ECP Alpine: Algorithms and Infrastructure for In Situ Visualization and Analysis

Presented By: Matt Larsen

LLNL-PRES-731545
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

Outline

- **Alpine Overview**
- Alpine In Situ
- VTK-h
- Current State
- Addressing in situ constraints

What is Alpine?

- Exascale computing project
- ~ 6M in funding over 3 years
- Goals
  - Infrastructure
    - Create a common ecosystem for visualization development
      - Algorithms written once are deployed in VisIt and ParaView
  - Algorithms
    - Implement production algorithms for exascale environments
      - E.g., time and memory constraints

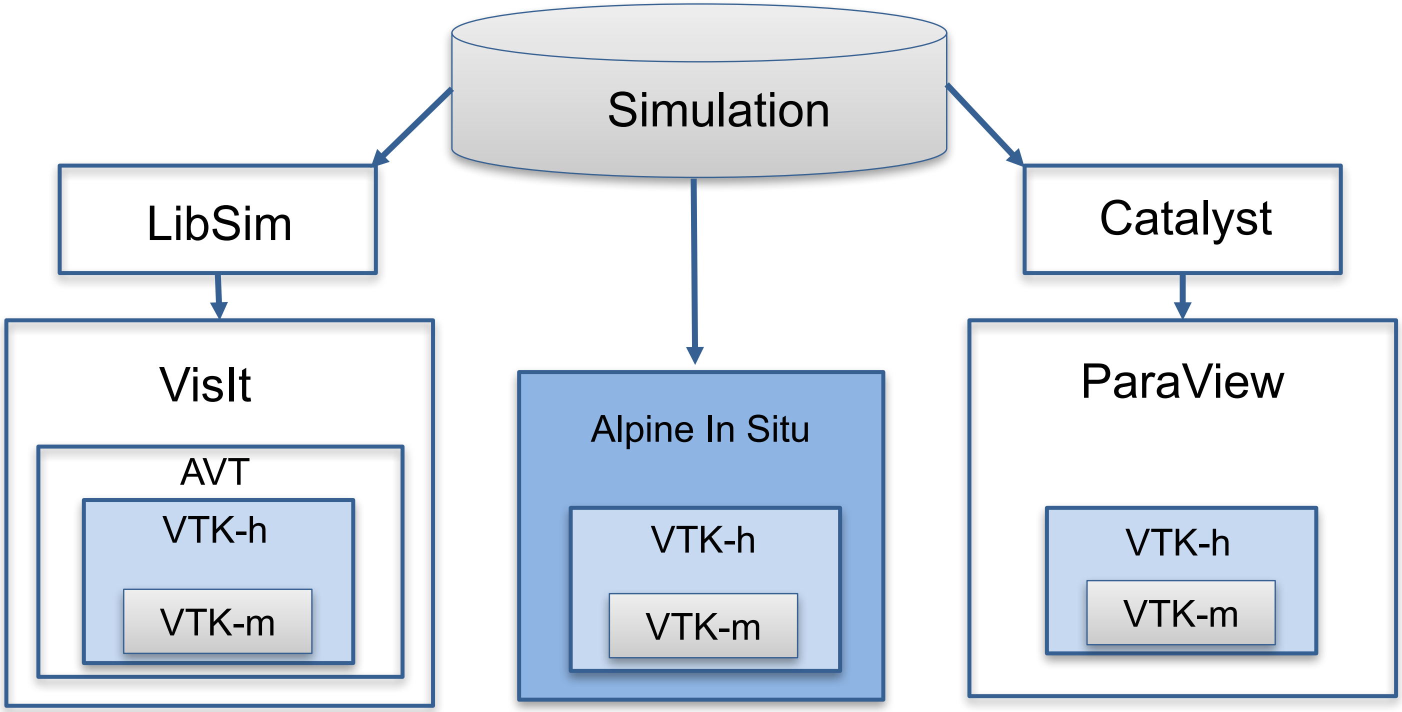
Alpine contributors

- Los Alamos National Laboratory (LANL)
  - James Ahrens (PI), Roxana Bujack, Jon Woodring
- Lawrence Livermore National Laboratory (LLNL)
  - Eric Brugger, Matt Larsen
- University of Oregon (UO)
  - Hank Childs
- Kitware, Inc.
  - Berk Geveci, Utkarsh Ayachit, Reid Porter
- Lawrence Berkeley National Laboratory (LBNL)
  - Gunther Weber, Oliver Ruebel

