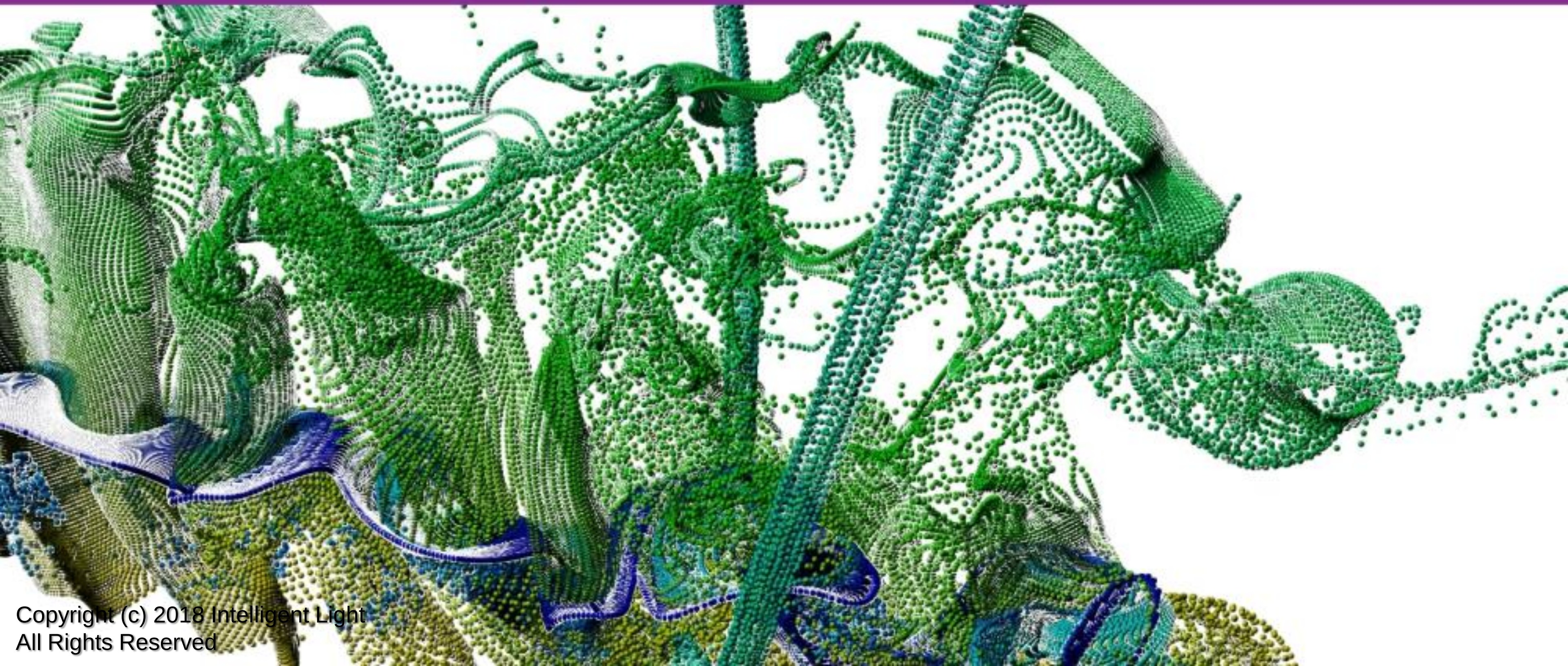
Applied Research Group

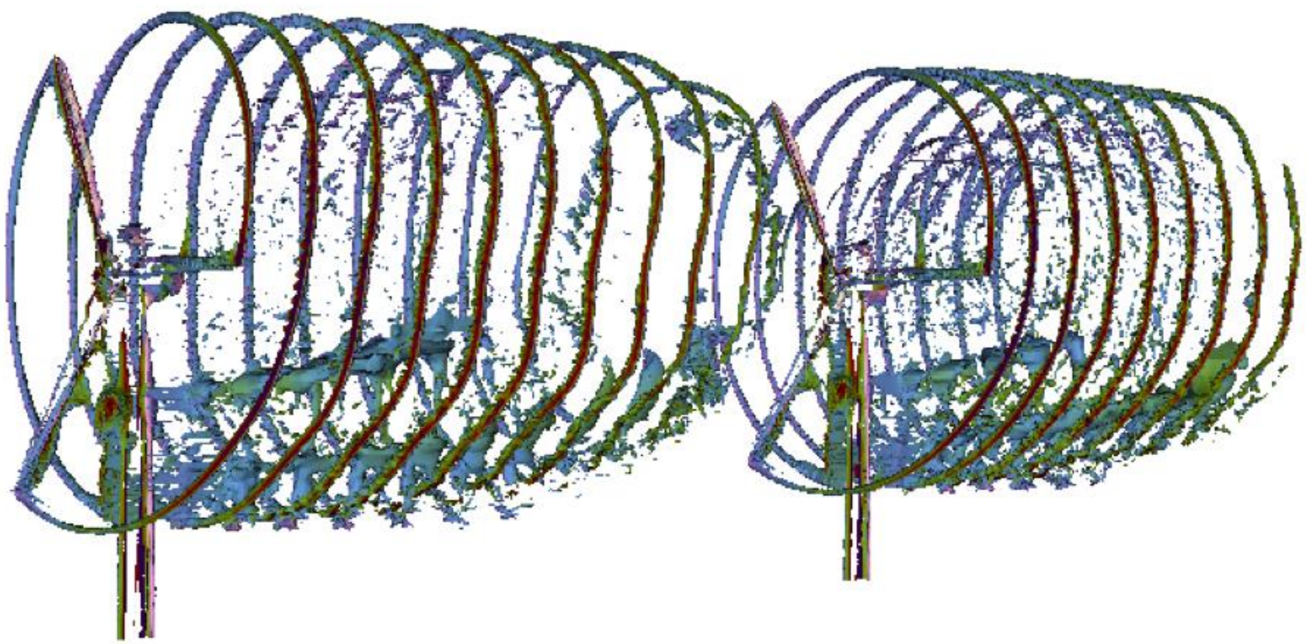
Seeking Answers, Deploying Solutions

Intelligent Light



USING LIBSIM FOR SCALABLE IN SITU

Brad Whitlock
Intelligent light



Intelligent Light

- Established in 1984
  - Three decades in the software & services business
  - **FieldView** launched in 1990
  - **VisIt/libsim** support since 2014
  - Global Customer Base
  - Multiple CFD practitioners on staff
- We bridge CFD & IT for customers

IT

CFD Solvers

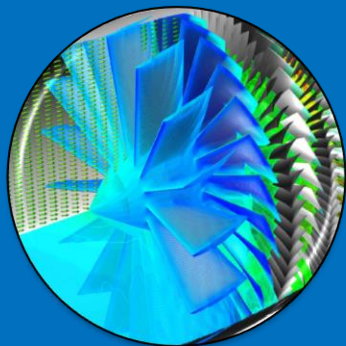
Aerospace & Defense

Automotive & Transport

Consumer & Heavy Industry

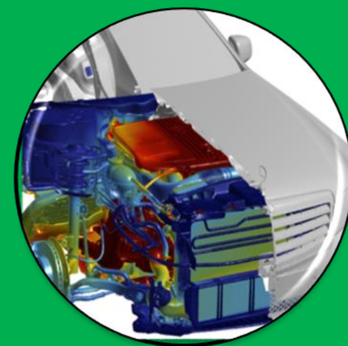
Energy & Environmental

Serving the Global CFD Community



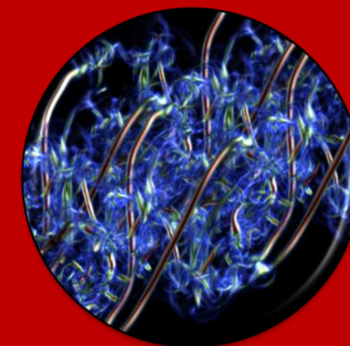
Software Products

- **FieldView** family, from laptops to HPC (licensed)
- **VisIt** scalable, advanced visualization (open source)
- Best-in-class global **customer support**



Custom Engineered Solutions

- Training (on-site, via the web or at Intelligent Light)
- Script development
- Customize FieldView & VisIt
- Workflow Automation & Optimization



Applied Research Group (ARG)

- R&D in advanced post-processing & CFD methods
- Teamed with global experts
- Feeds technology into FieldView and VisIt

Helping our customers to <u>do more with less</u> and <u>make better decisions</u>

Libsim puts VisIt in situ

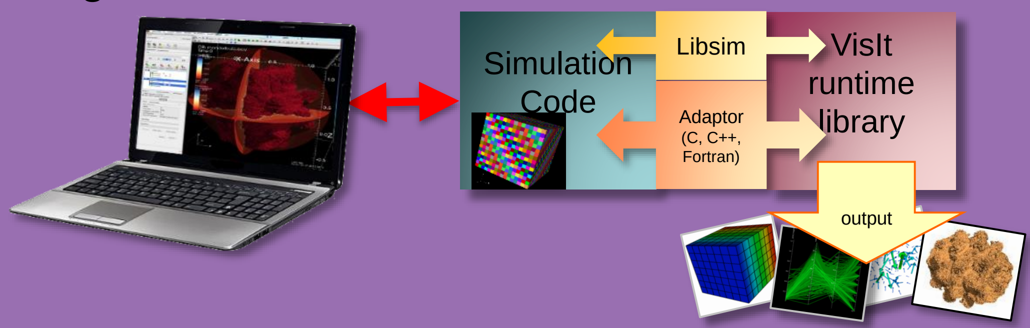
- Libsim lets VisIt connect to simulations and access their data
- Scales to 100K processors and beyond
- Avoids I/O and data movement
- Supports automated data product generation

VisIt

- Versatile open source software for visualizing and analyzing petascale simulation datasets

