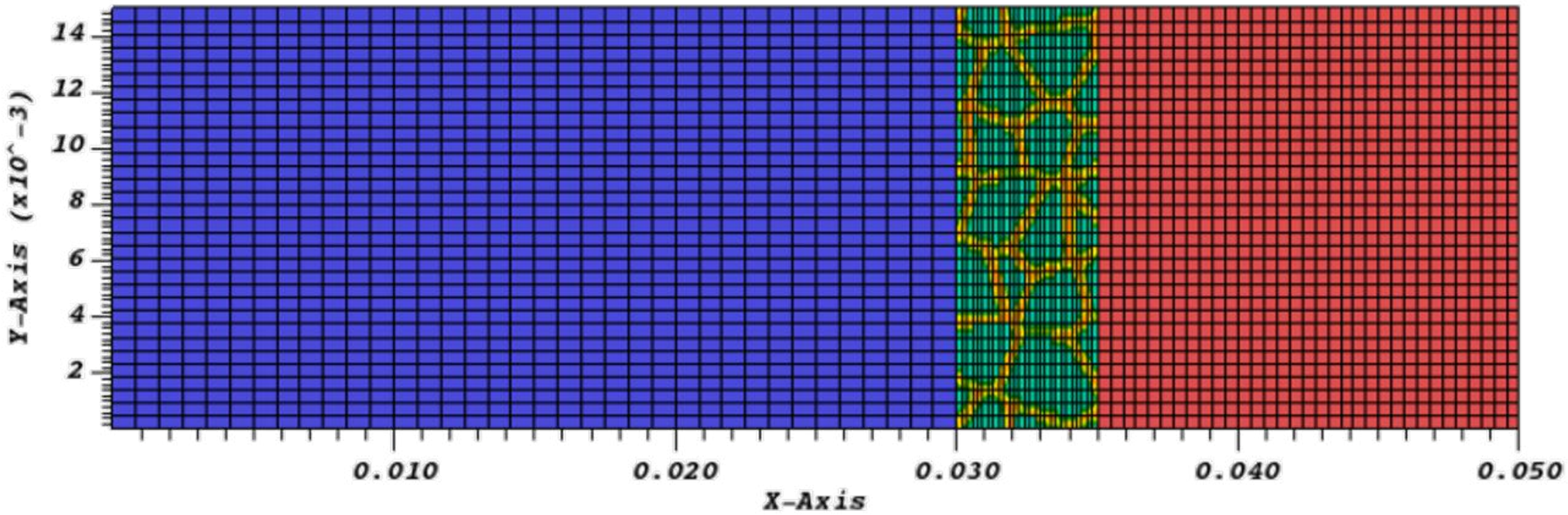
Ascent

- ADIOS
- Lagrangian Flow
- ParaView
- Contour Trees
- HDF5
- Derived Quantities
- VTK-m
- Anomaly Detection
- Sampling
- Optimal Viewpoint
- Cinema
- Jupyter Notebooks
- Data Binning

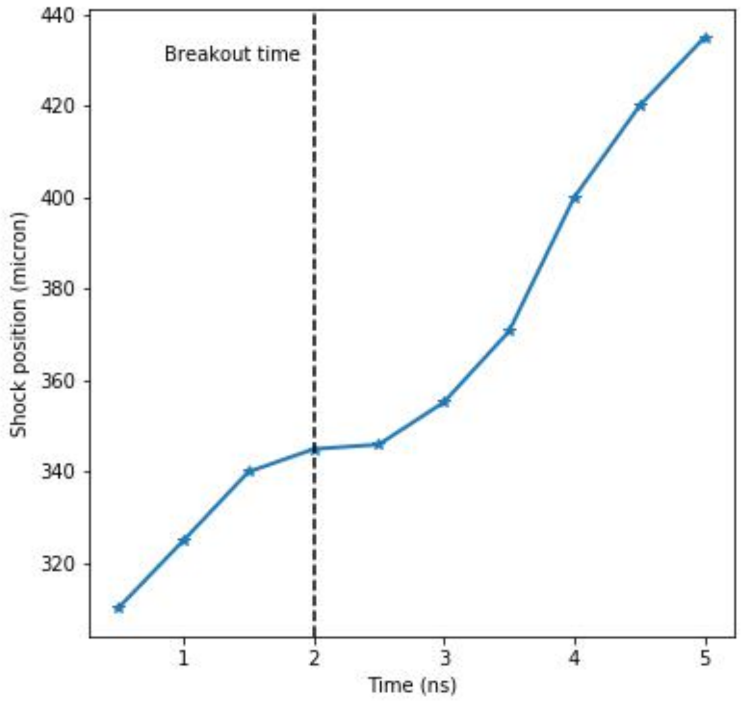
Science Enabling Results: Shock Front Tracking (VISAR)



