Ascent: Flyweight In Situ Visualization and Analysis for HPC Simulations

ATPESC 2025
Data Analysis and Visualization Track

2025/08/04

Ascent Development Team

LLNL

Prepared by LLNL under Contract DE-AC52-07NA27344.

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

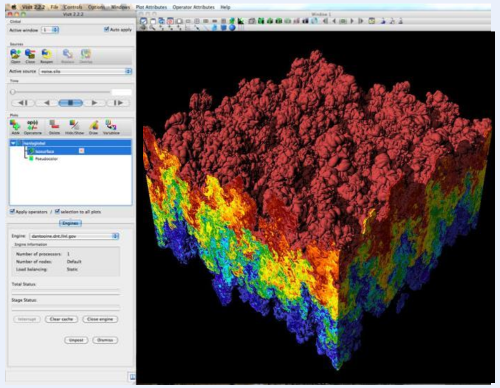
This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology,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

Outline

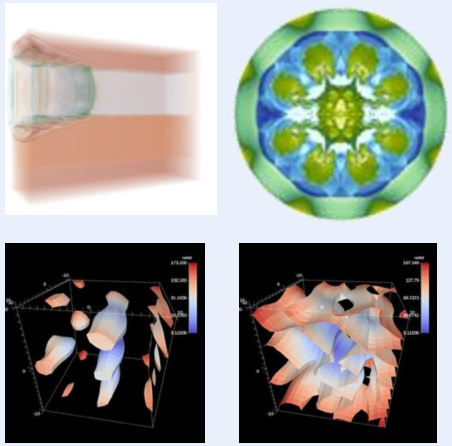
- Introduction:
  - In situ Visualization Concepts (10 min)
  - Ascent Project Overview (15 min)

- Hands-on:
  - Tutorial Exercises in cloud-hosted Jupyter Notebooks

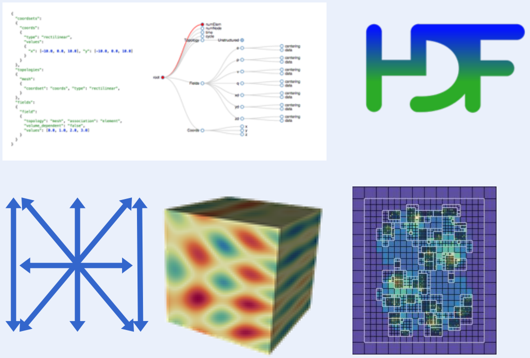
The VisIt team develops open-source Visualization, Analysis, and I/O tools



Turnkey HPC application for visualization and analysis of simulation data

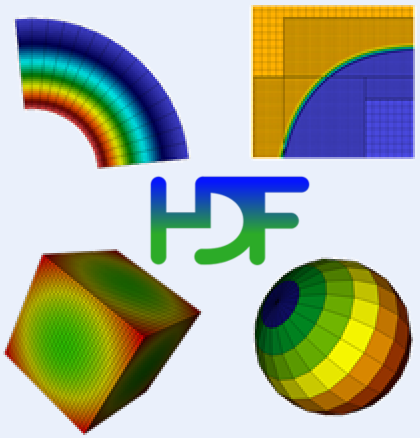


Easy-to-use flyweight in situ visualization and analysis library for HPC simulations



In-memory data description, HPC I/O, and shared schemas for simulation data exchange

Silo

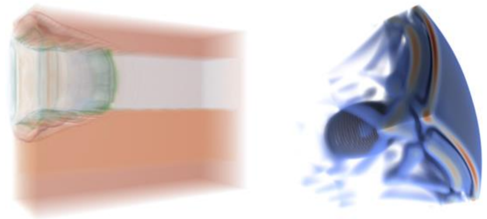


File-based, scientific data exchange library for checkpoint restart and visualization

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, HIP, 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 fewer dependencies than current tools (ex: no OpenGL)

Ascent



Visualizations created using Ascent

HDF python™ CINEMA

Extracts supported by Ascent

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

Website and GitHub Repo

Introduction to In Situ Processing Concepts

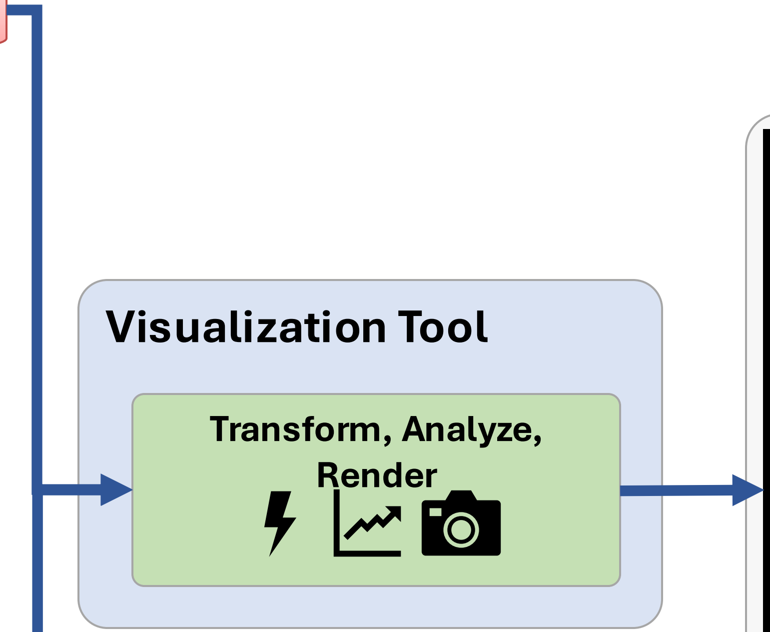
Scientific visualization tools transform, analyze, and render mesh-based data from HPC simulations

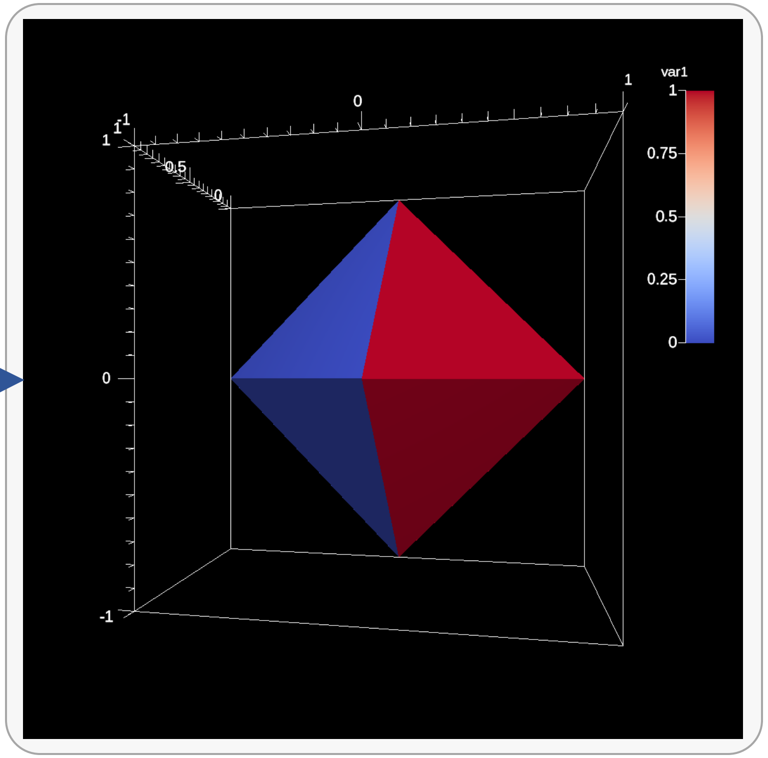
Mesh Data

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
    elements:
      shape: "tet"
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
  density:
    association: "element”
    topology: ”topo"
    values: [1.0, 2.0]
```

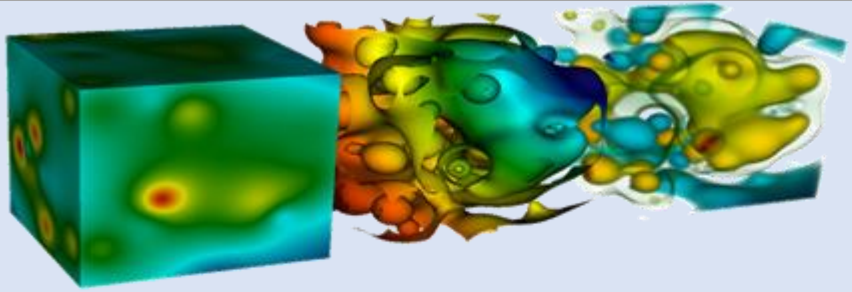
Actions