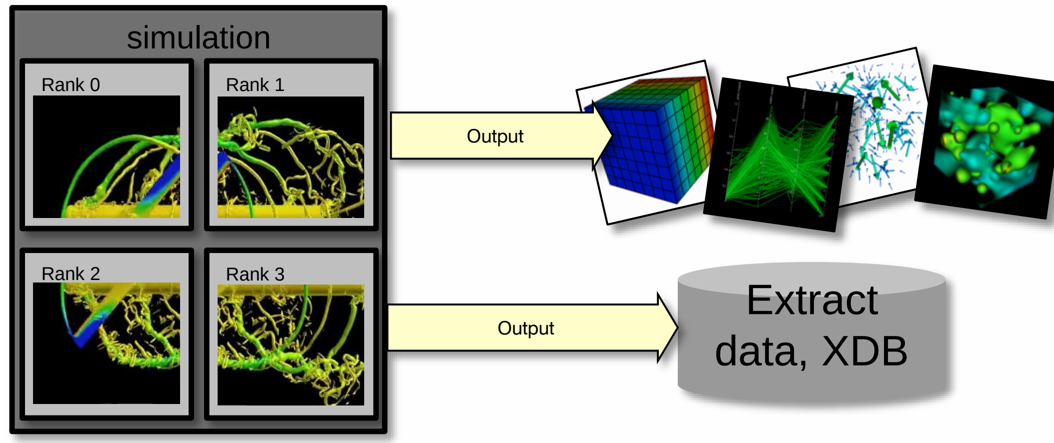
Libsim

- Enables simulations to perform data analysis and visualization in situ by applying VisIt algorithms to data.

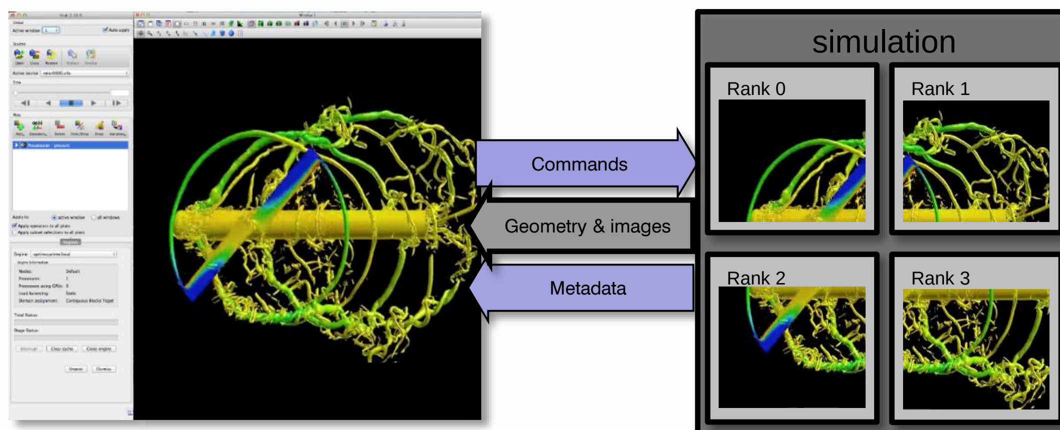


Libsim Enables Flexible Workflows

- Create automated routines to generate data in batch
  - Render images using VisIt plots and operators
  - Extract data and export



- Interactively connect via the VisIt GUI
  - Explore!
  - All functions available
  - Use custom simulation user interfaces to monitor simulation



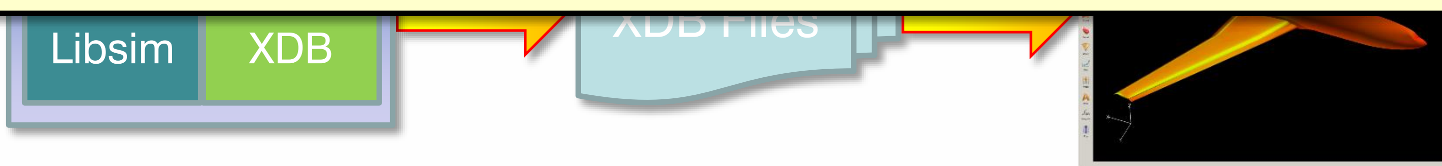
XDB Workflow

- Use Libsim to instrument simulation so it produces **FieldView XDB** files for later visualization in Fieldview
  - XDB is a CFD format made of surfaces and streamlines, which provides geometry

XDB's overcome in situ's greatest perceived weakness

– *that you need to have some idea of what you want to see in the end*

- Permits interactive exploration using post-processing methods
- Cheap enough to save frequently



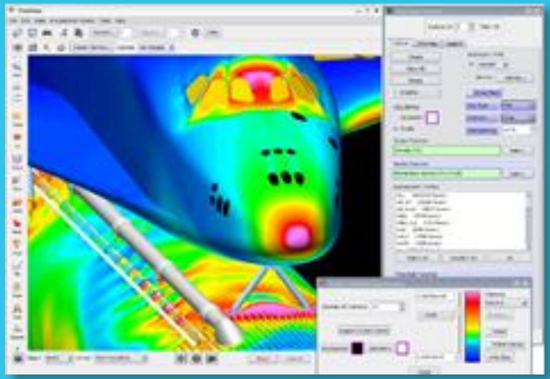
XDB produced in situ — XDB — XDB visualized

XDB Generation is Decoupled from Visualization

- XDB visualization can run on separate compute resources
- Fewer cores can be allocated
- Users can leverage their preferred visualization software

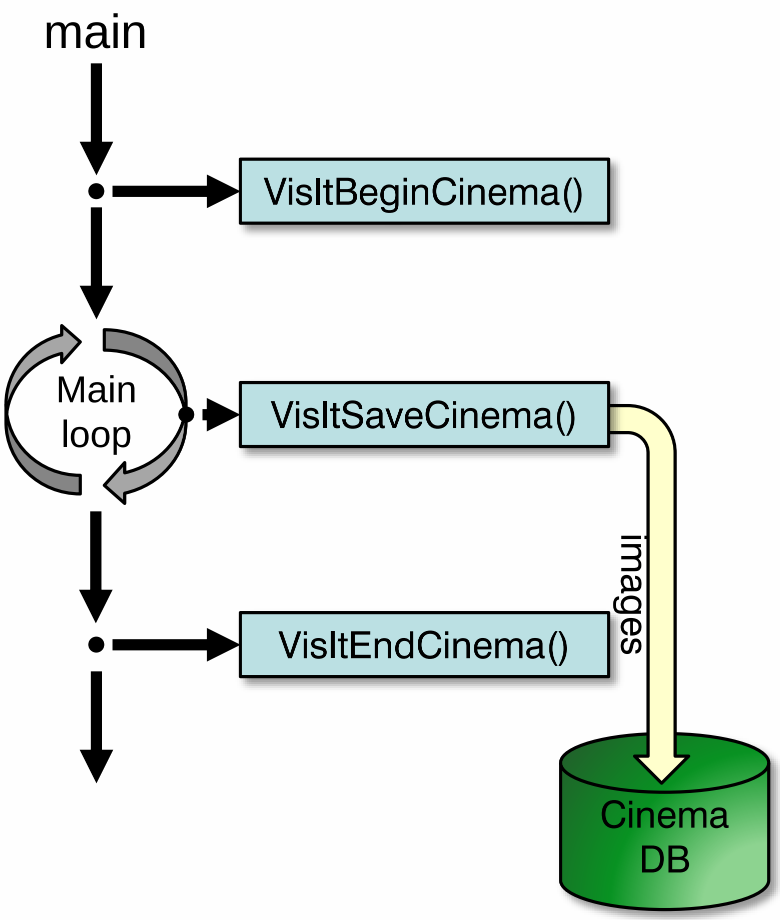
FieldView

- Sold for CFD post-processing since 1991
- Over 3000 licenses of FieldView in use today throughout the world
- Industries ranging from aerospace and automotive to nuclear engineering, turbomachinery, wind energy and food processing