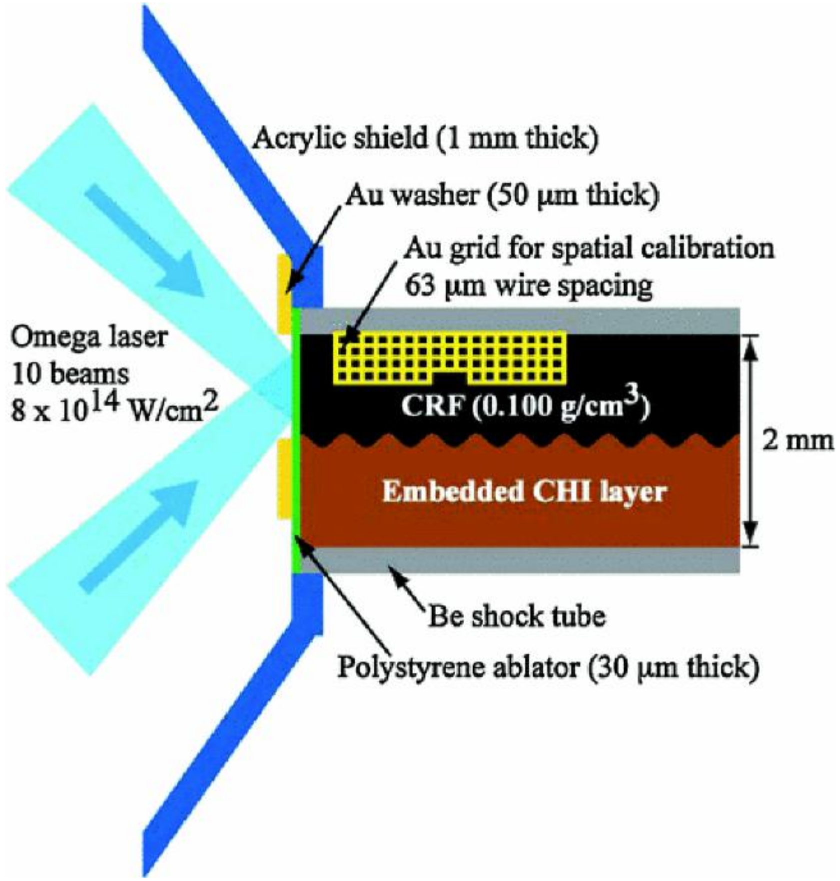
Velocity interferometer system for any reflector (VISAR)