Alpine major components

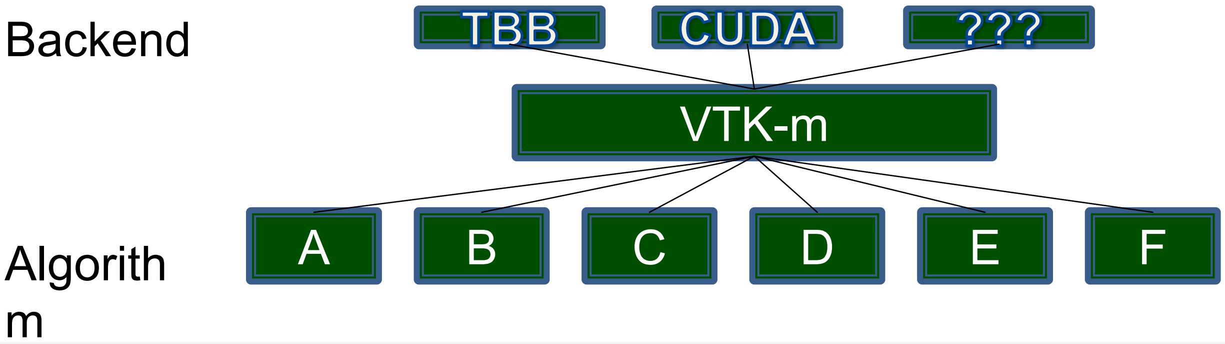
- VTK-m
  - Separate ECP software technology project for node-level parallelism
- VTK-h
  - Distributed memory layer build on top of VTK-m
- Alpine In Situ
  - Flyweight interface for VTK-h

Where does the Alpine project fit in the larger ecosystem?



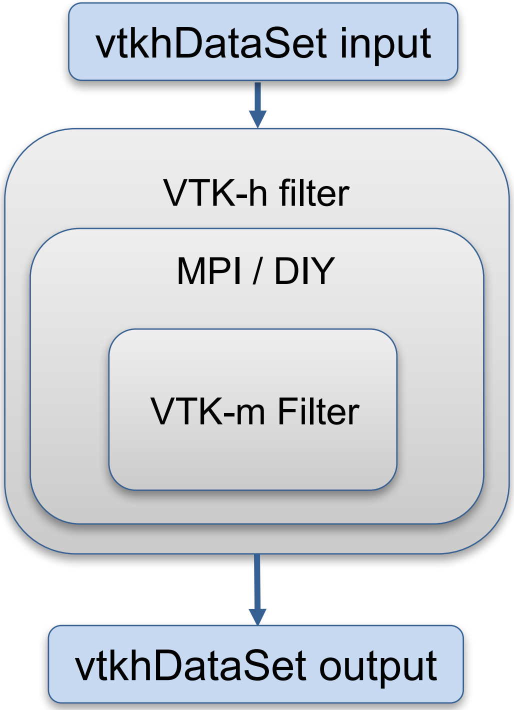
Alpine major components: VTK-m

- "m" for many-core
- Provides a data parallel abstraction
  - Algorithms composed of data parallel operations
  - Programming for portable performance
    - TBB and CUDA
- Flexible and efficient mesh data model



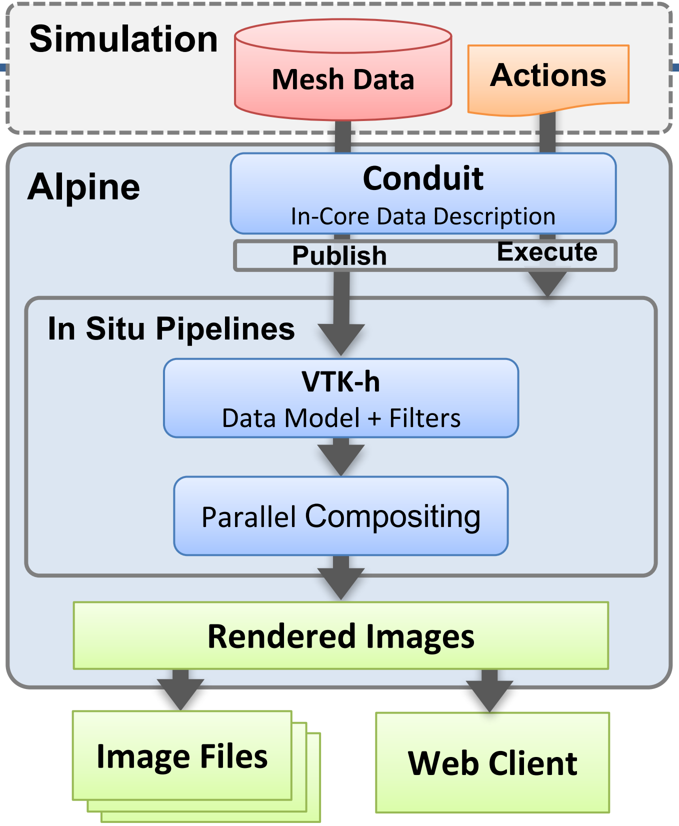
Alpine major components: VTK-h

- “h” for hybrid parallel
- Distributed memory layer on top of VTK-m filters
  - MPI or DIY (do-it-yourself analysis)
- Library provides distributed:
  - data model
  - filters



Alpine major components: Alpine In Situ

- "h" for hybrid parallel
- Distributed memory layer on top of VTK-m filters
  - MPI or DIY (do-it-yourself analysis)
- Library provides distributed:
  - data model
  - Distributed filters



Outline

- Alpine Overview
- **Alpine In Situ**
- VTK-h
- Current State
- Addressing in situ constraints

Alpine prototype is based on Strawman

Why should you care?