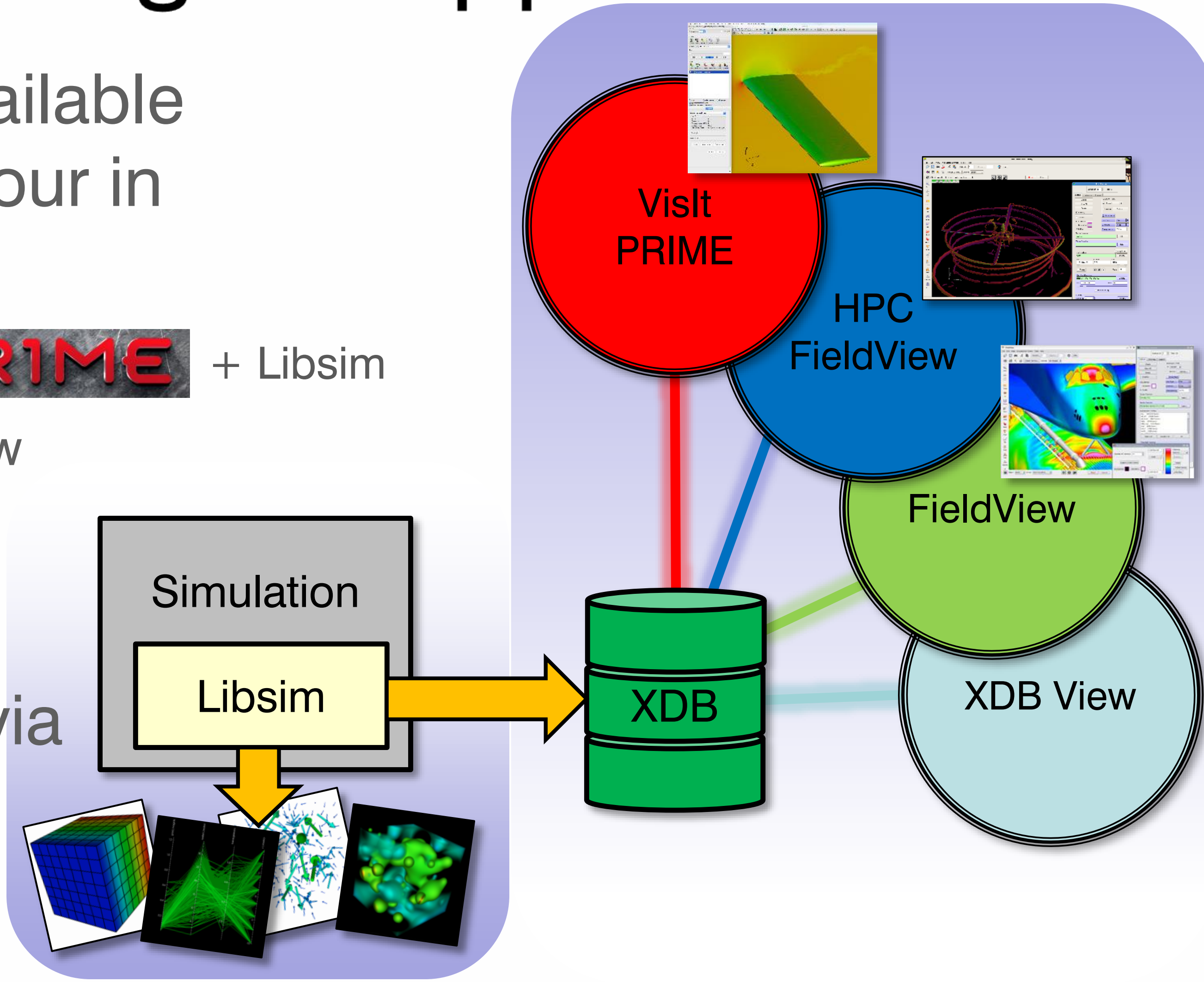
Libsim and Cinema

- Libsim extended with functions to make Cinema databases
  - Large set of plot images from various camera angles that provides a proxy for interaction/exploration
  - Sets up directory structure, saves images of VisIt plots, creates index

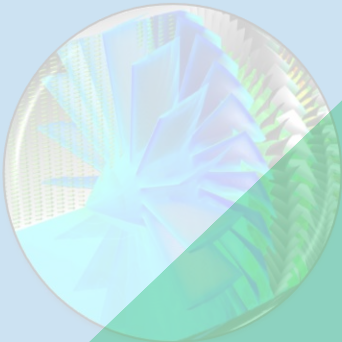


Intelligent Light Supports In Situ

- Products available to support your in situ efforts
  - VisIt PRIME + Libsim
  - HPC FieldView
  - FieldView
  - XDB View
- Connected via XDB

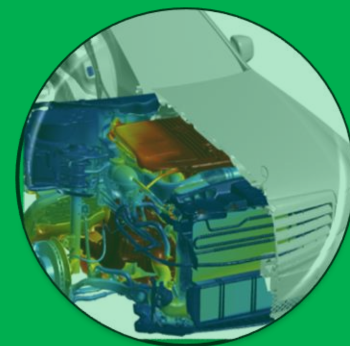


Intelligent Light Wants to Support You

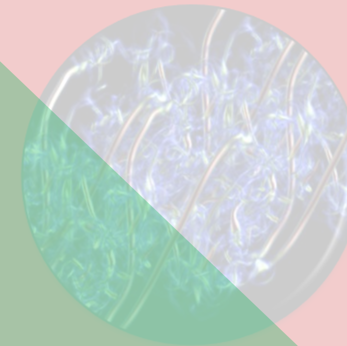


Software Products

- **FieldView** family, from



Custom Engineered Solutions

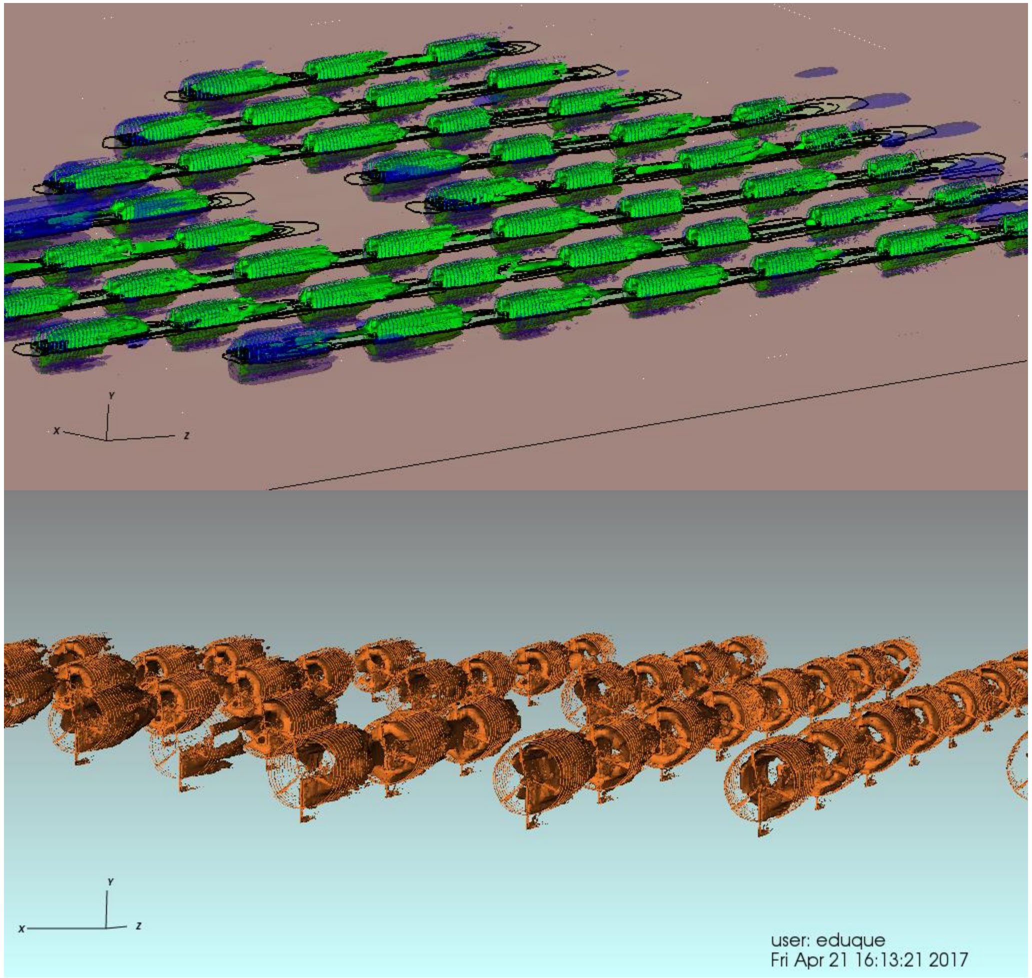


Applied Research Group (ARG)

- VisIt porting and installation
- Guidance for *do-it-yourself* solver instrumentation
- Complete solver instrumentation using Libsim
- Workflow analysis and streamlining
- Paid support and training

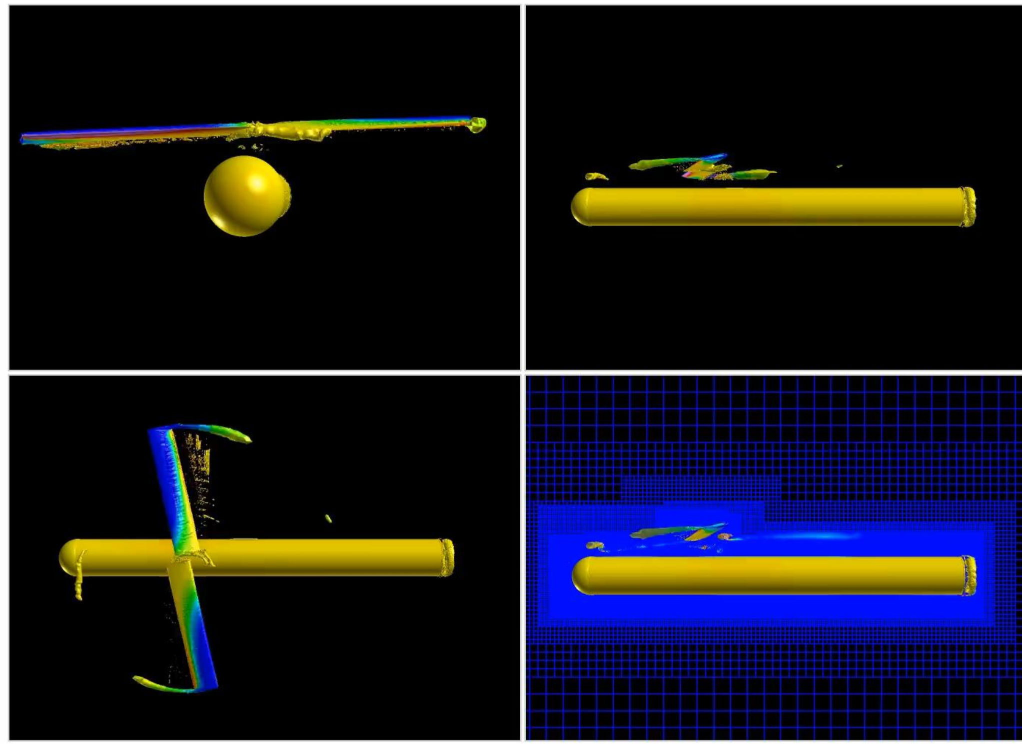
Libsim Enables Wind Farm Analysis

- Dr. Dmitry Mavriplis' group at University of Wyoming instrumented their code with Libsim
  - Simulations on Cheyenne and Yellowstone computers (~30K cores)
  - Higher order elements
  - Silo/VTK output
- Intelligent Light advised and aided in porting