Shock position tracked in Ascent

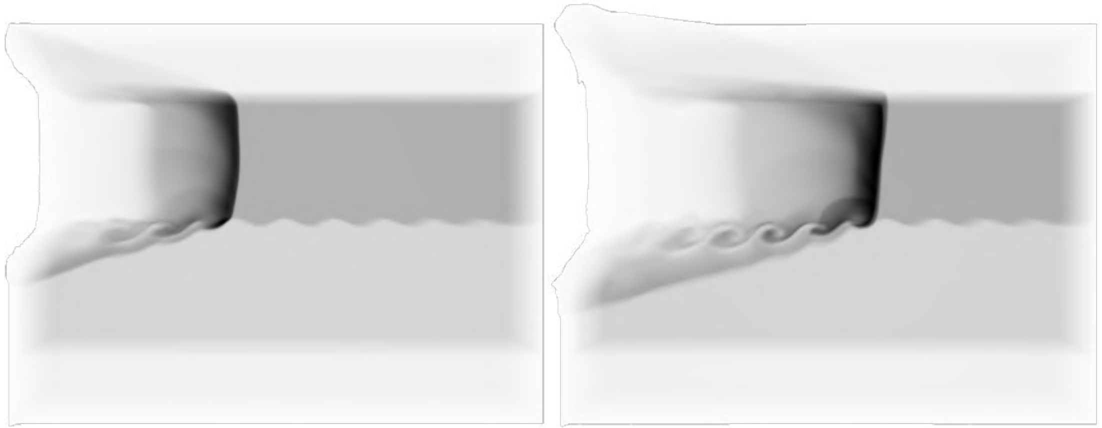


Science Enabling Results: Simulation Validation

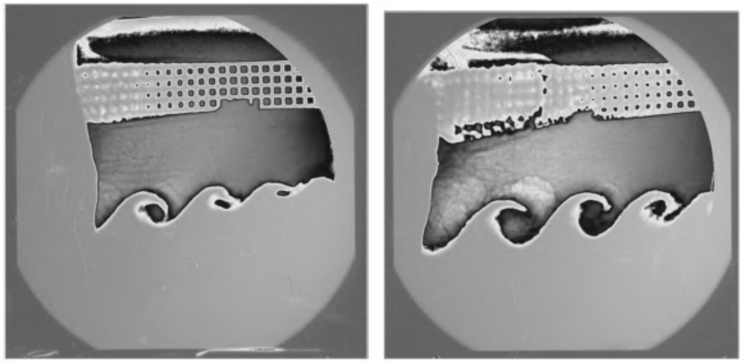


Radiographs