```
-
  action: "add_scenes"
  scenes:
    my_scene:
      plots:
        my_plot:
          type: "pseudocolor"
          field: "var1"
```

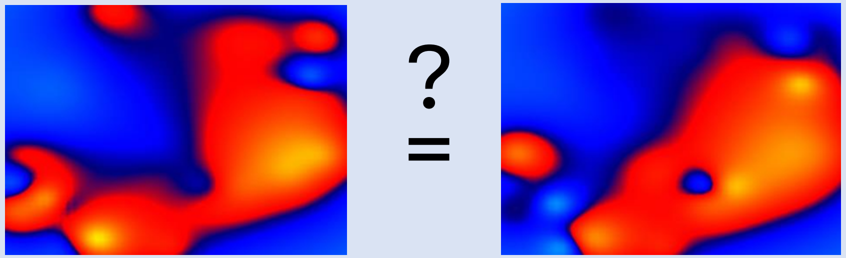




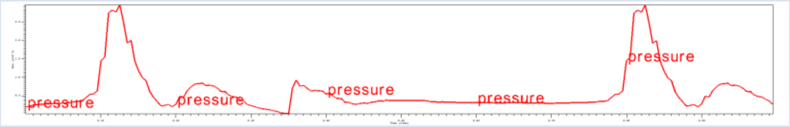
Scientific visualization tools support a wide range of use cases



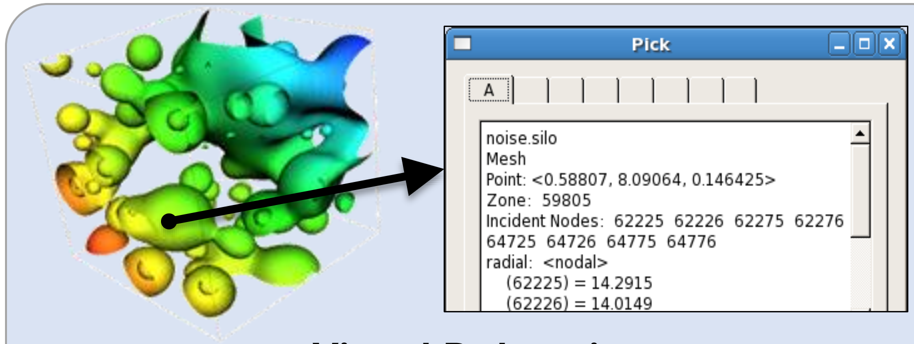
Data Exploration



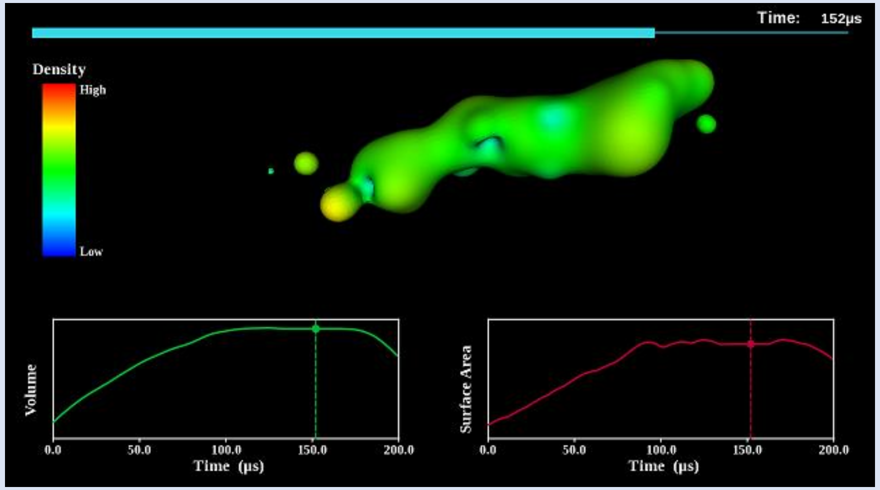
Comparative Analysis



Quantitative Analysis

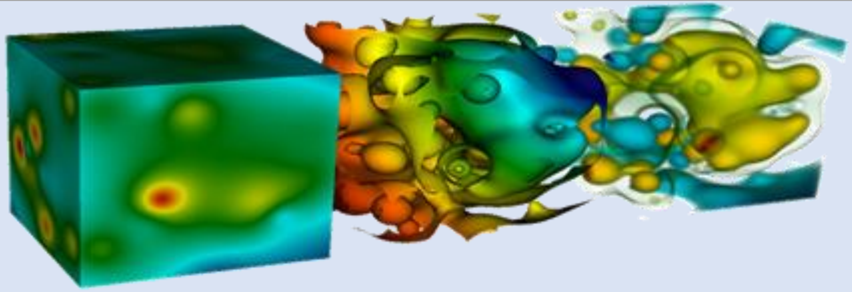


Visual Debugging

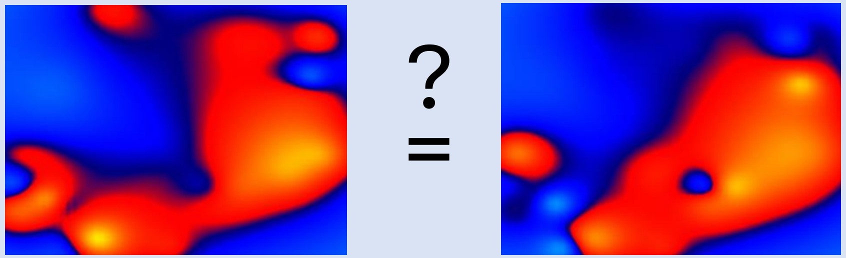


Presentation Graphics

Scientific visualization tools support a wide range of use cases

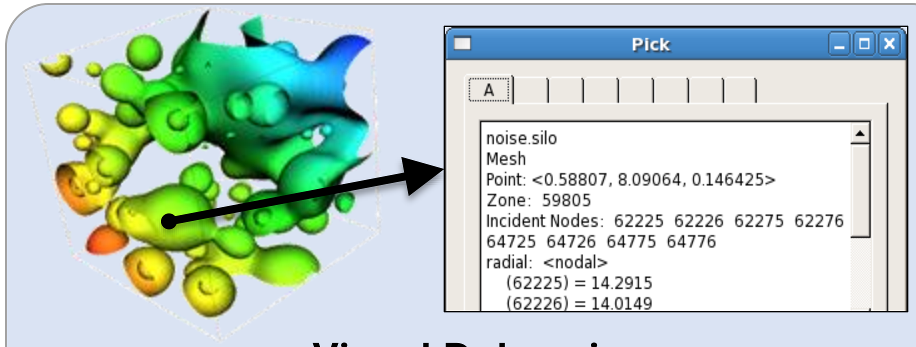


Data Exploration

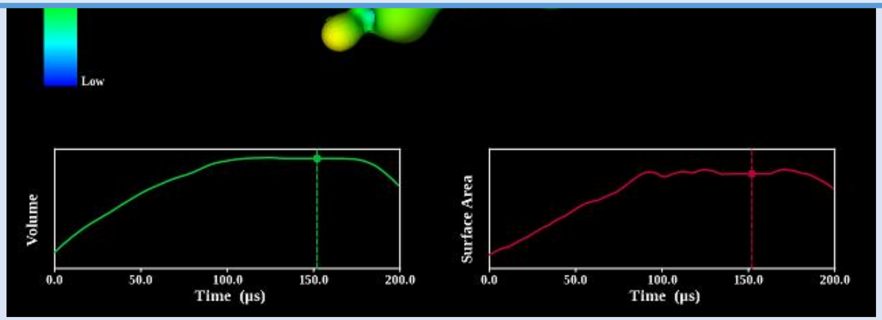


Comparative Analysis

These tools are used daily by scientists to digest and understand HPC simulation results



Visual Debugging



Presentation Graphics

Scientific visualization tools are used both post hoc and in situ

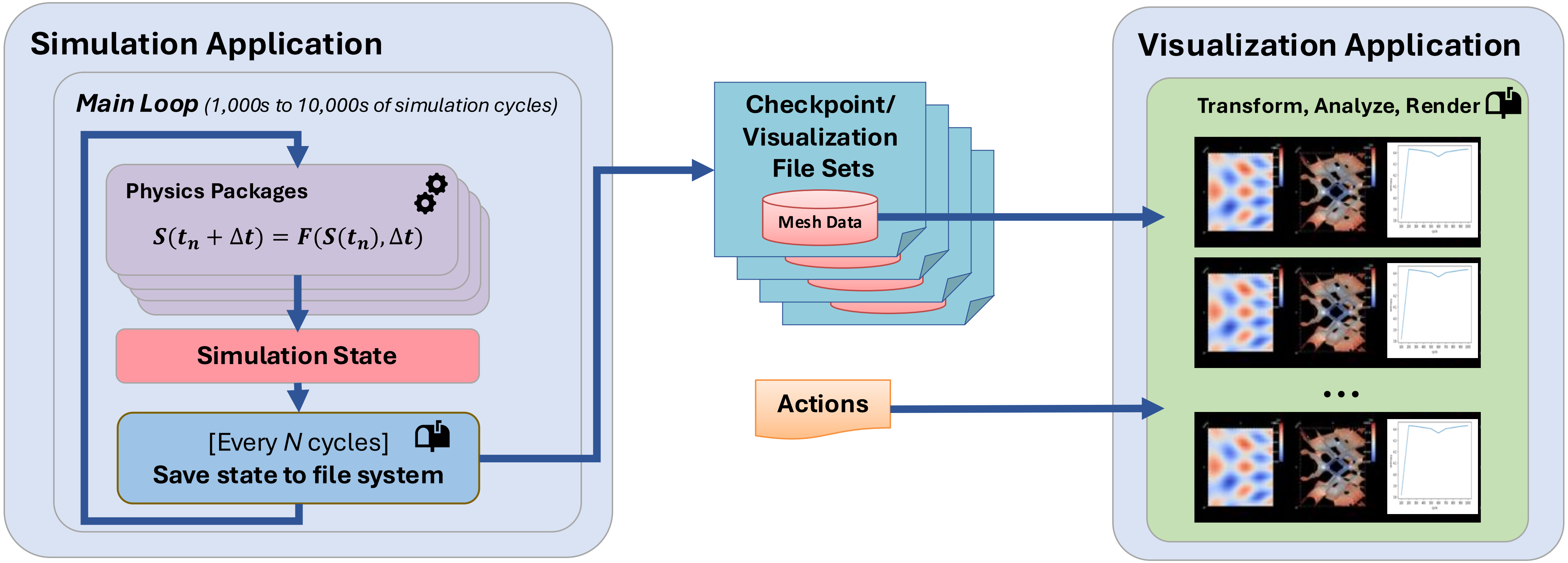
Post Hoc

Simulation data is processed after the simulation is run using distinct compute resources.

In Situ

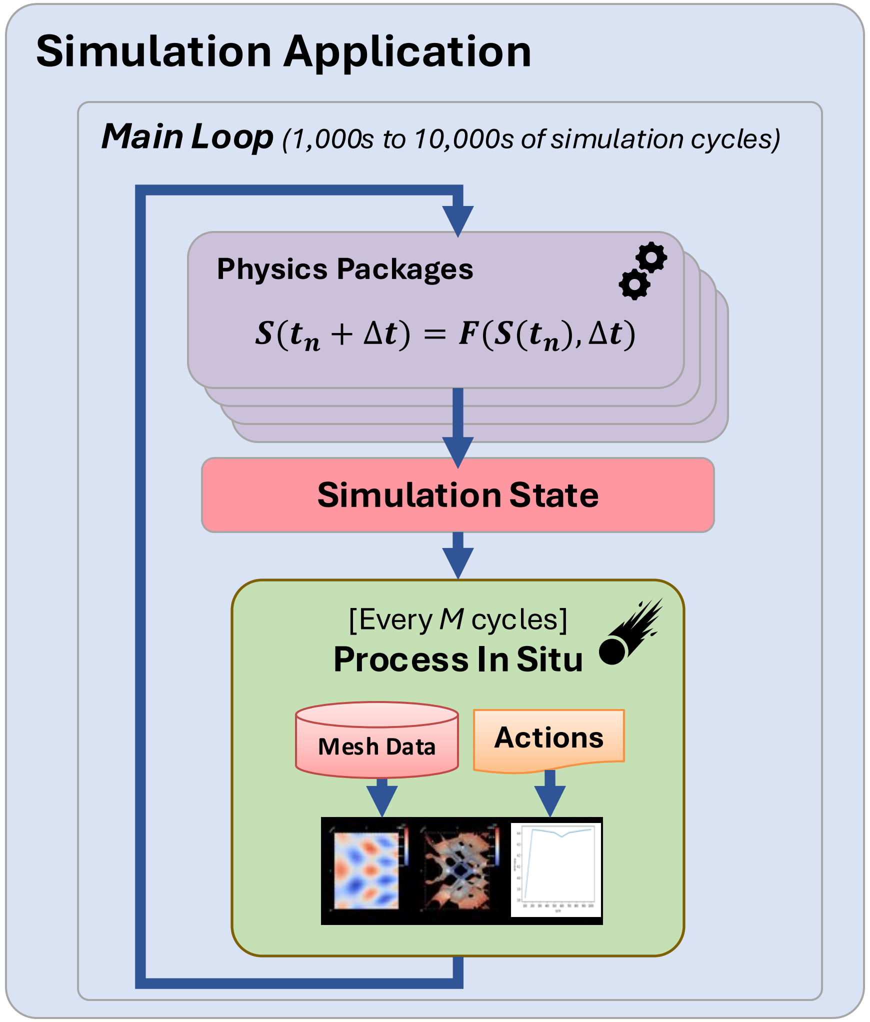
Simulation data is processed while it is generated, sharing compute resources with the simulation application.

Post Hoc visualization is the most widely used paradigm to process simulation results

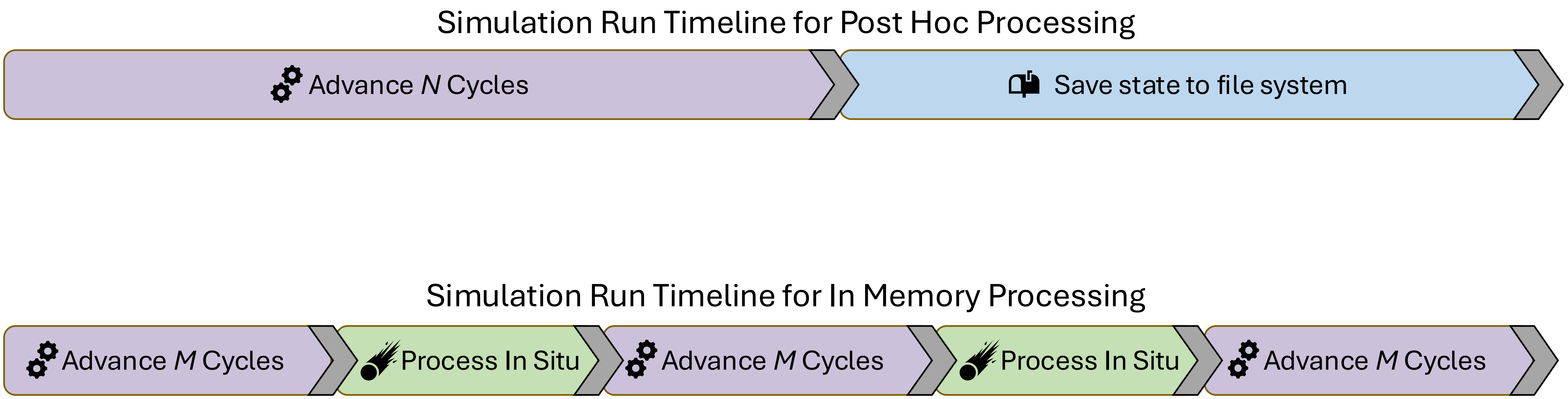


In situ processing expands the data we can access

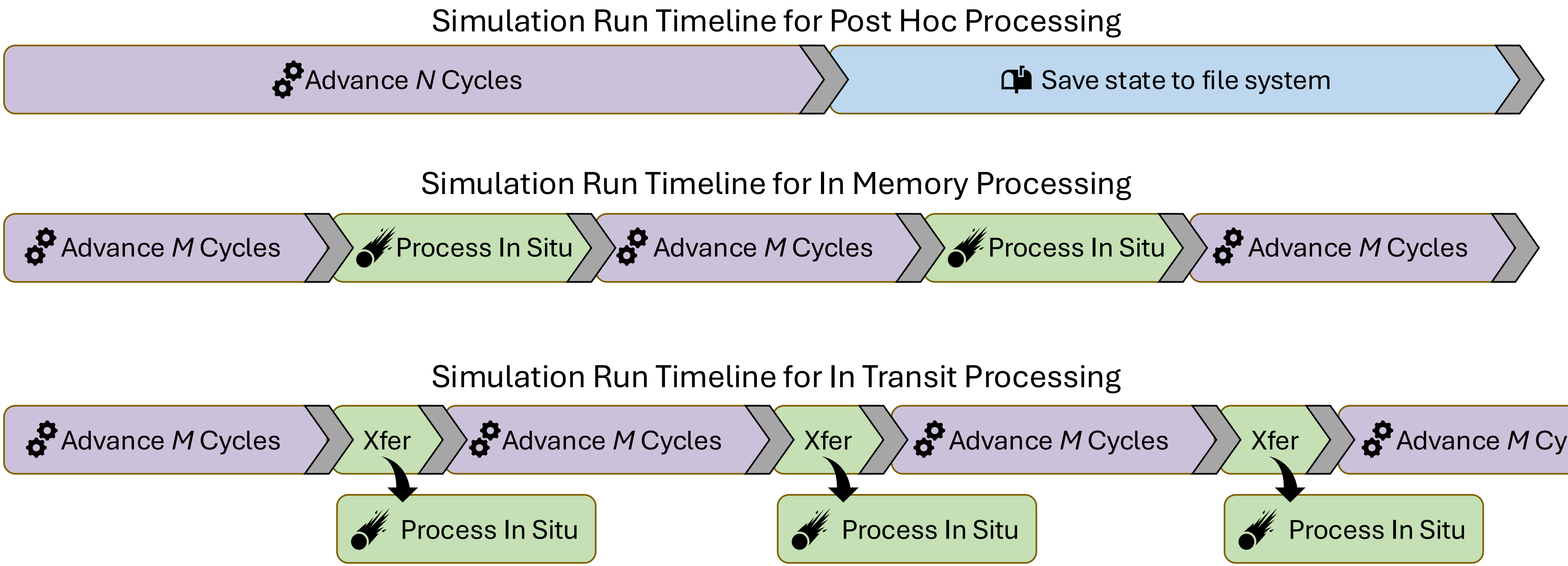
- In situ tools couple visualization and analysis routines with the simulation application (avoiding file system I/O)
- Pros:
  - No or greatly reduced I/O vs post hoc processing
  - Can access all simulation data
  - Computational power is readily available
  - Results are ready after simulation completes
- Cons:
  - More difficult when lacking a priori knowledge of what to visualize/analyze
  - Increasing complexity
  - Constraints (memory, network)



HPC Compute vs I/O speed ratios can favor in situ processing

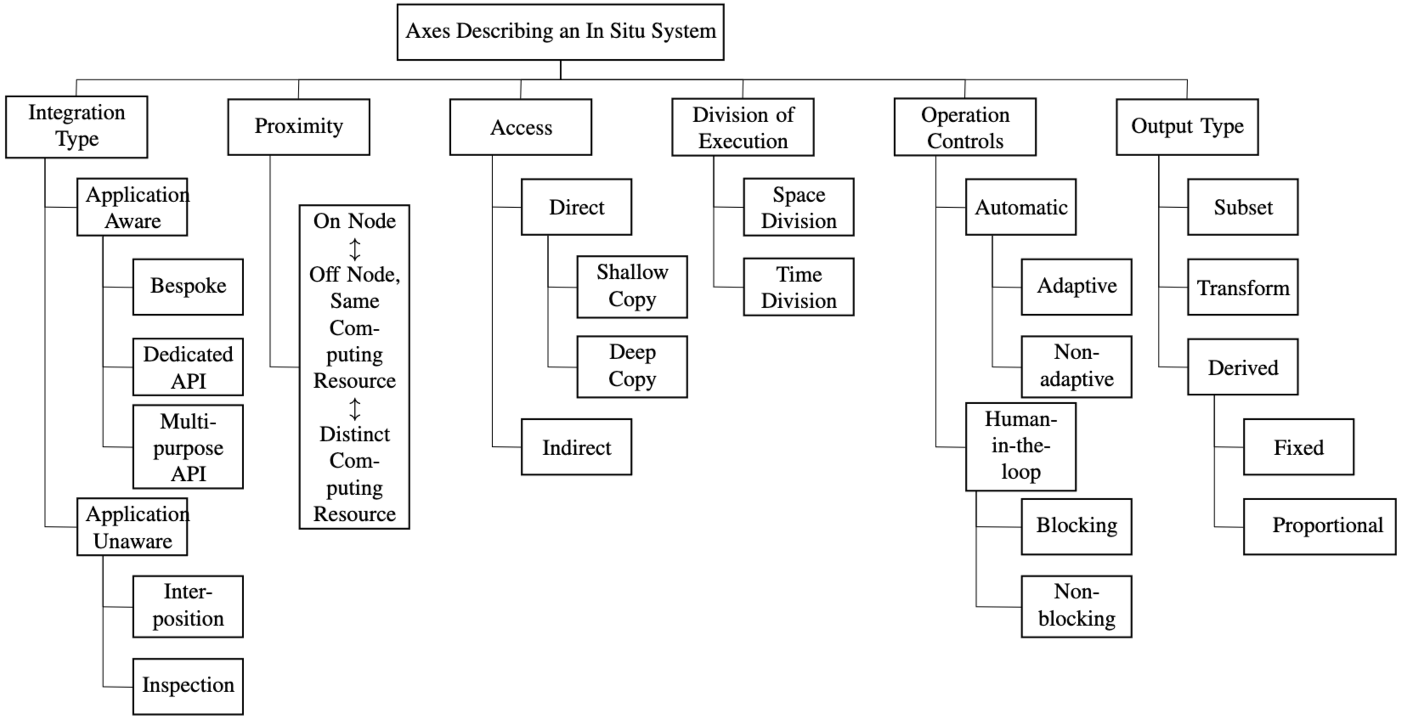


In transit is a flavor of in situ processing that can use additional resources to improve runtime



There are many considerations and flavors of in situ processing

Question: How deep does the rabbit hole go?



Answer: "A Terminology for In Situ Visualization and Analysis Systems", H. Childs, et al.
https://cdux.cs.uoregon.edu/pubs/ChildsIJHPCA.pdf

We need to pass simulation mesh data and user actions to our in situ visualization tool

Mesh Data

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
    elements:
      shape: "tet"
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
  density:
    association: "element”
    topology: ”topo"
    values: [1.0, 2.0]
```