- Flyweight in situ analysis library
  - Low simulation code footprint
  - Removes need for VisIt and ParaView dependencies
- Modular pipelines
  - VTK-h pipeline
  - HDF5 pipeline
  - [insert custom analysis here]
- Multiple languages bindings
  - C, C++, FORTRAN, Python

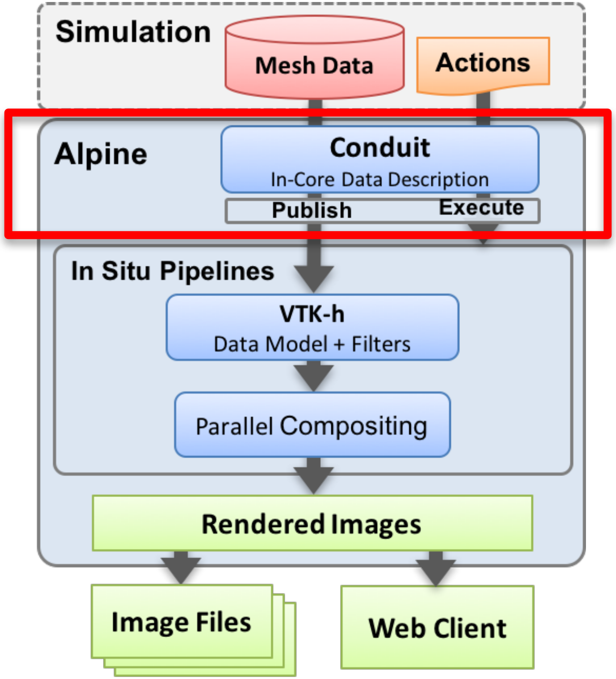
We collected requirements for tightly coupled in situ use cases.

- **3 Categories**
  - Portability
    - Architectures, languages, mesh types
  - Usability
    - Reduce integration time, data ownership, run-time control, easy to consume results
  - Minimal burden on simulation
    - Execution time, memory usage

See ISAV2015 paper for full list of requirements

What is the integration burden?

	Simulation Codes			
	LULESH	Kripke	CloverLeaf3D	Ares
Data Description	15	21	39	42
Action Descriptions	14	14	14	14
Alpine API Calls	7	7	9	7
Total Lines of Code	**36**	**42**	**62**	**63**

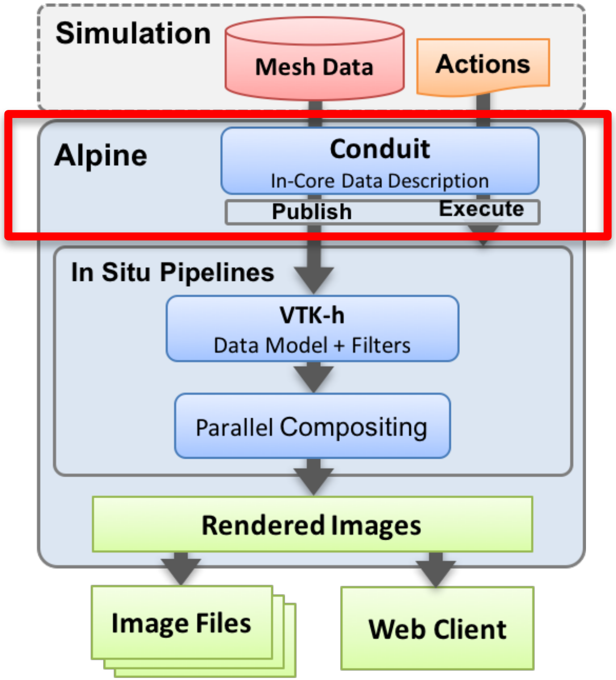


Conduit is used for in-core data description.

https://github.com/llnl/conduit

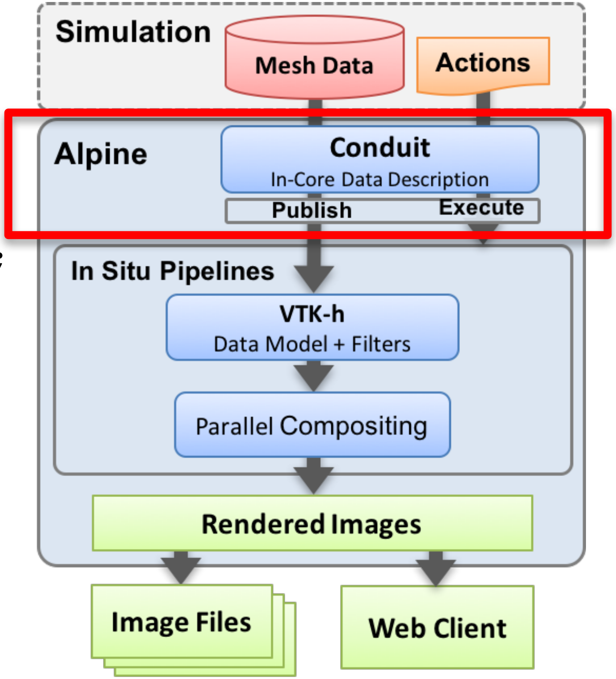
Conduit Provides:

- JSON style object model
- Type standardization (e.g., float64)
- Separates data and description
- Run-time focused
- Multiple language APIs



Integration example: Alpine in situ API calls

```
Alpine alpine;
Node options;
Options["mpi_comm"] = mpi_comm_handle(MPI_COMM_WORLD);
alpine.Open(options);
alpine.Publish(data);
alpine.Execute(actions);
alpine.Close();
```