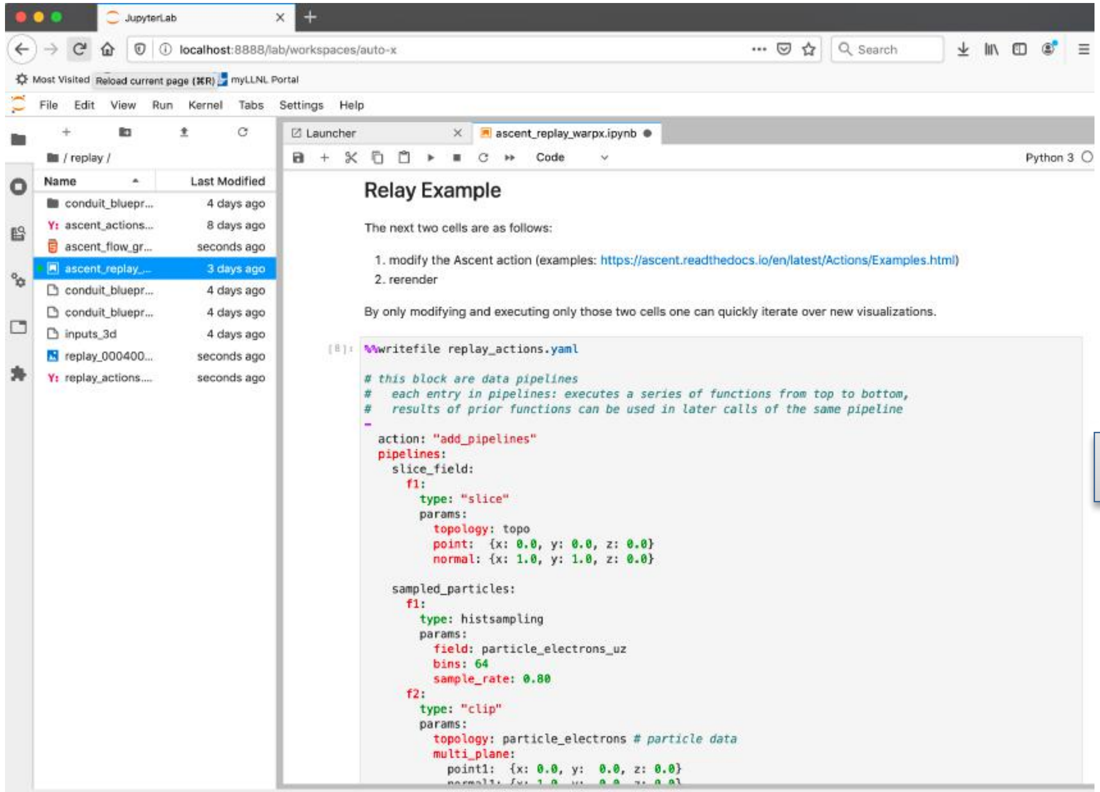
Simulated



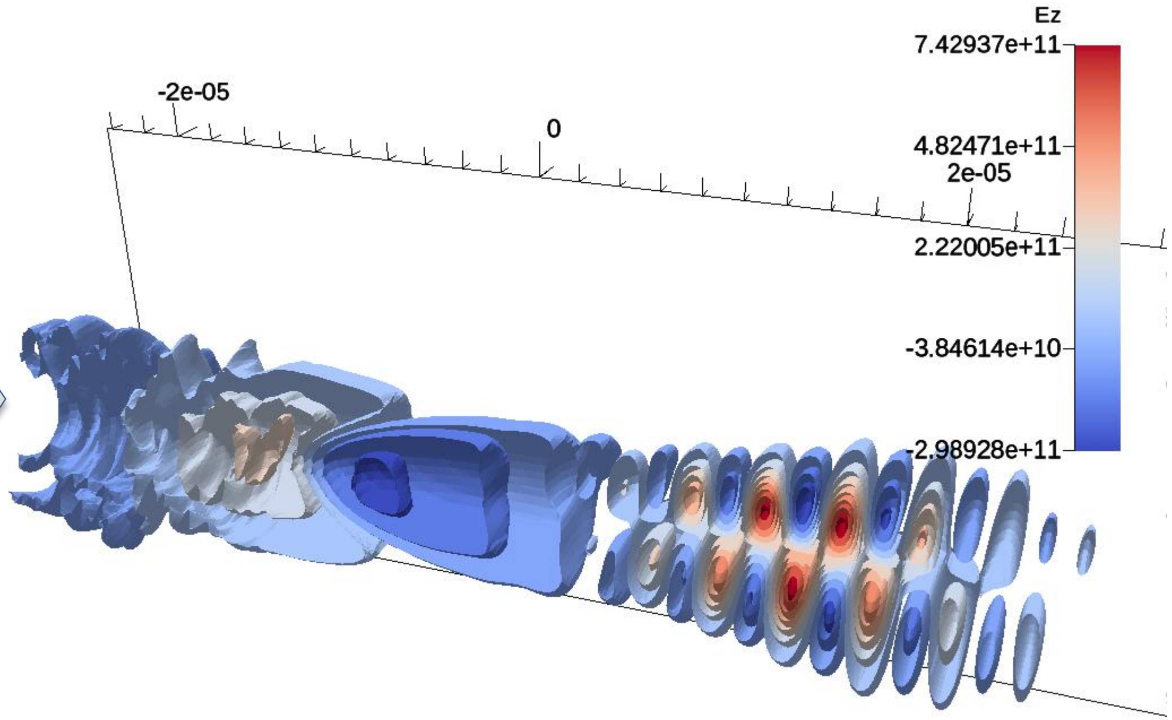
Experimental



Science Enabling Results: WarpX Workflow Tools (Jupyter Labs)