CREATE-AV Kestrel

- Fixed-wing air vehicle simulation suite
- Unstructured and AMR geometries
- Extract overhead 2-3% of solver runtime to output isosurface and slice extracts to classic XDB format on 1024 cores
  - Writing volume data at same frequency would take 30% of runtime
- Extracts 21x smaller (427Mb vs 9.1Gb)



B. Whitlock, J.R. Forsythe, S. M. Legensky *"In Situ Infrastructure Enhancements for Data Extract Generation"*, AIAA SciTech, January 2016, San Diego, CA

Libsim enables in situ for many codes, including:

Nek5000

GADGET-2

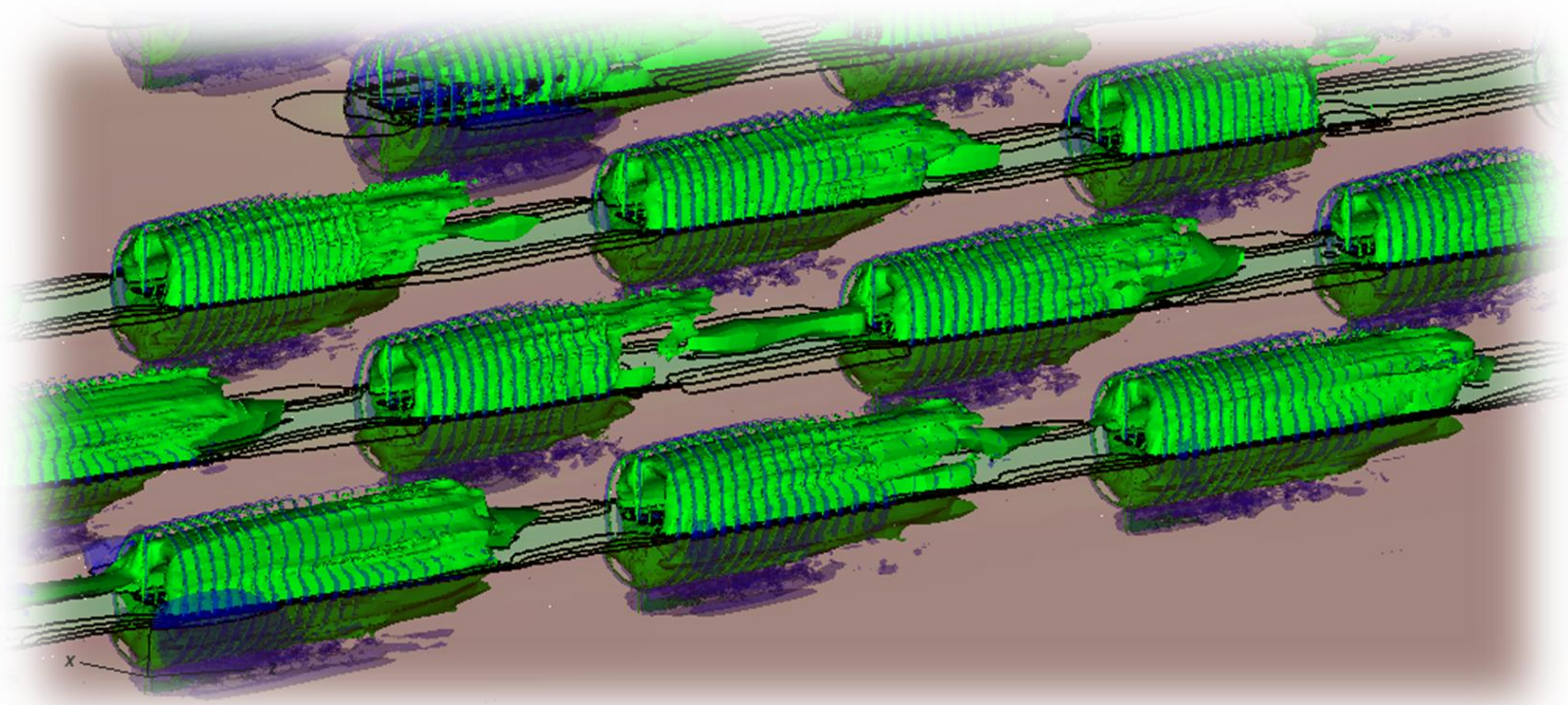
DoD CREATE-AV/Kestrel
- Libsim/XDB integration via Kestrel's CFD GUI
- To be released in standard distribution

OVERFLOW2
- Libsim/XDB integration in use at NASA Johnson
- To be released in standard distribution

Intelligent Light looks forward to helping more solvers adopt Libsim and XDB!
- JAXA has plans to use Libsim and XDB
- Intelligent Light can advise commercial CFD codes on instrumenting with Libsim and XDB

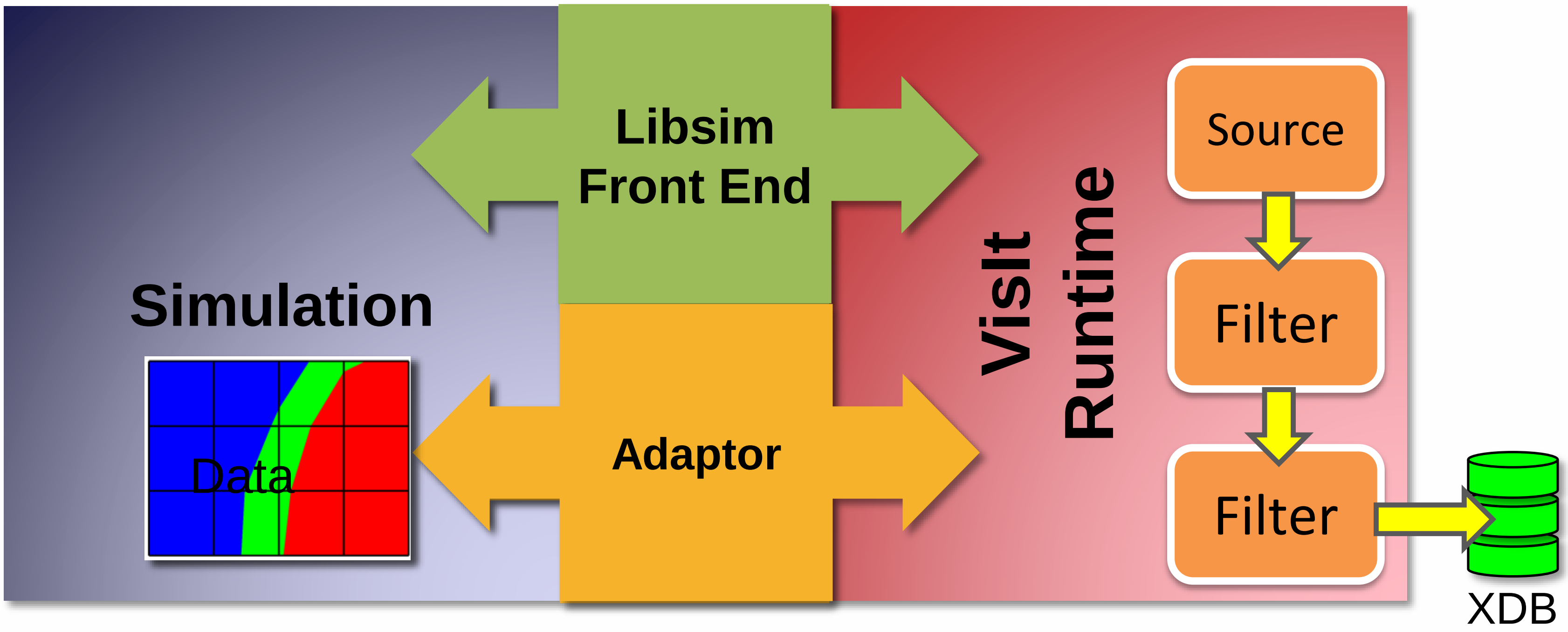
Japan Aerospace Exploration Agency

Instrumenting a Simulation



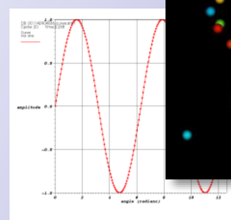
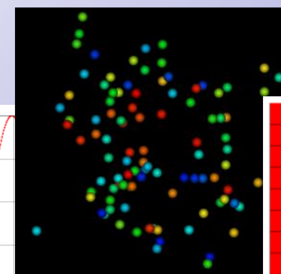
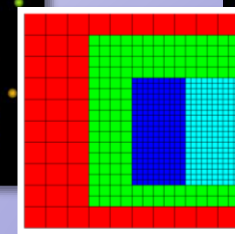
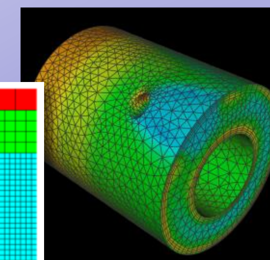
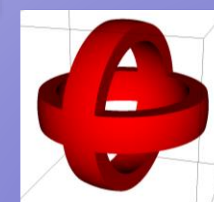
Libsim Connects Simulations and VisIt

- Expose simulation data structures to VisIt
  - Share arrays directly to avoid copies and data movement
- Enable VisIt to connect interactively to a simulation