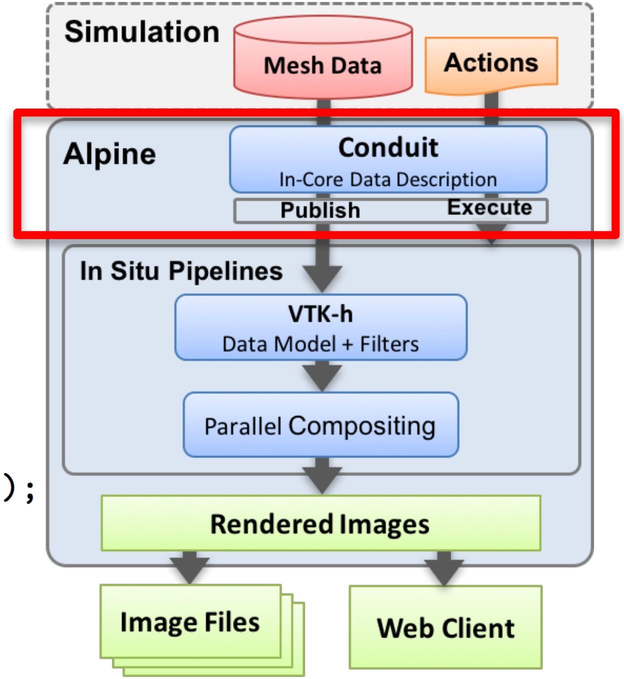
Meshes are described using the Conduit Mesh “Blueprint”

- **Coordinate Sets:**
  - Implicit: Uniform, Rectilinear
  - Explicit
- **Topologies:**
  - Implicit: Uniform, Rectilinear, Structured
  - Unstructured
    - Zoo Elements + Polygons and Polyhedra
- **Fields:**
  - Centerings and associated cells sets

The Blueprint provides a general set of conventions that allow us to easily target concrete APIs (VTK, VTKm, Silo, ADIOS, etc)

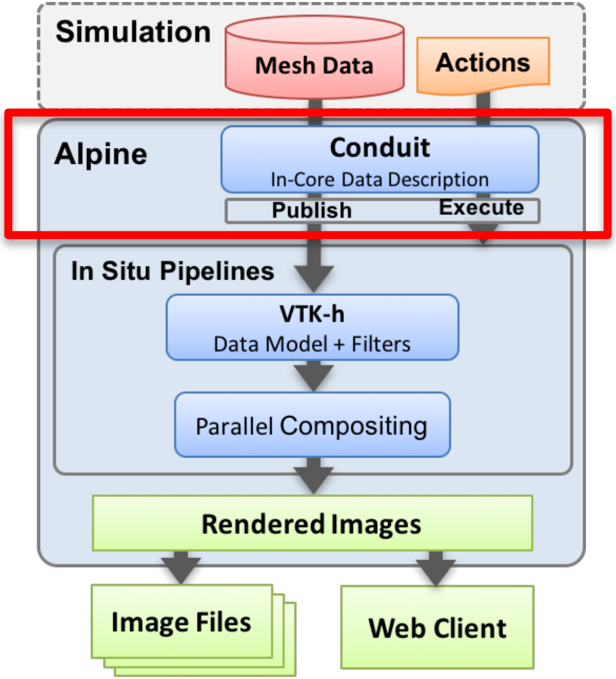
Integration Example: Describing LULESH's data

```
conduit::Node data;
data["state/time"].set_external(&time);
data["state/cycle"].set_external(&cycle);
data["state/domain"] = my_mpi_rank;
data["coords/type"] = "explicit";
data["coords/x"].set_external(x);
data["coords/y"].set_external(y);
data["coords/z"].set_external(z);
data["topology/type"] = "unstructured";
data["topology/coordset"] = "coords";
data["topology/elements/shape"] = "hexs";
data["topology/elements/connectivity"].set_external(nodelist);
data["fields/e/association"] = "element";
data["fields/e/type"]        = "scalar";
data["fields/e/values"].set_external(e);
```



Integration Example: Describing in situ actions

```
conduit::Node actions;
conduit::Node &add = actions.append();
add["action"] = "add_plot";
add["var"] = "pressure";
char file_name[30];
sprintf(file_name, "image%04d", cycle);
add["render_options/file_name"] = file_name;
add["render_options/width"] = 1024;
add["render_options/height"] = 1024;
conduit::Node &draw = actions.append();
draw["action"] = "draw_plots";
```



Version 0.1.0 Released

- Source code:
  - https://github.com/Alpine-DAV/alpine
  - Use the "develop" branch

- 3 included pipelines:
  - VTK-m (rendering)
  - HDF5 (I/O)
  - Empty (insert your code here)

Outline

- Alpine Overview
- Alpine In Situ
- **VTK-h**
- Current State
- Addressing in situ constraints

VTK-h

- ”H”brid parallel
  - Parallel across nodes (MPI)
  - Parallel on-node (VTK-m: CPU and GPU)
- Single environment to deploy algorithms
  - Deployed in:
    - Alpine In Situ
    - ParaView
    - VisIt

4 algorithmic focus areas in VTK-h

- Data selection
  - What subset of the data is interesting?
    - Feature centric analysis
    - Topological analysis
- Data reduction
  - Adaptive sampling
  - Lagrangian analysis (flow visualization)

What else will be available?