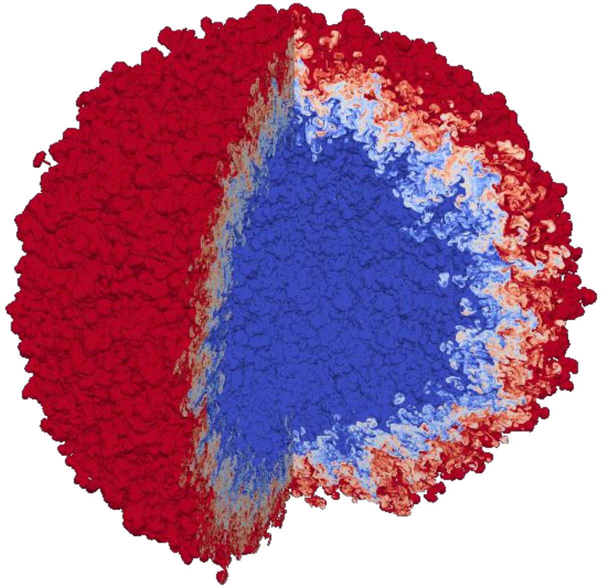
Jupyter Labs Interface



Resulting Image

Science Enabling Results: Rendering At Scale

- The ***97.8 billion*** element simulation ran across ***16,384 GPUs*** on ***4,096 Nodes***
- Time-varying evolution of the mixing was visualized in-situ using **Ascent**, also leveraging 16,384 GPUs
- Ascent leveraged **VTK-m** to run visualization algorithms on the GPUs



Visualization of an idealized Inertial Confinement Fusion (ICF) simulation of Rayleigh-Taylor instability with two fluids mixing in a spherical geometry.

Today we will teach you about Ascent's API and capabilities

You will learn:

- How to use Conduit, the foundation of Ascent's API
- How to get your simulation data into Ascent
- How to tell Ascent what pictures to render and what analysis to execute

Ascent tutorial examples are outlined in our documentation and included ready to run in Ascent installs

Ascent

latest

Search docs

- Quick Start
- Ascent User Documentation
- Developer Documentation
- ⊟ Tutorial
  - Tutorial Setup
  - Introduction to Ascent
  - CloverLeaf3D Ascent Demos
- Releases
- Publications and Presentations

Docs » Tutorial

Edit on GitHub

Tutorial

This tutorial introduces how to use Ascent, including basics about:

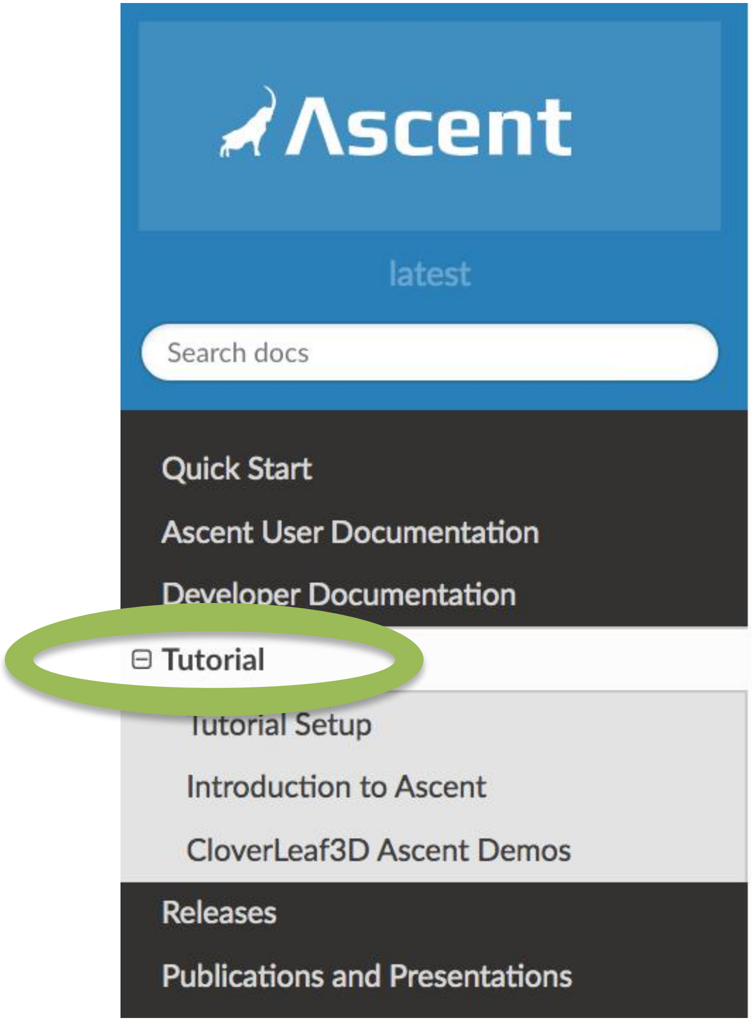
- Formating mesh data for Ascent
- Using Conduit and Ascent's Conduit-based API
- Using and combining Ascent's core building blocks: Scenes, Pipelines, Extracts, Queries, and Triggers
- Using Ascent with the Cloverleaf3D example integration