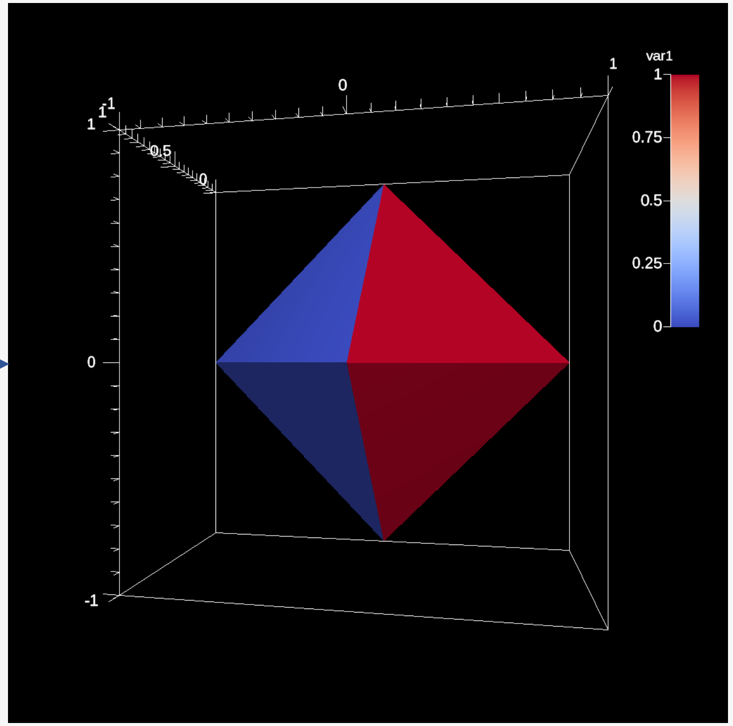
Actions

```
-
  action: "add_scenes"
  scenes:
    my_scene:
      plots:
        my_plot:
          type: "pseudocolor"
          field: "var1"
```