- Filters
  - Isosurfaces
  - Gradients
  - Histograms
  - And many more to come

- Data reduction
  - Image databases (Cinema)

Outline

- Alpine Overview
- Alpine In Situ
- VTK-h
- **Current State**
- Addressing in situ constraints

Current state

- In quarter 2 of a three year project
- Alpine In Situ prototype is released
  - Currently only rendering
  - https://github.com/Alpine-DAV/alpine
- Upcoming milestones
  - Y1/Q3: in situ algorithms prototypes
  - Y1/Q4: Alpine in situ API released
  - Y2/Q1: Initial release of Alpine

Outline

- Alpine Overview
- Alpine In Situ
- VTK-h
- Current State
- **Addressing in situ constraints**

Performance modeling of in situ rendering

- Assumptions:
  - Visualization and analysis will be increasingly performed in situ
  - Visualization and analysis will need to occur within simulation constraints

"Can your visualization routines run within my simulation code's constraints?"

“Can your visualization routines run within my simulation code’s constraints?”

- Current answers:
  - Honest: “I don’t know”
  - Reckless: “Let’s try it and see!”
  - Anecdotal: “I ran something similar before and worked.”
  - Extr
- Better
  - “I k
    - “

-or-

  - “I know the answer is no, and here’s why…”
    - “And if you want it to work, then here are the options…”

We believe performance modeling is a very promising approach for achieving the “better answers.”

How long does it take to render?

- Depe
  - re
  - co
    -
    -
  - w
    -
    -
    -
- But, e s

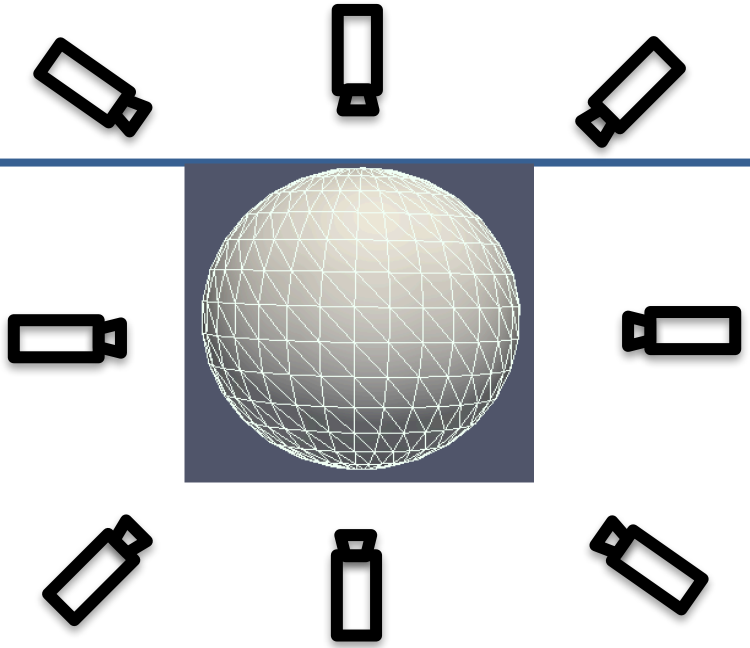
A new paradigm is emerging that requires rendering many, many images.

Increases rendering time by orders of magnitude.

This motivates need for performance modeling of rendering.

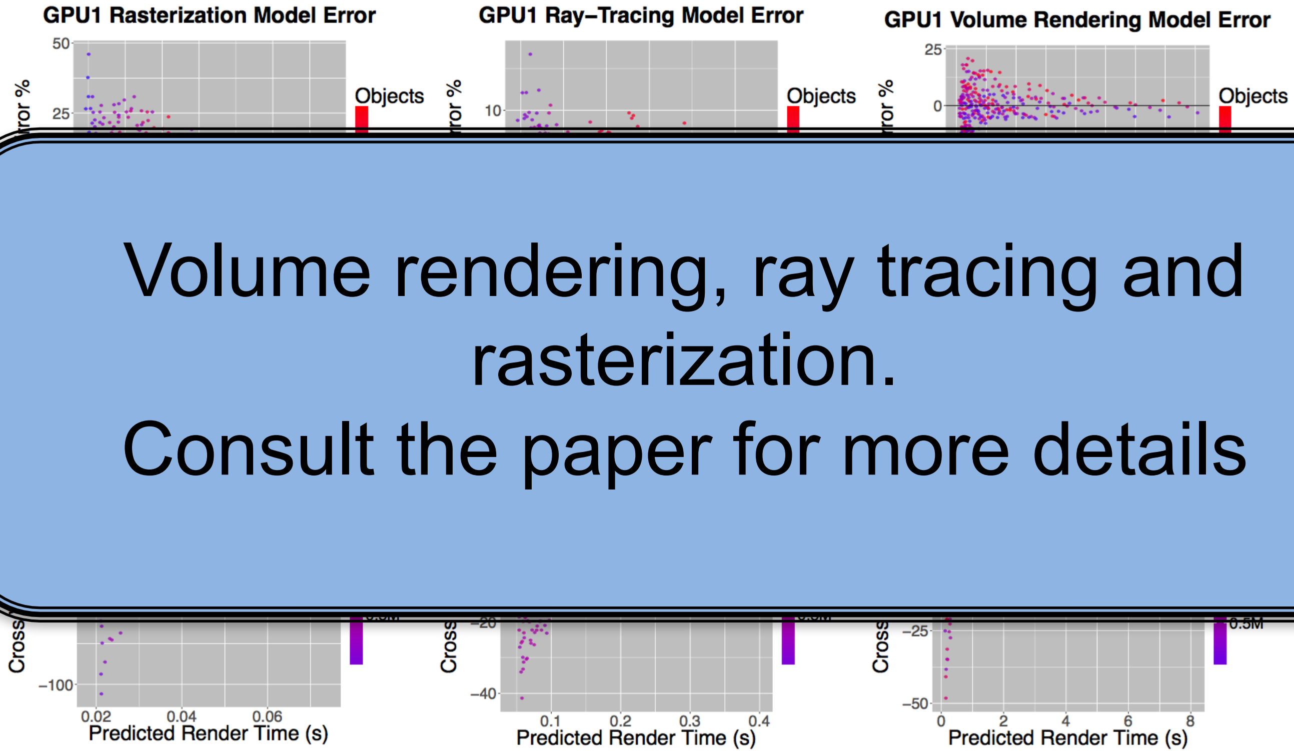
Cinema (LANL – cinemascience.org)