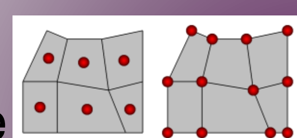


VisIt/Libsim Data Model

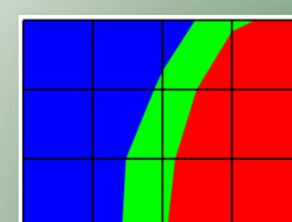
- Mesh Types
  - Structured meshes
    - Rectilinear/Curvilinear
    - I-Blanking
  - Particle meshes
  - Constructive Solid Geometry (CSG) meshes
  - Adaptive Mesh Refinement (AMR) meshes
  - Unstructured & Polyhedral meshes
    - Higher order

- Variables
  - 1 to N components
  - Zonal and Nodal
  - Enumerated type



- Materials
- Species



Libsim Programming Interface

- Control Interface
  - Handles connections and processing commands
- Data Interface
  - Handles passing data back to Libsim
- Libsim bindings exist for C, C++, Fortran, Python
- Libsim allows for a lot of flexibility
  - Interactive vs Batch (or support both)
  - Blocking vs Polling
  - A lot of common patterns can be copied from examples with little modification

Linking with Libsim

- Dynamic Linking
  - VisIt/Libsim runtime library dynamically loaded when features are used
  - C/C++
    - `LIBS=-lsimV2`
- Static Linking
  - VisIt/Libsim runtime, plugins, and 3rd party dependencies linked into simulation
  - C/C++
    - `LIBS=-lsimV2_static_par\ $(VTK_LIBS)`
- Fortran
  - Add `-lsimV2f` to LIBS
  - Includes Fortran adaptor functions for Libsim

Instrumenting a Simulation

Instrumentation can be performed **incrementally**

Step 1: Initialization

Step 2: Iteration

Step 3: Adaptor

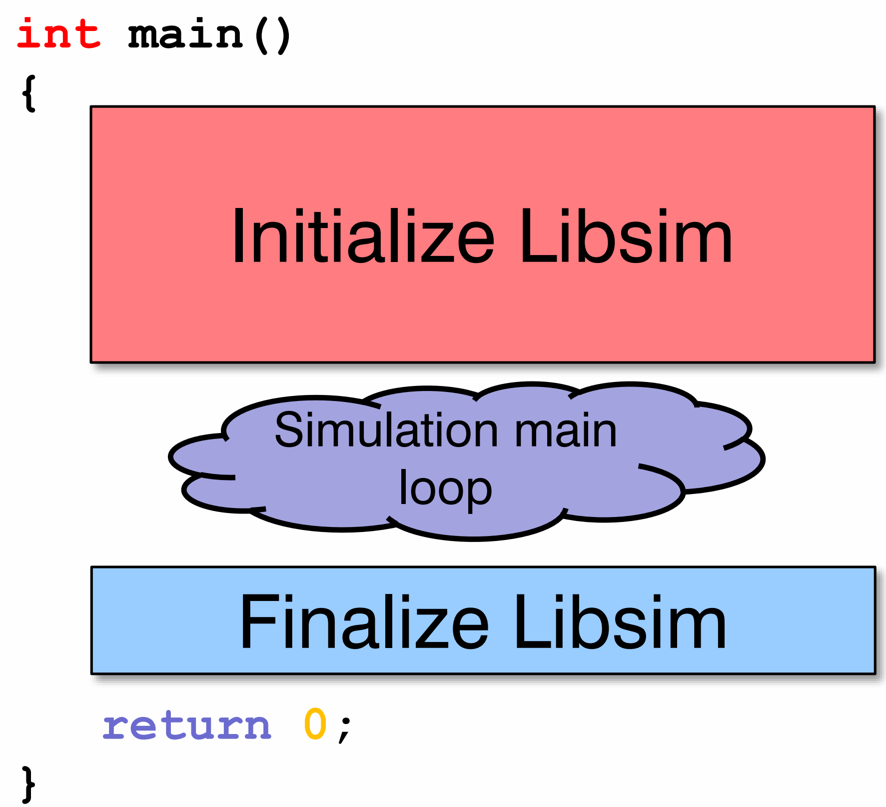
Step 4: User Interface

Create User Interface
Create adaptor
unctions that
espond to
commands from user
nterface
Send user interface
state to VisIt

Environment / Setup

Step 1

- Pass options to Libsim, such as path to VisIt
- Libsim needs to know about the environment to load the VisIt runtime library
- Initialize the runtime library
- Disconnect / shutdown when done with in situ



Set Rank, Parallel Flag, and Communicator (Parallel)

Step 1

- Libsim needs to know the rank and size of the process group
- An MPI communicator can be installed for Libsim that can be used to restrict operations to a subset of processors

```
/* Set parallel flag and rank*/
int par_rank = 0, par_size  = 1;
MPI_Comm_rank (MPI_COMM_WORLD,
               &par_rank);
MPI_Comm_size (MPI_COMM_WORLD,
               &par_size);
VisItSetParallel(par_size > 1);
VisItSetParallelRank(par_rank);

/* Tell Libsim which MPI
communicator to use. */
MPI_Comm comm;
MPI_Comm_dup(MPI_COMM_WORLD,
             &comm);
VisItSetMPICommunicator(
    (void *)&comm);
```

Batch vs Interactive

Step 1 Libsim permits multiple ways of instrumenting the main loop

Batch

- VisIt 2.9.0 extends Libsim with a batch-only support
  - Forces load of VisIt runtime library
  - Does not listen for interactive connections (simpler to implement)
  - Does not need VisIt clients to set up plots for in situ

Interactive

- The simulation must call Libsim periodically to respond to VisIt connection requests or commands
  - Opens a listen socket
  - Writes *"sim2"* file that VisIt can use to initiate a connection
  - A successful connection causes the VisIt runtime library to be loaded

Batch Initialization

Step 1

- Batch Initialization requires the VisIt runtime library to be loaded explicitly
- Once the runtime is loaded, register data adaptor functions
- Call functions to set up visualization

```
VisItInitializeRuntime();

VisItSetGetMetaData(SimGetMetaData, NULL);
VisItSetGetMesh(SimGetMesh, NULL);

VisItRestoreSessionFile(
    "/path/to/setup.session");
```

Interactive Initialization

Step 1

- Interactive initialization assumes that code for input processing will be added to the main loop
- VisIt connections are initiated by reading a *".sim2"* file created by the simulation on rank 0

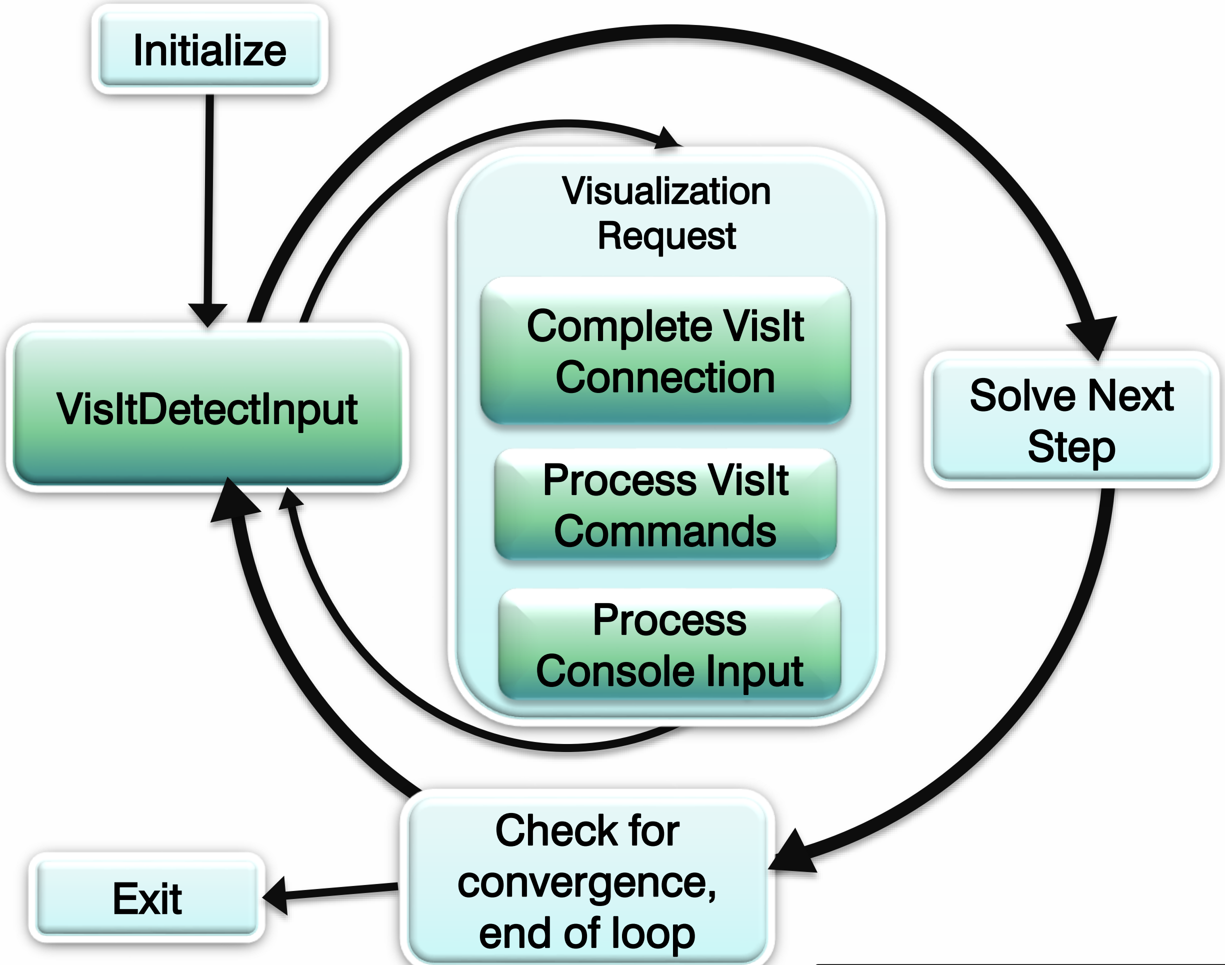
```
if(par_rank == 0)
{
    /* Write out .sim2 file that VisIt uses to connect. */
    VisItInitializeSocketAndDumpSimFile("sim_name",
        "A useful description of the simulation",
        "/path/to/where/sim/was/started",
        NULL, /* reserved */
        NULL, /* reserved */
        NULL  /* optional: pass filename for sim2 file */
    );
}
```