Visualization Tool

Transform, Analyze, Render

Ascent



We need to pass simulation mesh data and user actions to our in situ visualization tool

Mesh Data

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
```

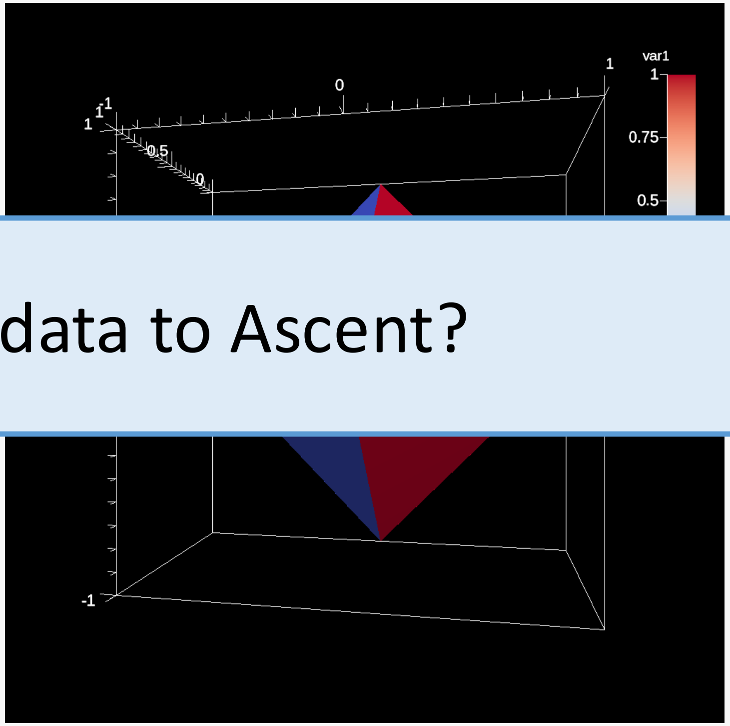
```
    topology: "topo"
    values: [1.0, 2.0]
```

Actions