Ascent installs include standalone C++, Python, and Python-based Jupyter notebook examples for this tutorial. You can find the tutorial source code and notebooks in your Ascent install directory under `examples/ascent/tutorial/ascent_intro/` and the Cloverleaf3D demo files under `examples/ascent/tutorial/cloverleaf_demos/`.

http://ascent-dav.org

Ascent tutorial examples are outlined in our documentation and included ready to run in Ascent installs

- http://ascent-dav.org
- Click on “Tutorial”



Ascent's interface provides five top-level functions

- **open() / close()**
  — Initialize and finalize an Ascent instance
- **publish()**
  — Pass your simulation data to Ascent
- **execute()**
  — Tell Ascent what to do
- **info()**
  — Ask for details about Ascent's last operation

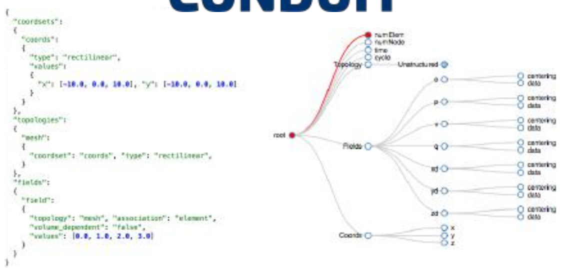
```
//
// Run Ascent
//

Ascent ascent;
ascent.open();

ascent.publish(data);
ascent.execute(actions);
ascent.info(details);

ascent.close();
```

The *publish(), execute()* and *info()* methods take a Conduit tree as an argument.
What is a Conduit tree?

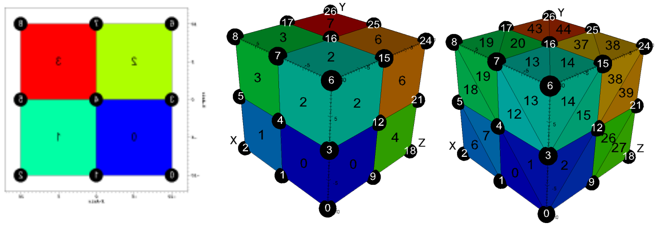
Conduit provides intuitive APIs for in-memory data description and exchange

- **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)

CONDUIT



Hierarchical in-memory data description



Conventions for sharing in-memory mesh data

http://software.llnl.gov/conduit

http://github.com/llnl/conduit

Website and GitHub Repo

Ascent uses Conduit to provide a flexible and extendable API

- Conduit underpins Ascent's support for C++, C, Python, and Fortran interfaces
- Conduit also enables using YAML to specify Ascent actions
- Conduit's zero-copy features help couple existing simulation data structures
- Conduit Blueprint provides a standard for how to present simulation meshes

Learning Ascent equates to learning how to construct and pass Conduit trees that encode your data and your expectations.