Skip for batch mode

Interactive Main Loop

Step 2

- Libsim opens a socket and writes out connection parameters
- Call **VisItDetectInput** to check for:
  - Connection request
  - VisIt commands
  - Console input



Skip for batch mode

Interactive Main Loop

Step 2

- Libsim opens a socket and writes out connection parameters
- Call **VisItDetectInput** to check for:
  - Connection request
  - VisIt commands
  - Console input

```
int main()
{   struct Simulation sim;
    while(! done) {
```

Computations

```
       insitu(&sim);
    }
}

void insitu(Simulation *sim) {
    int err = 0, visitstate, blocking;

    /* Get input from VisIt or timeout. */
    if(sim->par_rank == 0) {
        blocking = (sim->runMode == SIM_STOPPED) ? 1 : 0;
        visitstate = VisItDetectInput(blocking, fileno(stdin));
    }
    /* Broadcast VisItDetectInput return value. */
    MPI_Bcast(&visitstate, 1, MPI_INT, 0, MPI_COMM_WORLD);
    switch(visitstate) {
    case 1: /* Complete VisIt Connection*/
        CompleteVisItConnection(sim);
        break;
    case 2: /* Process VisIt Command. */
        ProcessVisItCommand(sim);
        break;
    case 3: /* Process Console Input */
        ProcessConsoleInput(sim);
        break;
    }
}
```

You supply these functions

Skip for batch mode

www.ilight.com

Interactive Main Loop

Step 2

- Libsim opens a
  - VisIt commands
  - Console input

```
int main()
{   struct Simulation sim;
```

```
/* Connect to VisIt, load runtime */
if(VisItAttemptToCompleteConnection()
   == VISIT_OKAY)
{
VisItSetCommandCallback(
    ControlCommandCallback,
    (void*)sim);
VisItSetSlaveProcessCallback2(
    SlaveProcessCallback,
    (void*)sim);

/* Install adaptor callbacks */
VisItSetGetMetaData(
    SimGetMetaData, (void*)sim);

VisItSetGetMesh(
    SimGetMesh, (void*)sim);

VisItSetGetVariable(
    SimGetVariable, (void*)sim);
}
```

```
/* Process a command from VisIt */
VisItProcessEngineCommand();

        insitu(&sim);
    }
}

void insitu(Simulation *sim) {
    int err = 0, visitstate, bloc

    /* Get input from VisIt or ti
    if(sim->par_rank == 0) {
        blocking = (sim->runMode
        visitstate = VisItDetectI
    }
    /* Broadcast VisItDetectInput
    MPI_Bcast(&visitstate, 1, MPI
    switch(visitstate) {
    case 1: /* Complete VisIt Connection */
        CompleteVisItConnection(sim);
        break;
    case 2: /* Process VisIt Command. */
        ProcessVisItCommand(sim);
        break;
    case 3: /* Process Console Input */
        ProcessConsoleInput(sim);
        break;
    }
}
```

```
/* Optional. Handle console */
char cmd[1000];
if (sim->par_rank == 0)
{
    if(VisItReadConsole(1000, cmd)
       == VISIT_ERROR)
    {
        strcpy(cmd, "quit");
    }
}

/* Broadcast the command */
MPI_Bcast(cmd, 1000, MPI_CHAR, 0,
MPI_COMM_WORLD);

/*Process the command */
if(strcmp(cmd, "quit") == 0)
    sim->done = 1;
```

You supply these functions

Skip for batch mode

Operations During an Iteration

Step 2

- Tell VisIt that the time step changed so new metadata will be obtained
- Create or update plots with new simulation data
- Save plots to an image or export them

Tell VisIt there are new data

```
// Set up plots using a session file
VisItRestoreSession(filename);
```

Save an image

Save an XDB

Setting up Plots

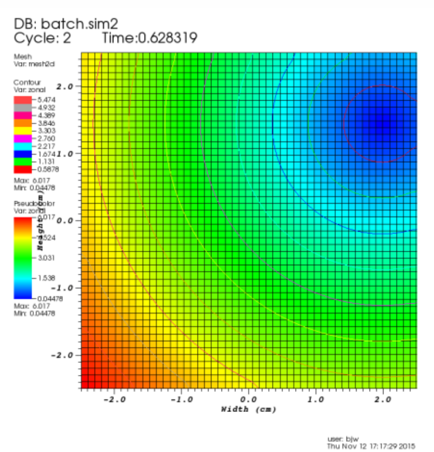
Contributed as open source

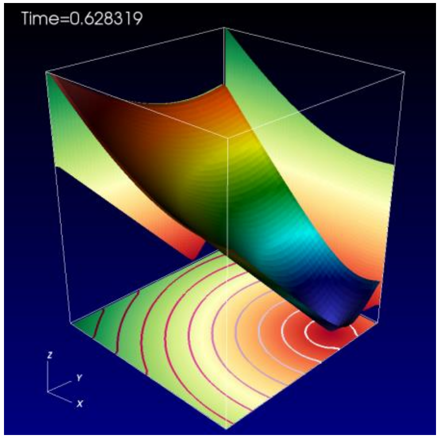
Step 2

- Libsim provides 2 ways to set up plots in situ:
  - Set up plots programmatically
  - Set up plots using VisIt session files

```
/* Set up some plots using libsim functions. */
VisItAddPlot("Mesh", "mesh2d");
VisItAddPlot("Contour", "zonal");
VisItAddPlot("Pseudocolor", "zonal");
VisItDrawPlots();
```

```
/* Set up some plots using a session file */
VisItRestoreSession("A.session");
```



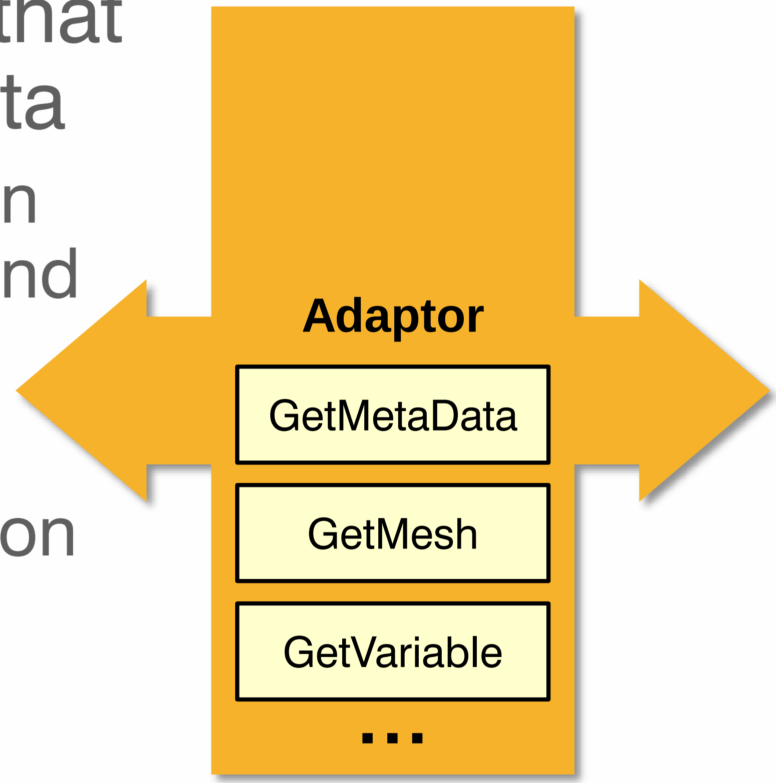


- *Dynamically create plots and set their attributes*
- *More customization is possible*

Adaptor

Step 3

- An adaptor comprises a set of functions in the simulation that VisIt calls when it needs data
  - Packages simulation's data in terms that VisIt can understand
  - Return actual pointers to simulation data (*zero copy*)
  - Return alternate representation that VisIt can free
  - Written in C, C++, Fortran, Python



Adaptor Functions

Function	Description
GetMetaData	Creates a metadata object that tells VisIt the entities advertised from the simulation
GetMesh	Returns a mesh object that contains the simulation's mesh coordinates and connectivity
GetVariable	Returns a data array object containing a simulation field
GetMaterial	Return a material object describing how the mesh can be decomposed into various materials
GetSpecies	Return a species object indicating how the mesh's materials are decomposed into various material species
GetDomainList	Return a list of domains owned by the current MPI rank

Additional adaptor functions return data for advanced features

Registering Adaptor Functions

Step 3

- Adaptor functions need to be registered with Libsim at runtime, once the VisIt runtime library has been loaded
  - Fortran adaptors rely on functions with specific names

```
visit_handle SimGetMetaData(void *cbdata)
{
    visit_handle md = VISIT_INVALID_HANDLE;
    if(VisIt_SimulationMetaData_alloc(&md)
       == VISIT_OKAY)
    {
        /* Add items here */
    }
    return md;
}
```

Simulation state /
User-defined
Data

```
VisItSetGetMetaData(SimGetMetaData, (void*)sim);
VisItSetGetMesh(SimGetMesh, (void*)sim);
VisItSetGetCurve(SimGetVariable, (void*)sim);
```

Example GetMetaData Function

Step 3

- Return the inventory of data that will be exposed to VisIt
  - Meshes
  - Scalars
  - Vectors
  - etc
- Used to populate menus, etc

```
visit_handle SimGetMetaData(void *cbdata)
{
    visit_handle md, mmd;
```