```
-
  action: "add_scenes"
  scenes:
    my_scene:
      plots:
        my_plot:
          type: "pseudocolor"
          field: "var1"
```

Visualization Tool

Ascent



Question 1: How do we pass simulation mesh data to Ascent?

HPC simulation applications implement and leverage a wide range of mesh data structures and APIs

- A variety of simulation codes leverage their own bespoke in-memory mesh data models.
- Other tools leverage a range of mesh-focused toolkits, frameworks, and APIs including: VTK, VTK-m, MFEM, SAMRAI, AMReX, (and many more …)
- A wide set of powerful analysis tools are mesh agnostic (NumPy, PyTorch, etc) and recasting mesh data into these tools is a challenge
- *A single full-fledged API will never cover all use cases across the ecosystem*

HPC simulation applications implement and leverage a wide range of mesh data structures and APIs

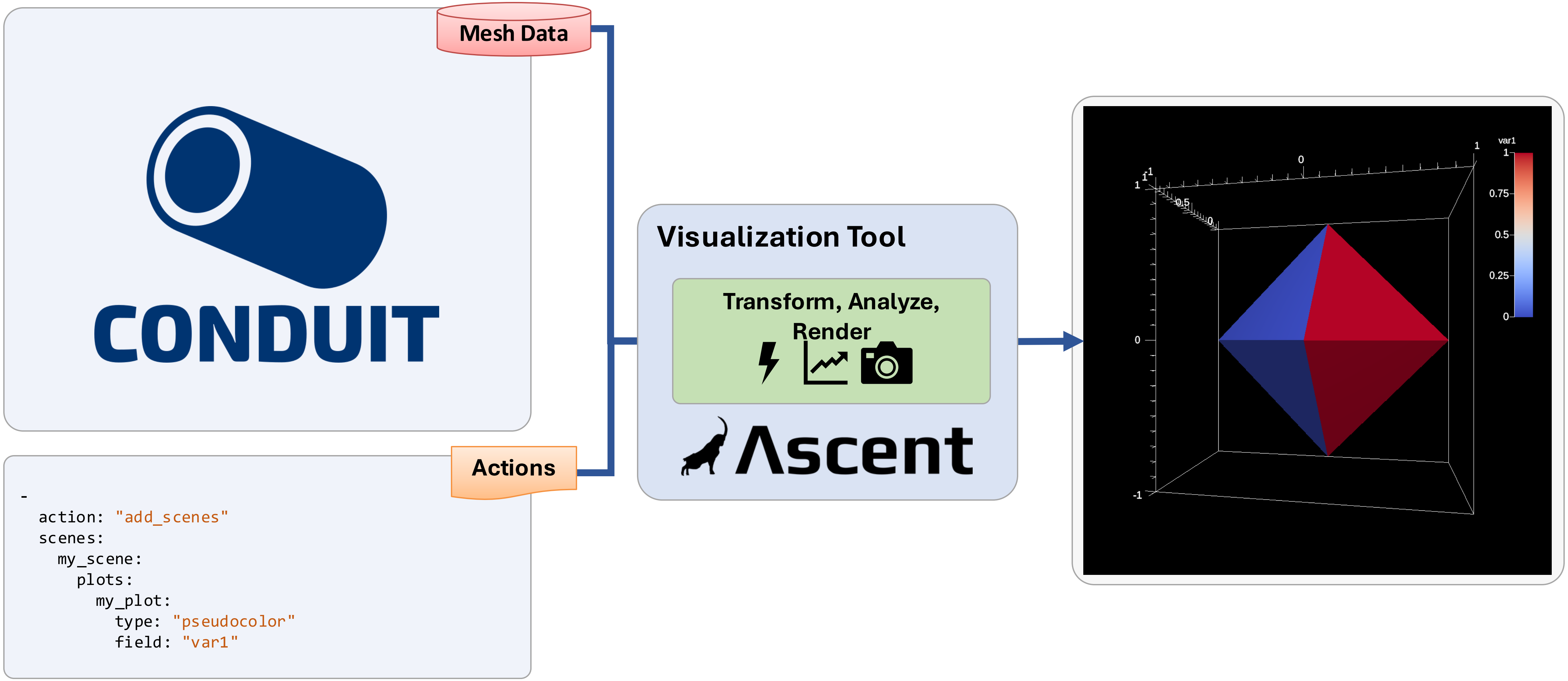
- A variety of simulation codes leverage their own bespoke in-memory mesh data models.

Conduit Mesh Blueprint provides a strategy to describe and adapt mesh data between a wide range of APIs

recasting mesh data into these tools is a challenge

- *A single full-fledged API will never cover all use cases across the ecosystem*

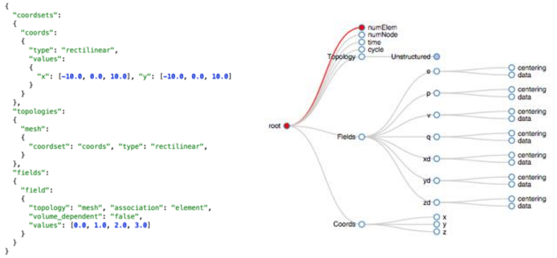
Ascent uses Conduit as a shared interface to describe and accept simulation mesh data



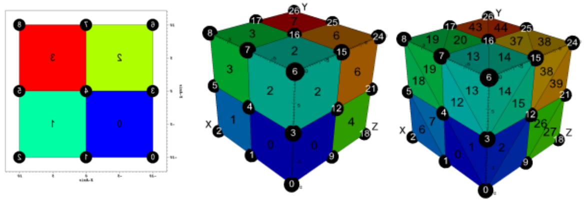
Conduit provides intuitive APIs for in-memory data description and exchange

- **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

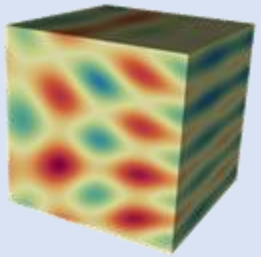
http://software.llnl.gov/conduit
http://github.com/llnl/conduit

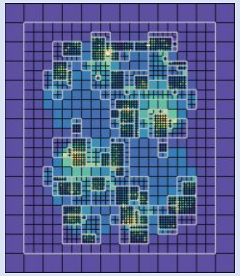
Website and GitHub Repo

The Conduit Blueprint library provides tools to share common flavors of data with Conduit

Blueprint

Supports shared higher-level conventions for using Conduit to represent data



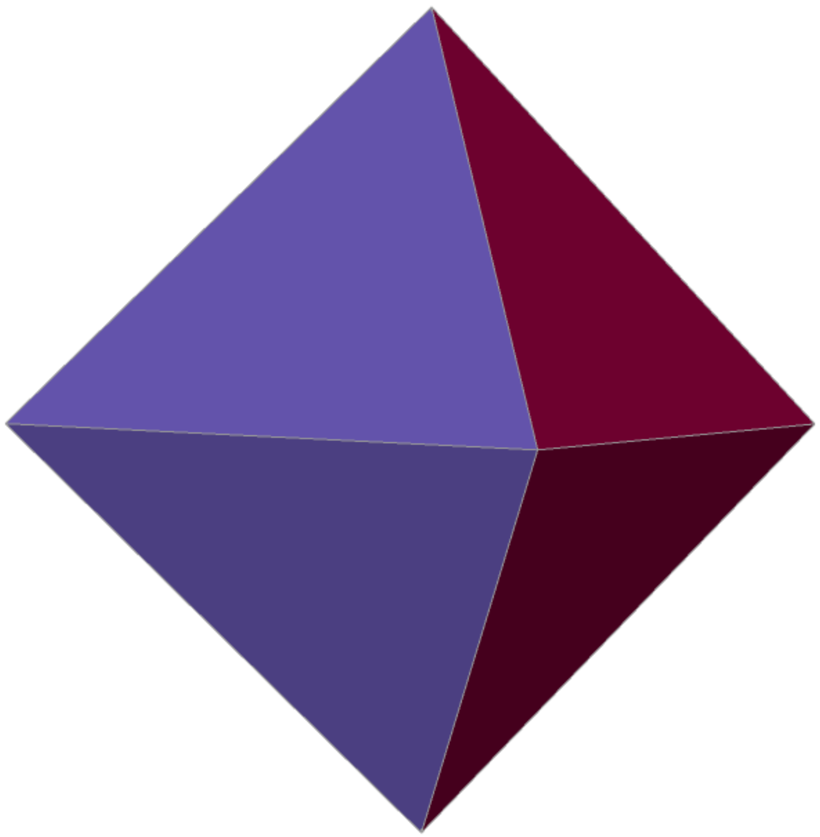


- Computational Meshes
- Multi-component Arrays
- One-to-many Relations
- Example Meshes
- Mesh Transforms

We will share several examples of Conduit “Blueprint” meshes in this tutorial


```yaml
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
    elements:
      shape: "tet"
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
  density:
    association: "element”
    topology: ”topo"
    values: [1.0, 2.0]
```


Example YAML Output



An unstructured tet mesh

We need to pass simulation mesh data and user actions to our in situ visualization tool

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
```

Mesh Data

Question 1: How do we pass simulation mesh data to Ascent?

```
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
```

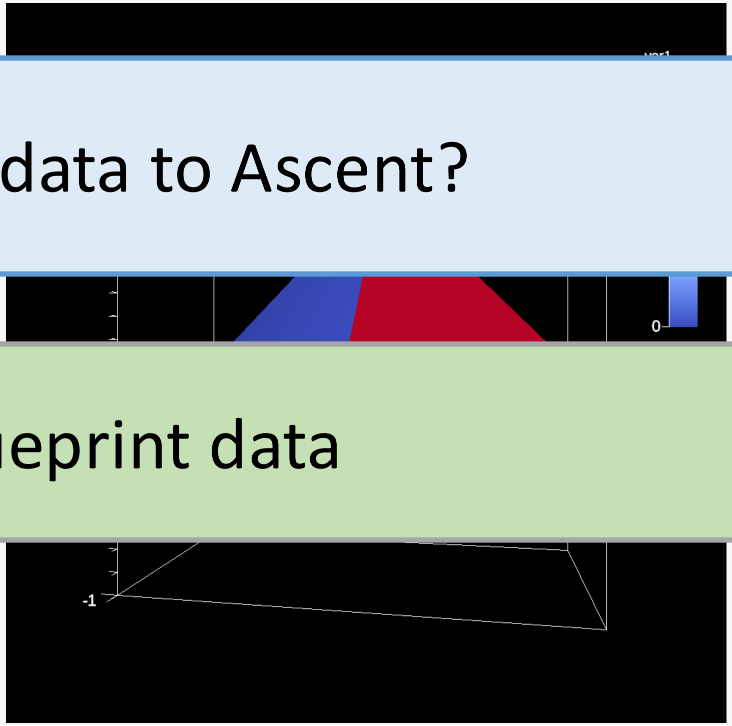
Transform, Analyze,

Answer: Ascent accepts Conduit Mesh Blueprint data

Actions

Ascent

```
-
  action: "add_scenes"
  scenes:
    my_scene:
      plots:
        my_plot:
          type: "pseudocolor"
          field: "var1"
```



We need to pass simulation mesh data and user actions to our in situ visualization tool