- Images as a form of data compression
  - Simulation mesh size > 10^{15}
  - Image size about 10^6
- Many camera angles
- Many operations
  - Contours
  - Slices
- Creates an interactively explorable image database
  - Can be explored in post-hoc manner



"An image-based approach to extreme scale in situ visualization and analysis," Ahrens et. al, SC14.

Three models and two architectures

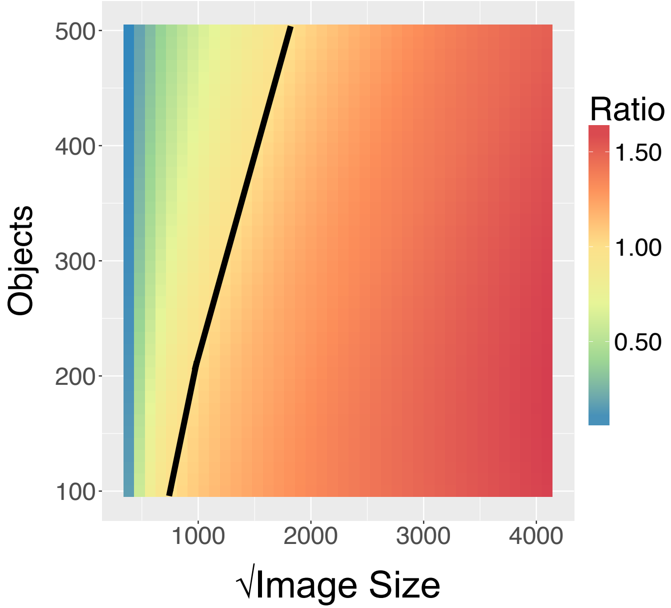


Volume rendering, ray tracing and rasterization.

Consult the paper for more details

With models we can ask questions: ray tracing versus rasterization

- 32 MPI ranks
- 100 images
  - One time initialization for ray-tracing is amortized
- Ray tracing
  - Wins when
    - Number of objects is large
    - Lower resolutions