To start, let’s look at the Ascent “First Light” Example in C++

- https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html

```cpp
#include <iostream>

#include "ascent.hpp"
#include "conduit_blueprint.hpp"

using namespace ascent;
using namespace conduit;

int main(int argc, char **argv)
{
    // echo info about how ascent was configured
    std::cout << ascent::about() << std::endl;

    // create conduit node with an example mesh using
    // conduit blueprint's braid function
    // ref: https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html#braid

    // things to explore:
    //  changing the mesh resolution

    Node mesh;
    conduit::blueprint::mesh::examples::braid("hexs",
                                              50,
                                              50,
                                              50,
                                              mesh);
```

This code generates an example mesh

To start, let’s look at the Ascent “First Light” Example in C++

- https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html

```cpp
// create an Ascent instance
Ascent a;

// open ascent
a.open();
```

Create an Ascent instance and set it up

```cpp
// publish mesh data to ascent
a.publish(mesh);
```

Now Ascent has access to our mesh data

To start, let's look at the Ascent "First Light" Example in C++

- https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html

```
//
// Ascent's interface accepts "actions"
// that to tell Ascent what to execute
//
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";

// Create an action that tells Ascent to:
// add a scene (s1) with one plot (p1)
// that will render a pseudocolor of
// the mesh field `braid`
Node & scenes = add_act["scenes"];

// things to explore:
//  changing plot type (mesh)
//  changing field name (for this dataset: radial)
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// set the output file name (ascent will add ".png")
scenes["s1/image_name"] = "out_first_light_render_3d";
```

Create a tree that describes the actions we want Ascent to do

```
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          field: "braid"
      image_name: "out_first_light_render_3d"
```

```
// view our full actions tree
std::cout << actions.to_yaml() << std::endl;
```

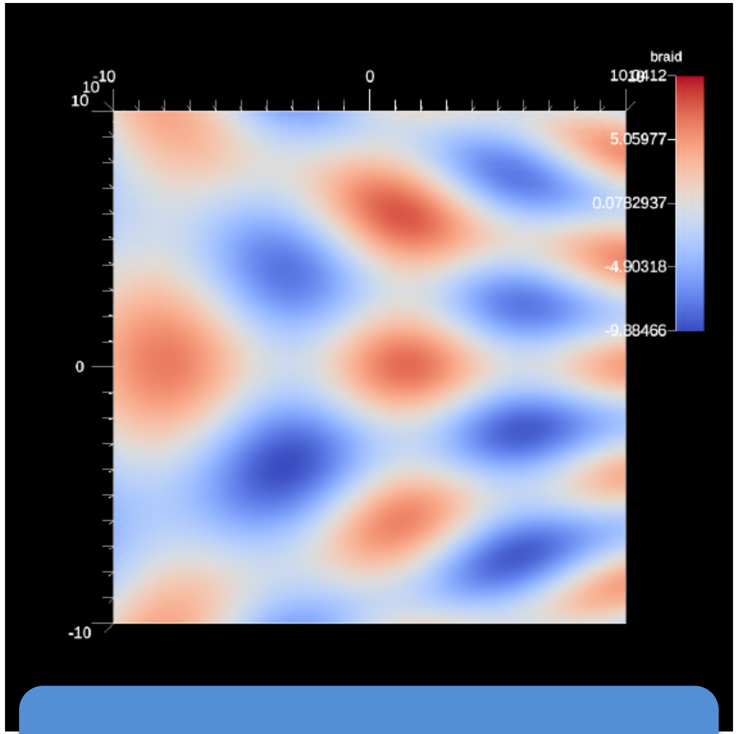
Equivalent YAML Description

To start, let's look at the Ascent "First Light" Example in C++

- https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html

```
// execute the actions
a.execute(actions);
```

Tell Ascent to execute these actions



Rendered Result!