Mesh Data

```yaml
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
```

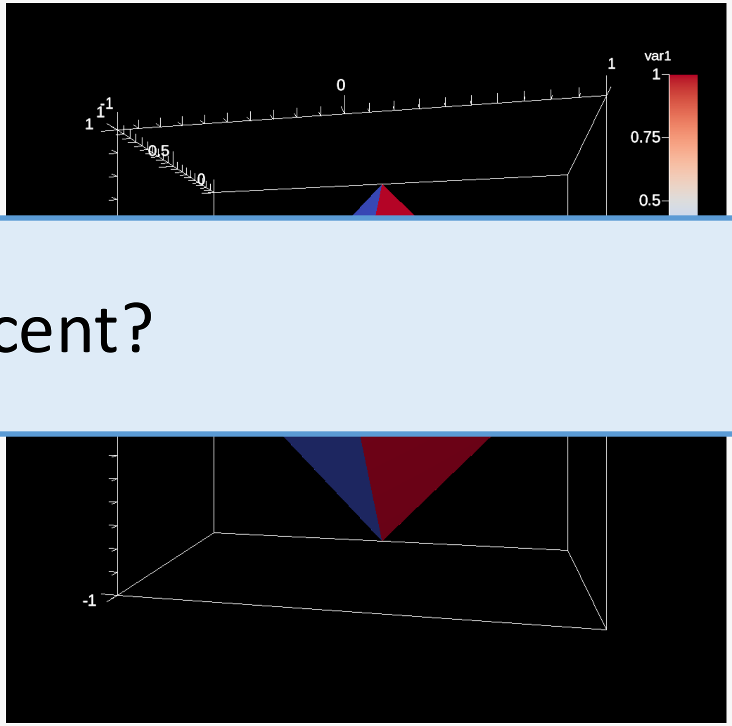
```yaml
    topology: "topo"
    values: [1.0, 2.0]
```

Actions

```yaml
-
  action: "add_scenes"
  scenes:
    my_scene:
      plots:
        my_plot:
          type: "pseudocolor"
          field: "var1"
```

Visualization Tool

Ascent



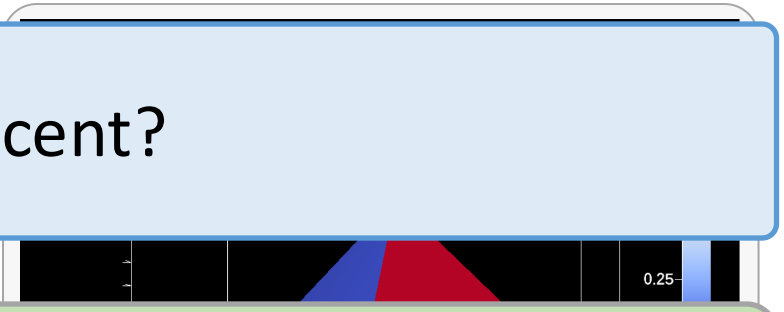
Question 2: How do we pass user actions to Ascent?

We need to pass simulation mesh data and user actions to our in situ visualization tool

Mesh Data

```
coordsets:
  coords:
    type: "explicit"
    values:
```

```
    elements:
      shape: "tet"
```



Question 2: How do we pass user actions to Ascent?

Answer: Ascent uses Conduit to create an Actions API (available in C++, Fortran, Python, and YAML).

You will learn about Ascent's API in the hands-on session.

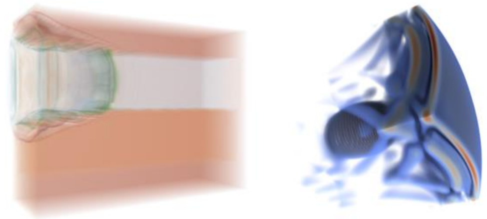
```
      my_plot:
        type: "pseudocolor"
        field: "var1"
```

Ascent

Introduction to Ascent

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, HIP, 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 fewer dependencies than current tools (ex: no OpenGL)



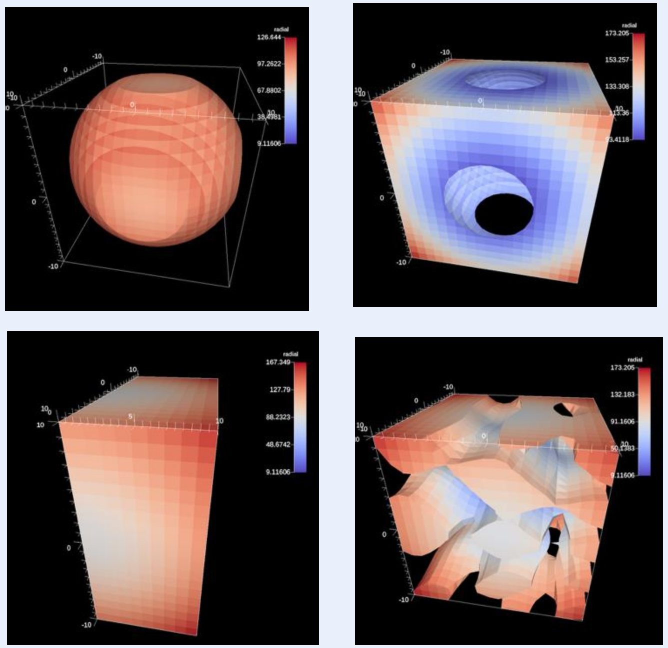
Visualizations created using Ascent

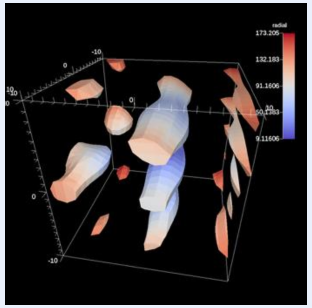
Extracts supported by Ascent

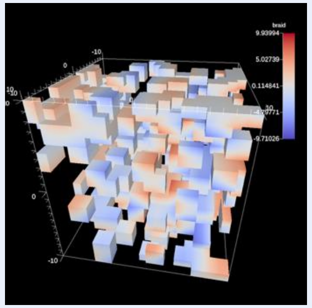
http://ascent-dav.org
https://github.com/Alpine-DAV/ascent

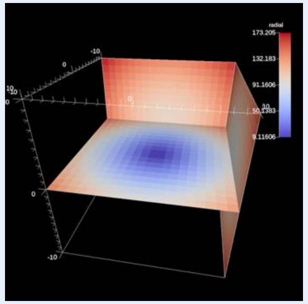
Website and GitHub Repo

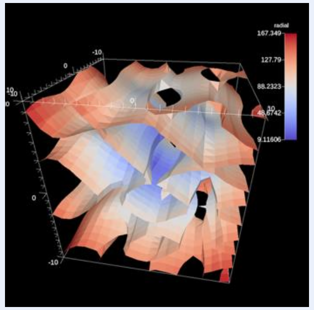
Ascent supports common visualization use cases


Clips


Iso-Volume


Threshold


Slice


Contour

[powered by]

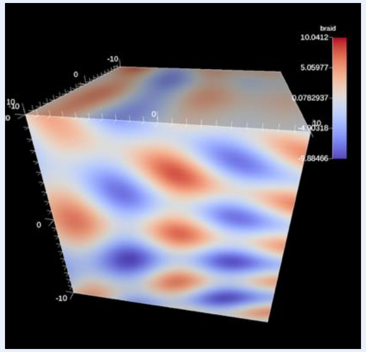
VTK-m

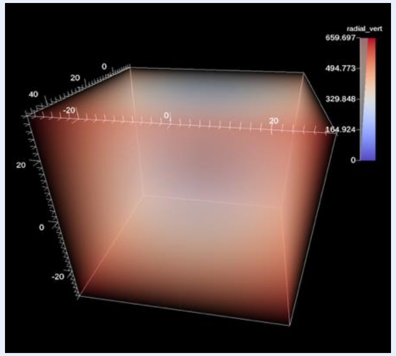
RAJA

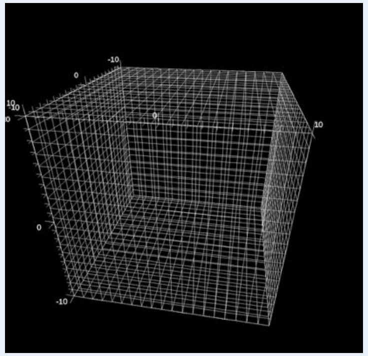
mfem

Devil Ray

Rendering


Pseudocolor

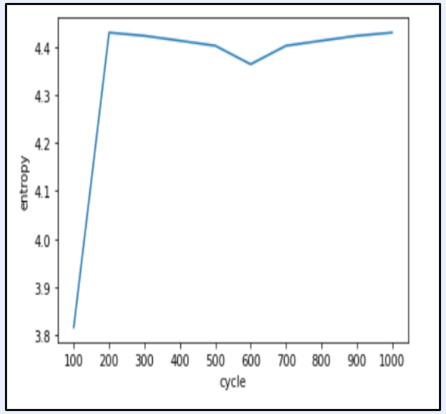

Volume


Mesh

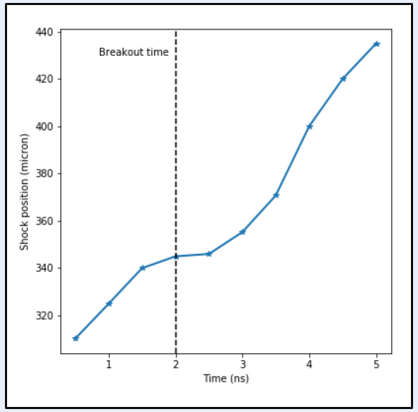
Ascent supports common analysis use cases