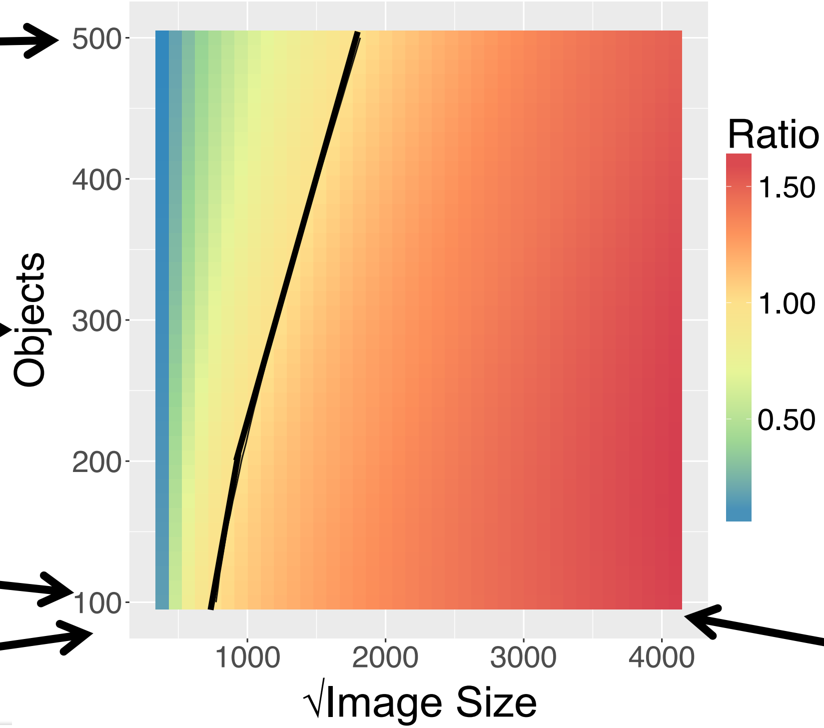
CPU ray tracing versus rasterization

125 M Triangles

27 M Triangles

1 M Triangles

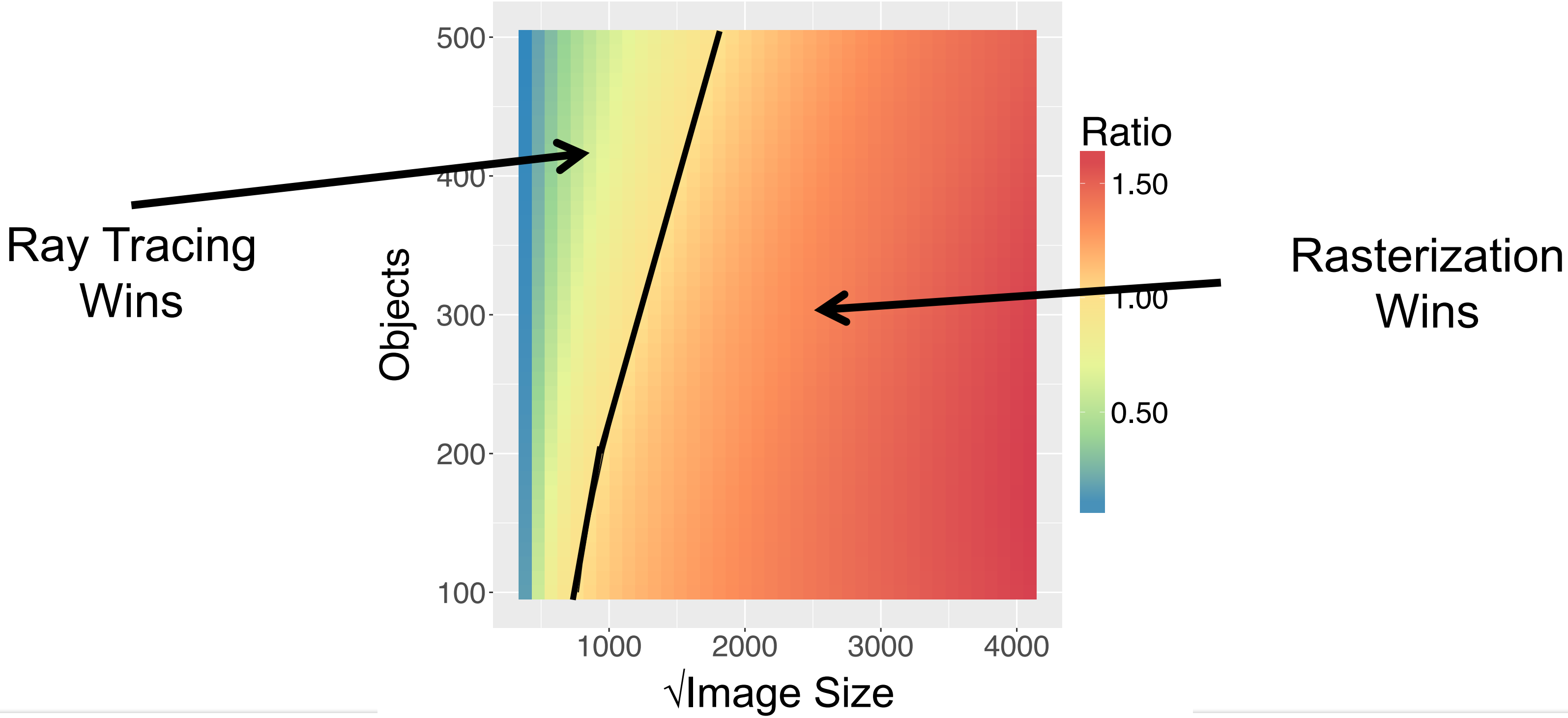
384^2 Resolution



4000^2 Resolution

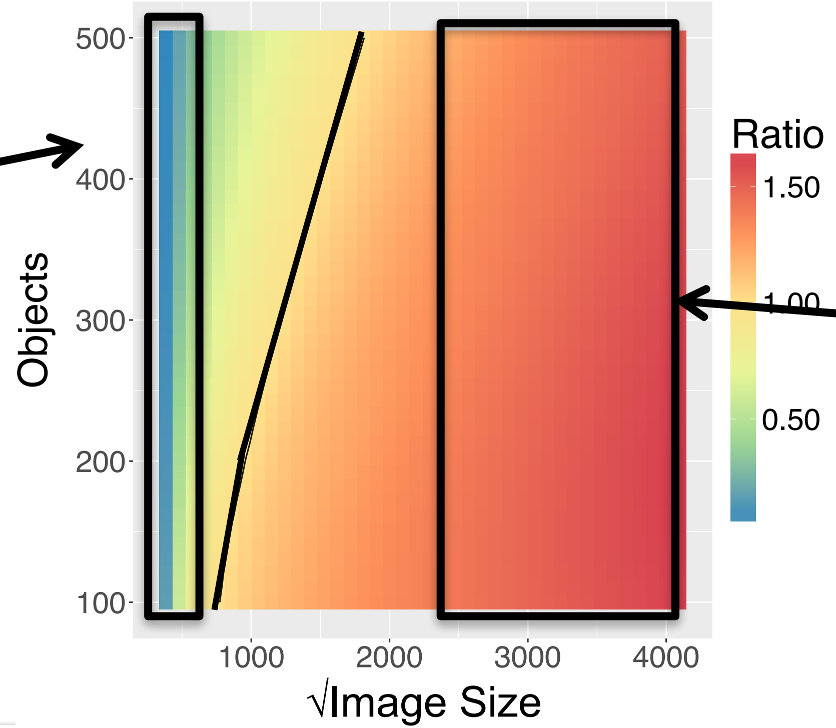
$$\text{Ratio} = \frac{\text{Ray Tracing Render Time}}{\text{Rasterization Render Time}}$$