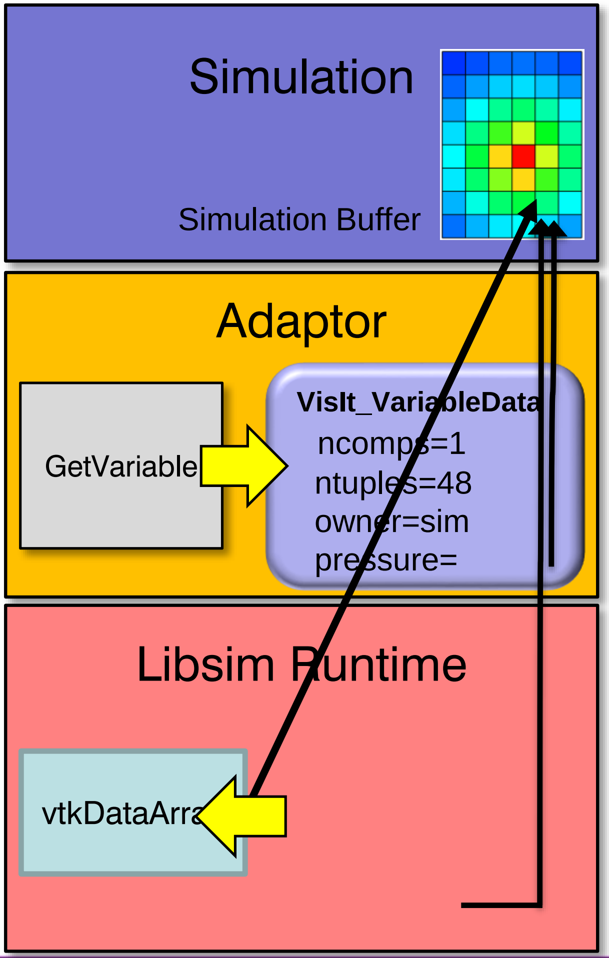
Set global simulation information into metadata

Add mesh metadata to the simulation metadata

```
    return md;
}
```

VisIt_VariableData

- Libsim describes data arrays using the VisIt_VariableData object
- VisIt_VariableData stores:
  - Pointer to the data
  - Number of Components
  - Number of Tuples
  - Owner of the data
- Libsim accepts contiguous data zero-copy
  - VisIt_VariableData_setDataX()

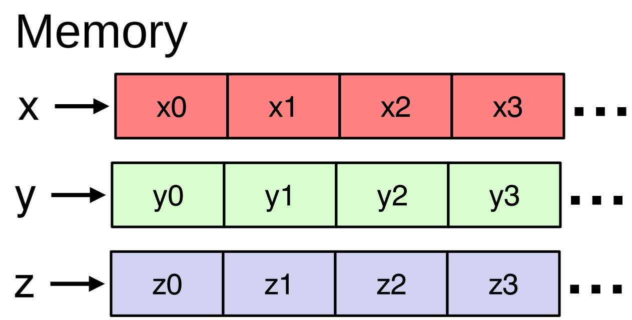


Passing Data for Structure of Arrays

Obtain XYZ coordinate values zero-copy:

```
int N = 100;
float *x = new float[N];
float *y = new float[N];
float *z = new float[N];

visit_handle coords;
int stride = sizeof(float);
VisIt_VariableData_alloc(&coords);
VisIt_VariableData_setArrayDataF(coords, 0, VISIT_OWNER_SIM, N, 0,
stride, (void *)x);
VisIt_VariableData_setArrayDataF(coords, 1, VISIT_OWNER_SIM, N, 0,
stride, (void *)y);
VisIt_VariableData_setArrayDataF(coords, 2, VISIT_OWNER_SIM, N, 0,
stride, (void *)z);
```

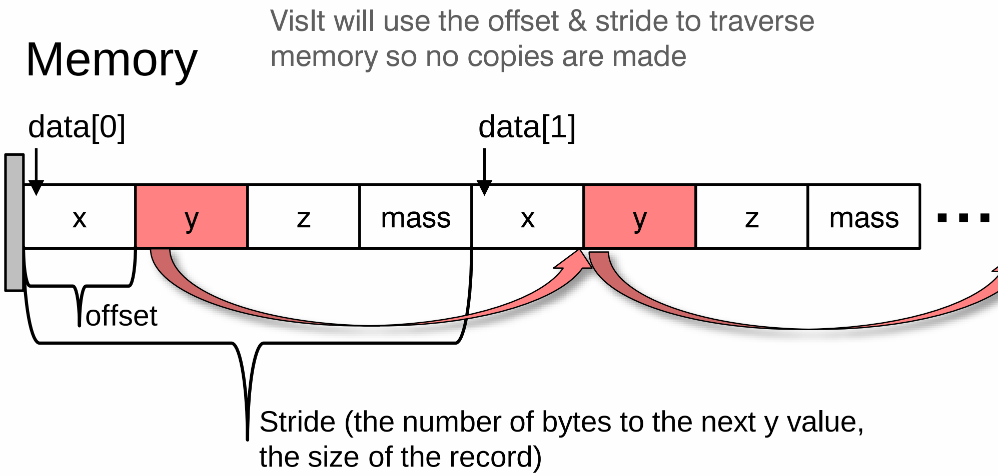


Passing Data for Array of Structures

Obtain Y values zero-copy:

```
struct Particle {
   float x,y,z,mass;
};
int N = 100;
Particle *data = new Particle[N];

visit_handle yvalues;
int offset = sizeof(float);
int stride = sizeof(Particle);
VisIt_VariableData_alloc(&yvalues);
VisIt_VariableData_setArrayDataF(yvalues, 0, VISIT_OWNER_SIM,
N, offset, stride, (void *)data);
```



- Number of tuples
- Offset: starting address of y field within Particle
- Stride: number of bytes in between Y data values
- Data: the start of the particle data
- First component

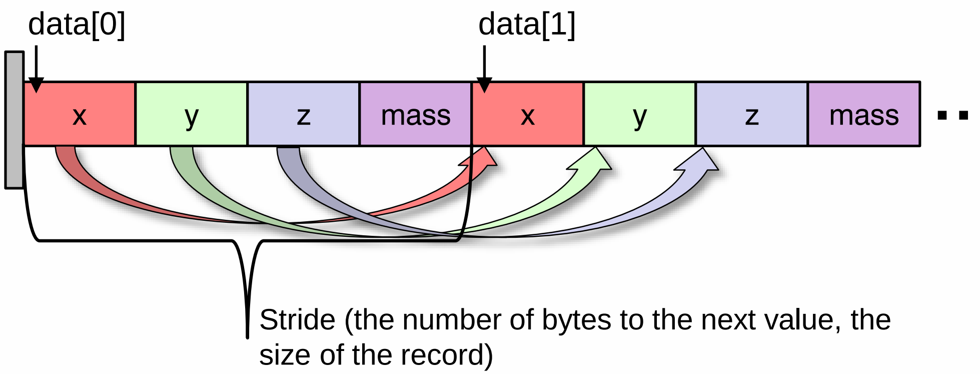
Passing Data for Array of Structures

Obtain XYZ coordinate values zero-copy:

```
struct Particle {
   float x,y,z,mass;
};
int N = 100;
Particle *data = new Particle[N];

visit_handle coords;
int offset = sizeof(float);
int stride = sizeof(Particle);
VisIt_VariableData_alloc(&coords);
VisIt_VariableData_setArrayDataF(coords, 0, VISIT_OWNER_SIM, N, 0,
stride, (void *)&data[0].x);
VisIt_VariableData_setArrayDataF(coords, 1, VISIT_OWNER_SIM, N, 0,
stride, (void *)&data[0].y);
VisIt_VariableData_setArrayDataF(coords, 2, VISIT_OWNER_SIM, N, 0,
stride, (void *)&data[0].z);
```

Memory

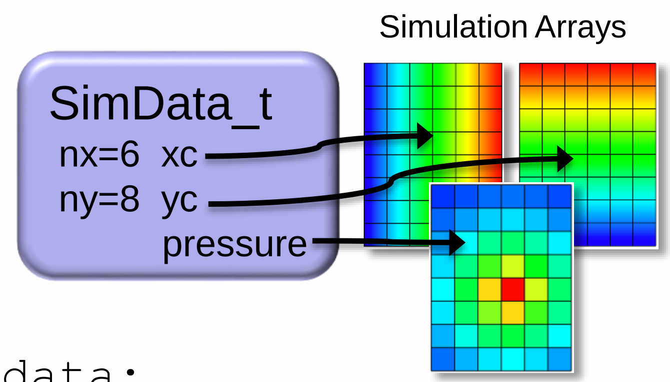


Example GetVariable Function

Step 3

```
// Example
visit_handle
GetVariable(int domain, char *name,
    void *cbdata)
{
    visit_handle h;
    SimData_t *sim = (SimData_t *)cbdata;

    return h;
}
```



Allocate VariableData object, save information about simulation array

Indicates owner of the

Indicates number of array

The array being shared

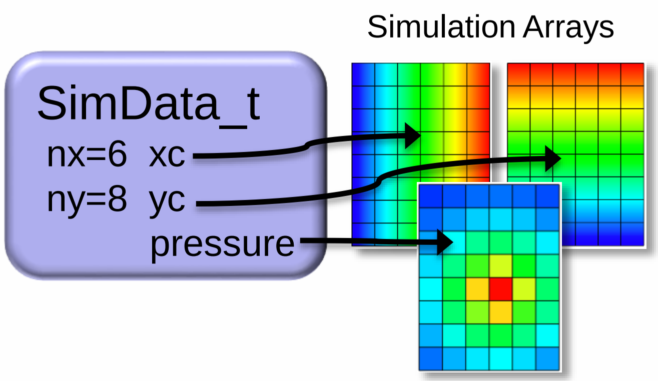
GetMesh Function

- This function is called when VisIt needs the simulation's mesh
- Returns a Libsim mesh object for the specified mesh+domain
- The mesh can be:
  - Rectilinear
  - Structured
  - Unstructured
  - AMR
  - CSG