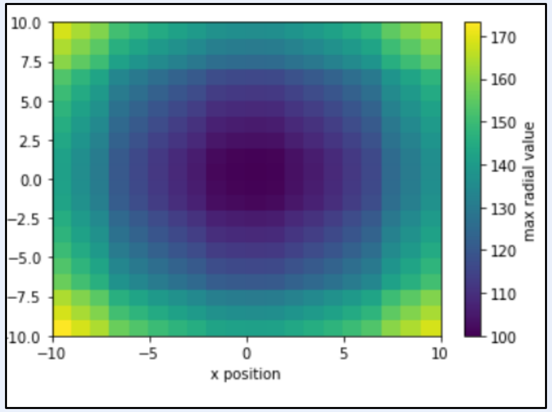
```
expression: |
  du = gradient(field('velocity','u'))
  dv = gradient(field('velocity','v'))
  dw = gradient(field('velocity','w'))
  w_x = dw.y - dv.z
  w_y = dw.z - dv.x
  w_z = dw.x - dv.y
  vector(w_x,w_y,w_z)
name: vorticity
```

Derived Fields



Time Histories





Lineouts and Spatial Binning

```
condition:
  entropy - history(entropy,
                    relative_index = 1) > 0.5
```

Triggers

Extracts

Scalar Images

HDF5 Files

Cinema Databases

Ascent is being used at scale on the DOE's exascale supercomputers

- Ascent is being used for rendering and data reduction on El Captain
  - Ascent extracts and external surfaces extracts are enabling post-hoc movies using VisIt
  - High demand and high hopes for in situ data reduction to support AI/ML efforts
- Ascent was used with NASA FUN3D INCITE runs on OLCF's Frontier and will be used as part of their ALCF Aurora simulations

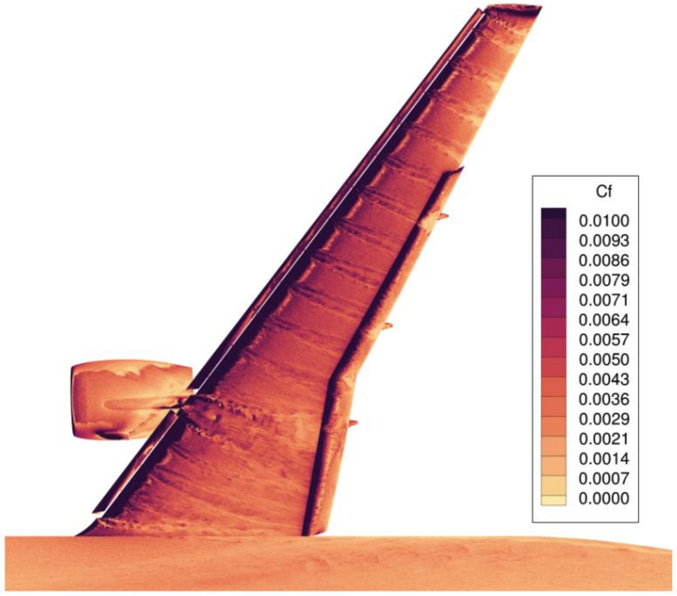


Fig. 12 Wing planform view showing contours of the skin friction magnitude.

Image from Mark Lohry, NASA Langley

Nielsen, E.J., Walden, A., Nastac, G., Wang, L., Jones, W., Lohry, M., Anderson, W.K., Diskin, B., Liu, Y., Rumsey, C.L. and Iyer, P., 2024. Large-Scale Computational Fluid Dynamics Simulations of Aerospace Configurations on the Frontier Exascale System. In *AIAA AVIATION FORUM AND ASCEND 2024* (p. 3866) https://doi.org/10.2514/6.2024-3866

This success is the result of 10+ years of development since the Strawman Viz Proxy App. Ascent owes its success to extensive work across the HPC ecosystem in key libraries: VTK-m/Viskores, Conduit, Devil Ray, MFEM, RAJA, Umpire, and Kokkos.

We released Ascent 0.9.4 this July

Highlights:

- VTK-m 2.3 support
- Extracts: Adding ZFP HDF5 options and adding Silo
- New logging and performance annotation infrastructure & more runtime diagnostic output
- Adding 2d camera view modes for project_2d scalar renderer
- Affine transform filter to rotate, scale, reflect and translate mesh coordinates
- Improvements to default scene cameras
- Improvements to project_2d, uniform grid sampling, slicing, and simulated radiography filters.
- Added camera frustum information of rendered images to Ascent::info()
- Adding support for unstructured topologies with mixed element types
- New external surfaces and point-based sampling filters
- Unified file name formatting options
- H5Z-ZFP Compression Support (1D aware)

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 docs 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:

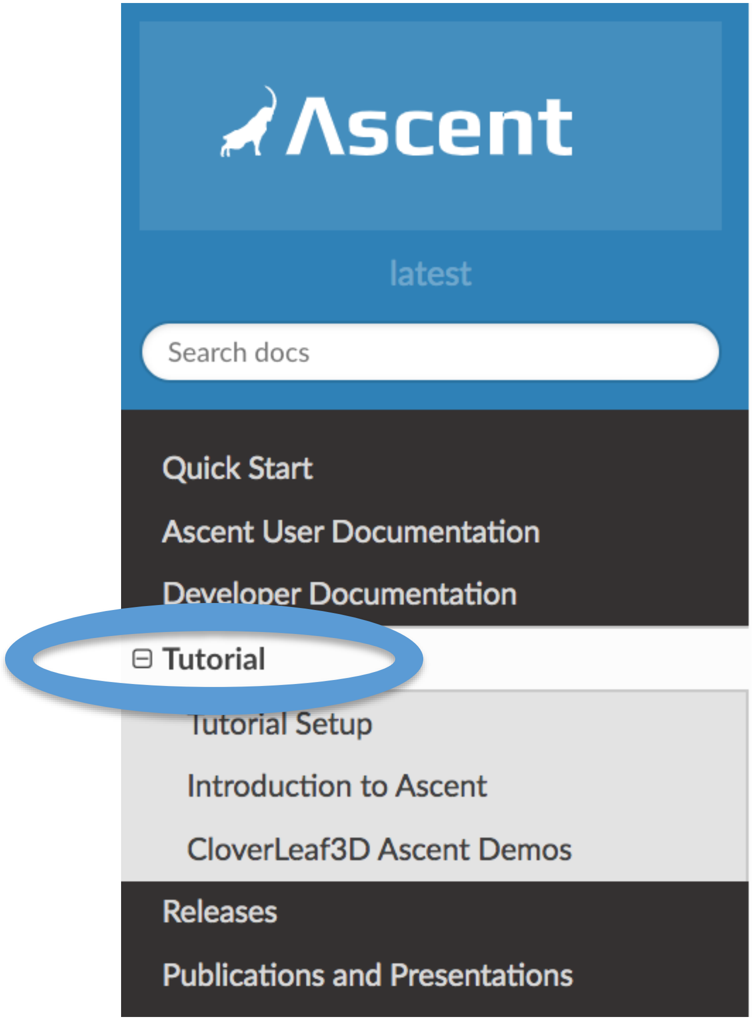
- Formating mesh data for Ascent
- Using Conduit and Ascent's Conduit-based API
- Using and combining Ascent's core building blocks: Scenes, Pipelines, Extracts, Queries, and Triggers
- Using Ascent with the Cloverleaf3D example integration

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 docs 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

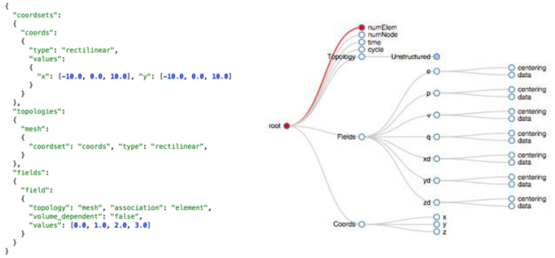
```
//
// Run Ascent
//

Ascent ascent;
ascent.open();

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

ascent.close();
```

The *publish(), execute(),* and *info()* methods take Conduit trees as an argument.

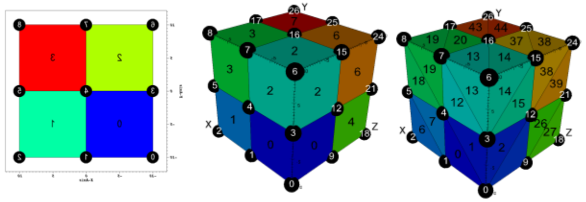
Conduit provides intuitive APIs for in-memory data description and exchange

- **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);
```

Instrument your “main” loop or similar function with access to evolving simulation state

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

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

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

Create an Ascent instance and set it up