CPU ray tracing versus rasterization



CPU ray tracing versus rasterization

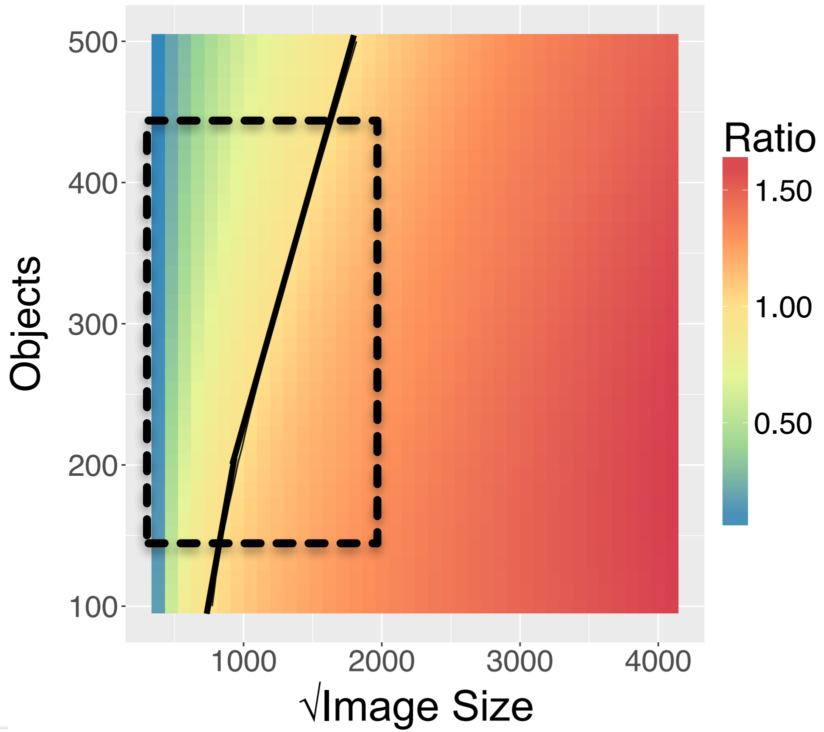
Ray Tracing
~10x More Images
At Lower
Resolutions

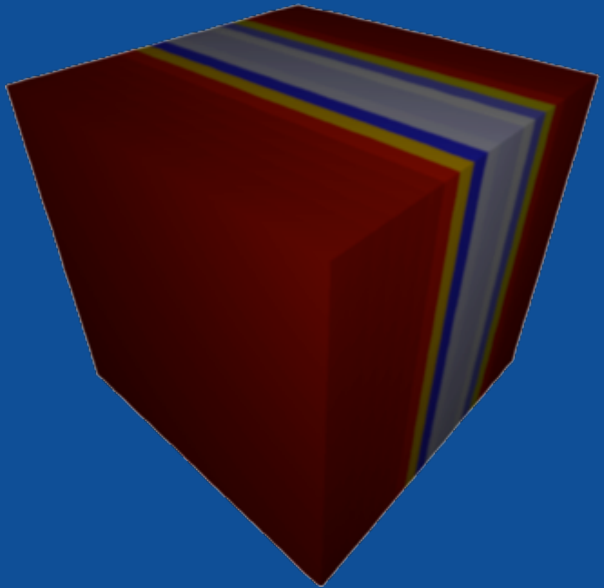


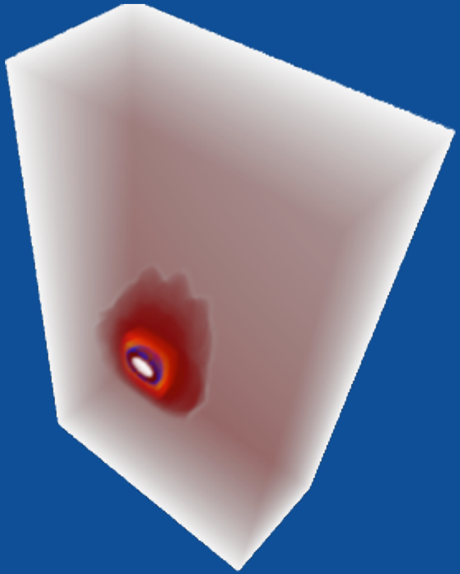
Rasterization
~1.5x More Images
At High Resolutions

CPU ray tracing versus rasterization

Common Configurations







Lawrence Livermore
National Laboratory