Example GetMesh Function

Step 3

```
visit_handle
SimGetMesh(int domain, const char *name,
    void *cbdata) {
SimData_t *sim = (SimData_t*)cbdata;
visit_handle h, hxc, hyc;
int dims[2]; dims[0] = sim->nx; dims[1] = dim->ny;
```



Allocate mesh object
Allocate VariableData object for coordinates

Store coordinate array information in VariableData

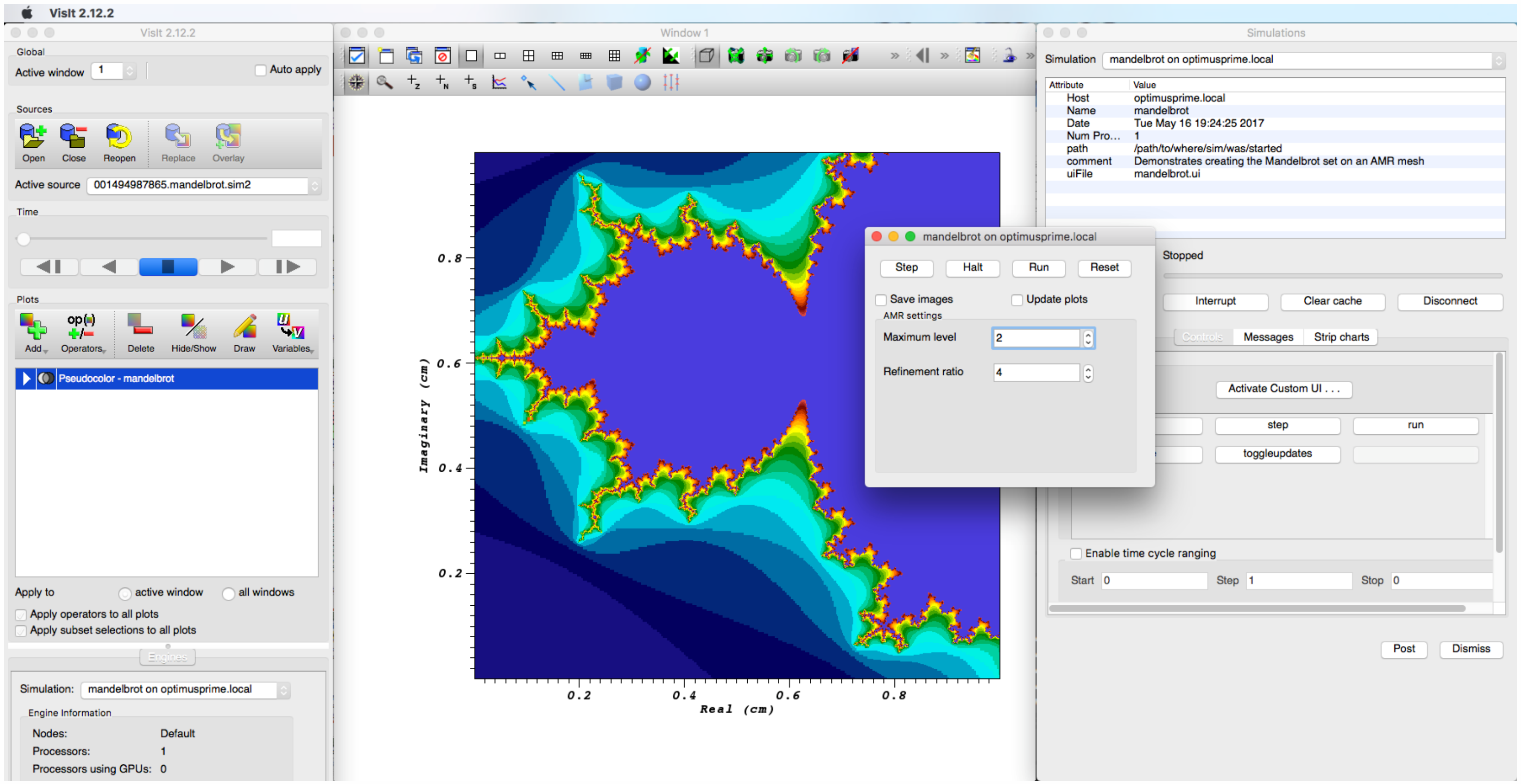
Associate coordinates with mesh

```
return h;
}
```

Simulation User Interface

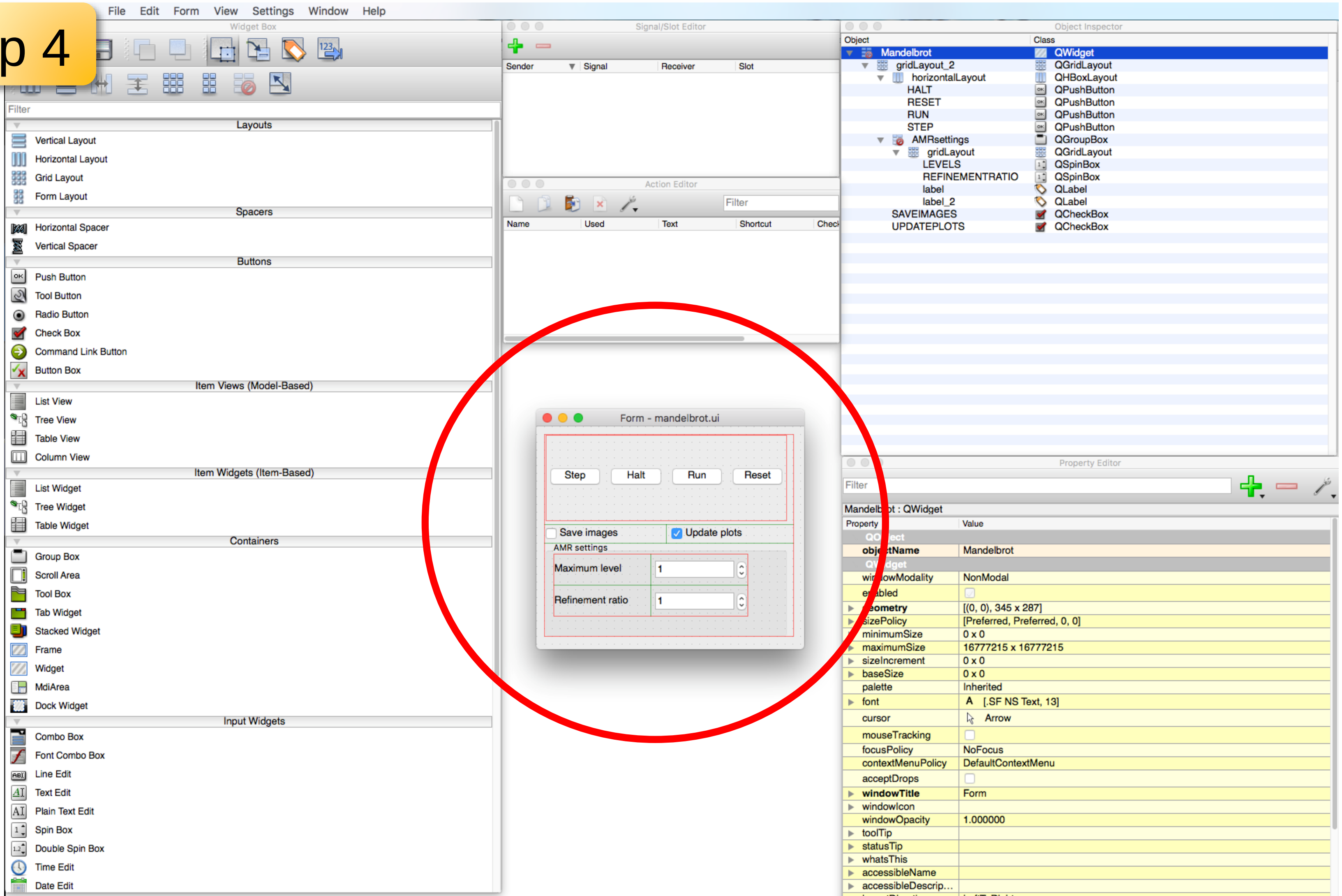
Step 4

- VisIt's GUI can create dynamic user interface for simulation



Create User Interface in Qt Designer

Step 4



Create UI Handler Functions

Step 4

- Register UI handler functions in simulation adaptor
  - Associate function with the name of the corresponding control in the UI (by name)
  - Handler function alters simulation state

```
void
ui_levels_changed(int value,
    void *cbdata)
{
    simulation_data *sim =
        (simulation_data *)cbdata;
    sim->max_levels = value;
}


/* Register a ui action */
VisItUI_valueChanged("LEVELS",
        ui_levels_changed, sim);
```

Name of control in UI window

Summary

- In Situ is necessary to handle the large amounts of data produced by simulations
  - Using less storage
  - Using less time
- Libsim is a scalable in situ infrastructure
  - Freely available
  - Get it today and generate XDBs in parallel
- In Situ extract creation provides added benefits
  - Accelerate post-processing by not operating on volume-based results
  - Leverage FieldView XDB format

Libsim Information

Information about instrumenting a simulation can be found at the following sources:

- Getting Data Into VisIt (https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf)
- VisIt Example Simulations (http://visit.ilight.com/trunk/src/tools/DataManualExamples/Simulations)
- VisIt Wiki (http://www.visitusers.org)
- VisIt Email List (visit-users@email.ornl.gov)