```
// 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;
```

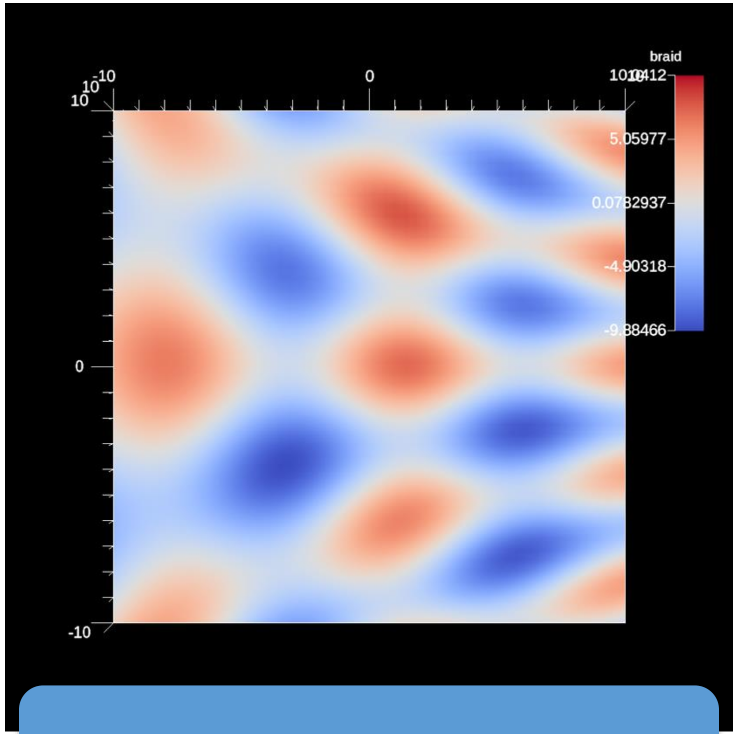
Equivalent YAML Description

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 to users

Scenes
(Render Pictures)

Pipelines
(Transform Data)

Extracts
(Capture Data)

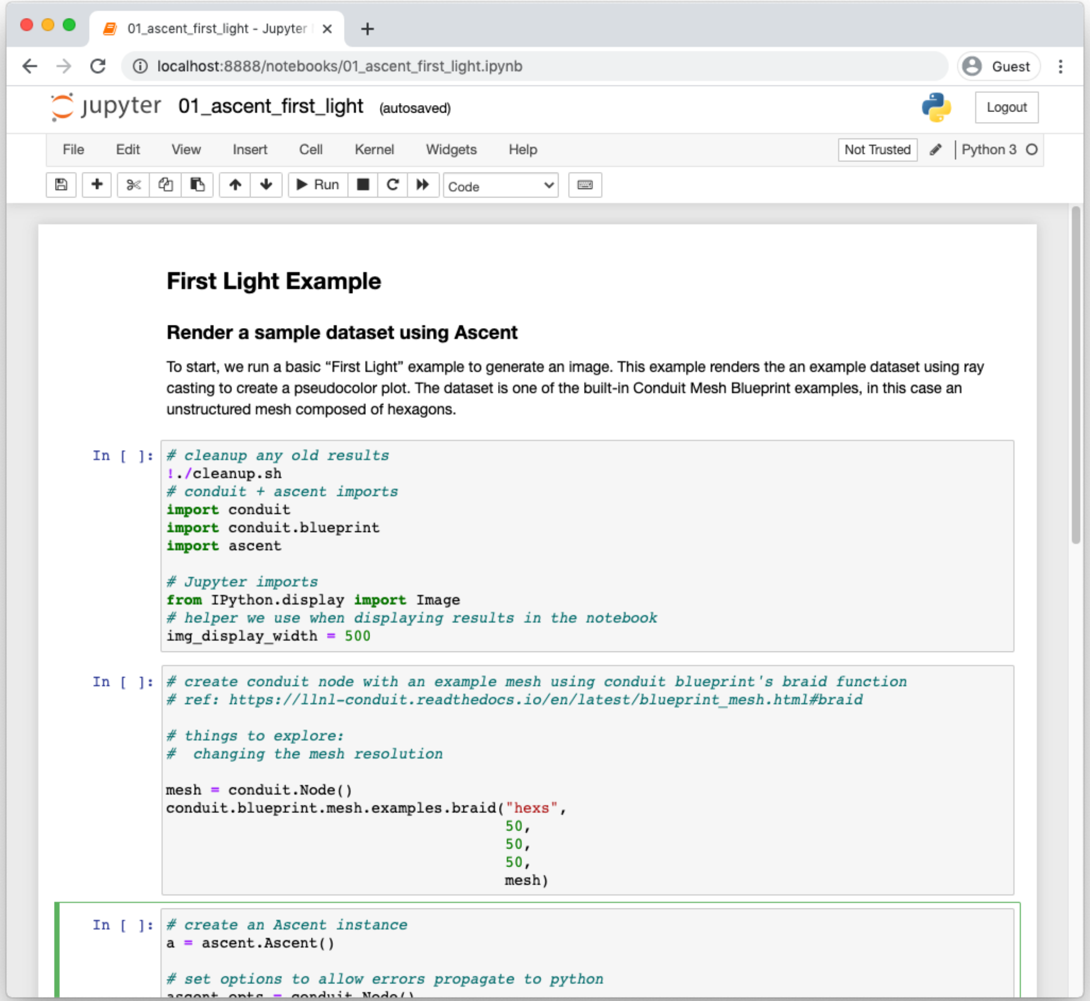
Queries
(Ask Questions)

Triggers
(Direct Actions)

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

NOTE:

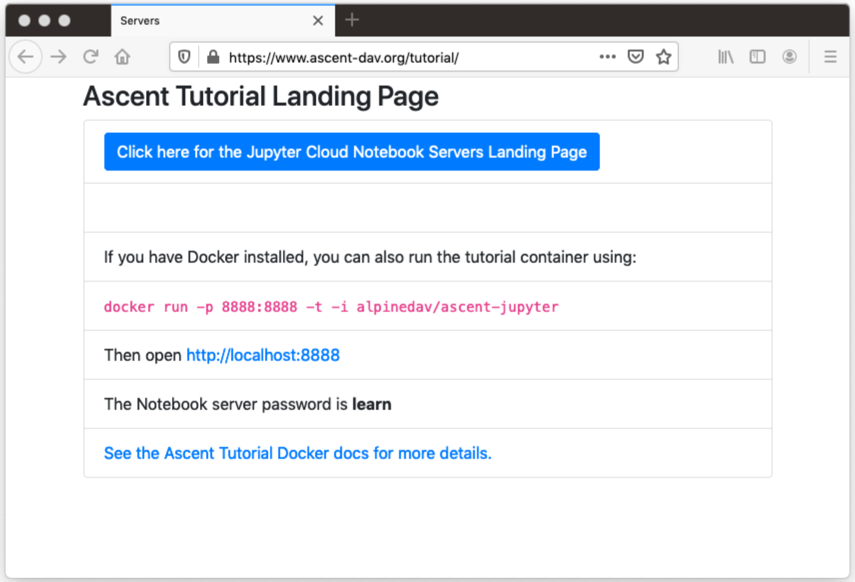
- VPNs or firewalls may block access to general AWS IP addresses and ports
- You may need to disconnect from VPN or request a firewall exemption
- LLNL attendees, you can use the EOR process: https://cspservices.llnl.gov/eor/



You can run our tutorial examples Jupyter Lab via Docker

Start here:

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



Ascent Actions and Runtime Aspects

Principles of the Ascent Actions Interface

- User API is YAML or Conduit (C, C++, Fortran, & Python)
- Provides well defined building blocks:
  Scenes, Pipelines, Extracts, Queries, Triggers
- Uses Hierarchical description linked using unique names
- Aims for a concise set of params for simple cases
  - Examples:
    - Mesh plot only requires a topology name if there are multiple topologies present
    - Azimuth, Elevation, and Zoom provide intuitive way to adjust default camera
- Supports optional parameters for complex cases
  - Example: Detailed Camera parameters

Actions YAML Examples

```yaml
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          field: "var1"
      image_name: "out_scene_ex1_render_var1"
    s2:
      plots:
        p1:
          type: "pseudocolor"
          field: "var2"
      image_name: "out_scene_ex1_render_var2"
```

Actions YAML Examples

```yaml
-
  action: "add_pipelines"
  pipelines:
    pl1:
      f1:
        type: "contour"
        params:
          field: "braid"
          iso_values: [0.200000002980232, 0.400000005960464]
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          pipeline: "pl1"
          field: "braid"
      image_name: "out_pipeline_ex1_contour"
```

Actions YAML Examples

```yaml
-
  action: "add_pipelines"
  pipelines:
    pl1:
      f1:
        type: "threshold"
        params:
          field: "braid"
          min_value: 0.0
          max_value: 0.5
      f2:
        type: "clip"
        params:
          sphere:
            center:
              x: 0.0
              y: 0.0
              z: 0.0
            radius: 12
```

```yaml
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          pipeline: "pl1"
          field: "braid"
      image_name: "out_pipeline_ex2_thresh_clip"
```

Actions YAML Examples

```yaml
-
  action: "add_pipelines"
  pipelines:
    pl1:
      f1:
        type: "contour"
        params:
          field: "braid"
          iso_values: [0.200000002980232, 0.400000005960464]
-
  action: "add_extracts"
  extracts:
    e1:
      type: "relay"
      pipeline: "pl1"
      params:
        path: "out_extract_braid_contour"
        protocol: "blueprint/mesh/hdf5"
```

Actions YAML Examples

```yaml
-
  action: "add_queries"
  queries:
    q1:
      params:
        expression: "entropy(histogram(field('gyre'), num_bins=128))"
        name: "entropy"
-
  action: "add_triggers"
  triggers:
    t1:
      params:
        condition: "cycle() == 500"
        actions_file: "cycle_trigger_actions.yaml"
    t2:
      params:
        condition: "entropy - history(entropy, relative_index = 1) > 0.5"
        actions_file: "entropy_trigger_actions.yaml"
```

Principles of the Ascent Actions Interface

- Blueprint naturally supports multiple topologies with complex domain decompositions
- All Filters must support domain-decomposed meshes (including empty cases)
- VTK-m, Devil Ray, and RAJA are used for Device (GPU) Execution
- Provides an expression language (DSL) that underpins Queries, Triggers, and can be used for filter parameters
- Data Flow Networks are used for execution planning and execution
  - Filter inputs are arbitrary
  - Intermediate results are tracked and released when they are no longer needed
  - Creative execution supports JIT expressions (prototype / limited cases)
- Filters can request data in several forms:
  - Conduit Blueprint / LOR Conduit Blueprint (uses MFEM) / VTK-m / Devil Ray