Ascent's interface provides five composable building blocks

Scenes
(Render Pictures)

Pipelines
(Transform Data)

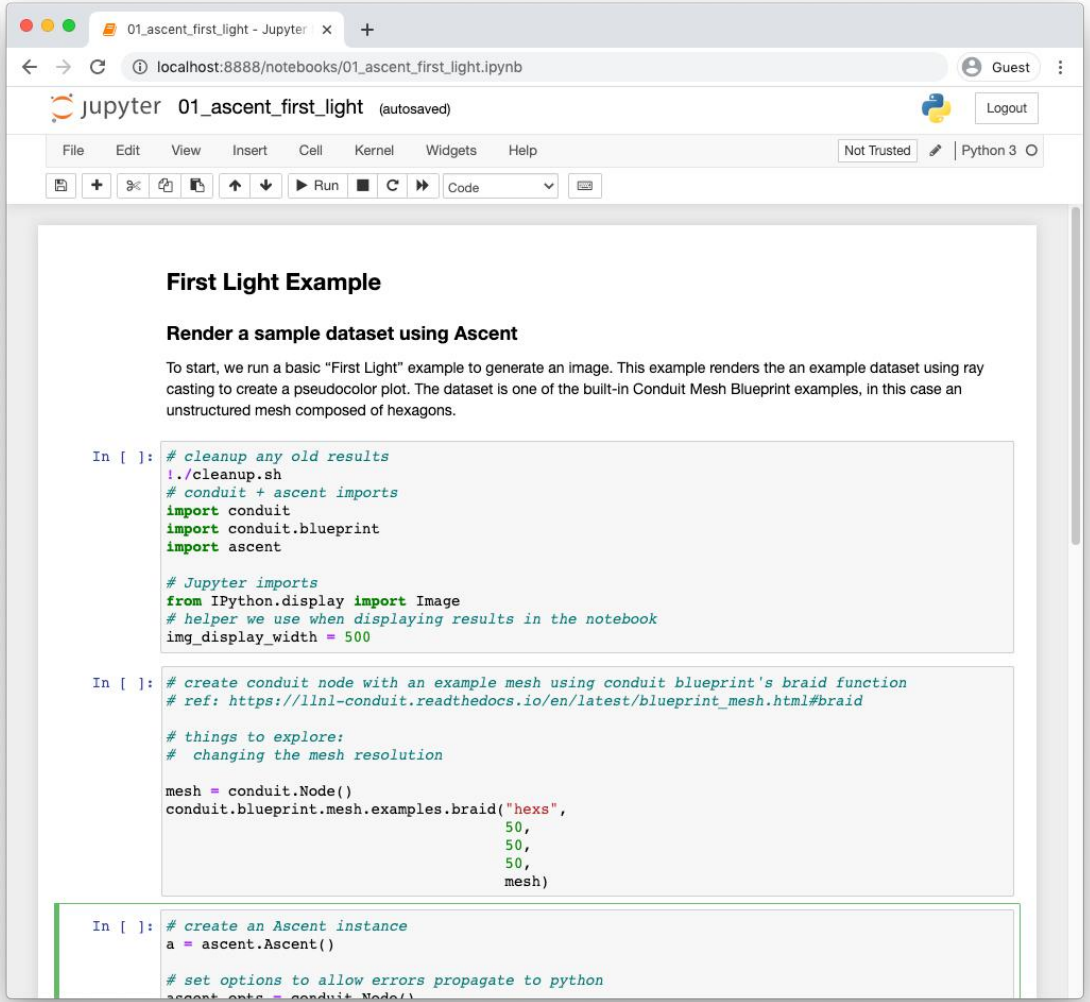
Extracts
(Capture Data)

Queries
(Ask Questions)

Triggers
(Adapt Actions)

The tutorial provides examples for all of these.

For the reminder of the tutorial, we will run the Ascent Tutorial examples using Jupyter Notebooks



You can follow along using cloud hosted Jupyter Lab servers

Start here:

https://www.ascent-dav.org/tutorial/

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.

Lawrence Livermore National Laboratory