In-situ Visualization using VisIt

Brad Whitlock
Overview

- In-situ visualization
- History of in-situ support in VisIt
- VisIt Architecture
- How in-situ works
- Demonstration
- Adapting the simulation
- Connecting to VisIt
- Providing metadata
- Providing data
- Future work
In-situ visualization

- Adds visualization capabilities inside the simulation so it can visualize the data
- Visualize the data using the same level of resources as are being used to calculate the data (vis often gets 1/10th the resources)
- Writing plot dump files not required so plot images can be saved at greater frequency
- Visualization routines get direct access to the simulation’s memory
- Good for making canned movies
History of in-situ support in VisIt

- 2004 -- In-situ support was added by Jeremy Meredith
  - Structured meshes
  - Unstructured meshes
  - Scalar data
  - Materials
- 2005 – Whitlock added Fortran bindings
- 2009 – Whitlock added support for new data types
  - Species
  - Vector, Tensor data
  - AMR meshes
  - CSG meshes
- 2010 – Whitlock changed to function-based API
  - Does not rely on certain symbols being exposed
  - Additional error checking
  - Users don’t allocate memory
  - Single style for C and Fortran
VisIt Architecture

- VisIt consists of several cooperating programs
  - GUI
  - Viewer
  - Database server
  - Compute server

- VisIt provides *libsim* library that simulations use for in-situ visualization
- Libsim turns C, C++, Fortran simulations into a VisIt compute server that can operate directly on simulation data
- Libsim is included with VisIt binary distributions
How in-situ works

- Call functions from your simulation that let VisIt connect
- Link your simulation with libsim
- Add data access functions to your simulation that expose your arrays as data VisIt will process

Run the simulation and connect with VisIt
- Simulations write a .sim2 file that VisIt uses to connect to the simulation
- Libsim dynamically loads VisIt runtime into simulation’s address space
- You will be able to perform any of VisIt’s operations on your live simulation data
- Advance the simulation and watch your plots update
Example simulation

- Calculate Mandelbrot Set as a field on a mesh
- Mandelbrot Set is a fractal so it has detail at very fine levels
- Use Adaptive Mesh Refinement (AMR) to better resolve fine details
  - Determine which regions need more detail
  - Split regions into sub-regions and calculate
  - Recurse until desired refinement is reached

```cpp
int mandelbrot(const complex &C) {
    const int MAXIT = 30;
    complex Z; zit = 0; zit < MAXIT; ++zit)
    { Z = (Z + Z) + C;
        if(Z.magnitude() > 2.f) return zit+1;
    } return 0;
}
```
Example simulation data hierarchy

- The AMR data hierarchy is a tree where parts of the initial image are refined into smaller patches.
- Each patch is then further refined into more refined patches.
- Each AMR patch is represented as a different domain in VisIt.
  - A domain is a mesh that makes up part of a larger mesh.
  - VisIt parallelizes over domains.
Demonstration of mandelbrot example
Adapting the simulation

- Call VisIt initialization functions
- Change simulation main loop to accept inbound connections and input from VisIt
- Register a function to provide metadata to VisIt
- Register a function to provide mesh data to VisIt
- Register a function to provide any variables for VisIt
- Add mechanism to handle commands
## Control functions

<table>
<thead>
<tr>
<th>Libsim functions</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisItSetDirectory</td>
<td>Set path to VisIt</td>
</tr>
<tr>
<td>VisItSetOptions</td>
<td>Add options used when VisIt is called to determine environment</td>
</tr>
<tr>
<td>VisItSetupEnvironment</td>
<td>Determines runtime environment for latest VisIt</td>
</tr>
<tr>
<td>VisItInitializeSocketAndDumpSimFile</td>
<td>Create a socket that will be used to detect VisIt connections. Write out .sim2 file that VisIt will use to connect to simulation.</td>
</tr>
<tr>
<td>VisItDetectInput</td>
<td>Listen for VisIt connections and console input</td>
</tr>
<tr>
<td>VisItAttemptToCompleteConnection</td>
<td>Connect VisIt and load runtime library</td>
</tr>
<tr>
<td>VisItProcessEngineCommand</td>
<td>Process commands from VisIt that tell the runtime library to execute plots. This indirectly causes your plugin’s Data Access Code to be called.</td>
</tr>
</tbody>
</table>
Initialization functions

- Call initialization functions and enter the application’s main loop where it iterates

```c
int main(int argc, char **argv)
{
    VisItOpenTraceFile("amr_trace.txt");

    /* Initialize environment variables. */
    SimulationArguments(argc, argv);
    VisitSetupEnvironment();

    /* Write out .sim2 file that VisIt uses to connect. */
    VisitInitializeSocketAndDumpSimFile("mandelbrot",
                                        "Demonstrates creating an AMR mesh of mandelbrot set",
                                        "/path/to/where/sim/was/started",
                                        NULL, NULL, NULL);

    /* Read input problem setup, geometry, data. */
    read_input_deck();

    /* Call the main loop. */
    mainloop();

    VisitCloseTraceFile();

    return 0;
}
```
Sample Event Loop

• Call VisitDetectInput from inside your main loop
  • 1 -> Call VisitAttemptToCompleteConnection
  • 2 -> Call VisitProcessEngineCommand
  • Otherwise do simulation work

• This example blocks on input from VisIt if the simulation is halted and polls when the simulation is running

```c
void mainloop(simulation_data *sim)
{
    int blocking, visitstate, err = 0;
    /* If we’re not running by default then simulate once there’s */
    /* once VisIt connects. */
    if(sim->runMode == SIM_STOPPED)
        simulate_one_timestep(sim);

    /* main loop */
    do
    {
        blocking = (sim->runMode == SIM_RUNNING) ? 0 : 1;
        /* Get input from VisIt or timeout so the simulation can run */
        if(visitstate == VISIT_OKAY)
            visitstate = VisitDetectInput(blocking, -1);
        /* Do different things depending on the output from VisIt */
        else if(visitstate == 1)
            /* VisIt is trying to connect to sim. */
            if(VisitAttemptToCompleteConnection() == VISIT_OKAY)
                /* sim->runMode = SIM_STOPPED; */
                VisitSetCommandCallback(ControlCommandCallback, (void*)sim);
                VisitSetSetMetaData(SimGetSetMetaData, (void*)sim);
                VisitSetSetMesh(SimGetSetMesh, (void*)sim);
                VisitSetSetVariable(SimGetSetVariable, (void*)sim);
                VisitSetSetDomainNesting(SimGetSetDomainNesting, (void*)sim);
            else
                fprintf(stderr, "VisIt did not connect\n");
        else if(visitstate == 2)
            /* VisIt wants to tell the engine something. */
            if(VisitProcessEngineCommand() == VISIT_OKAY)
                /* * Disconnect on an error or closed connection. */
                if(VistDisconnect() && VisitClose()
                    /* Start running again if VisIt closes. */
                    sim->runMode = SIM_RUNNING;
                }
        /* There was no input from VisIt, return control to sim */
        simulate_one_timestep(sim);
    } while(!sim->done && err == 0);
}
```
Data Access Code

- Data Access Code is code that you write for your simulation that provides metadata about your meshes and variables as well as the problem-sized data.
- Data Access Code are callback functions that VisIt calls when metadata and data are needed.
  - Write your functions in C, C++, Fortran (*currently supported bindings*).
- Data Access Code callback functions let you allocate data objects that tell VisIt about your arrays.
  - Pass arrays directly (data layout must match VTK).
  - Allocate new arrays that VisIt will use and free.
- VisIt’s runtime calls your simulations as needed, depending on the pipeline being executed so you don’t have to convert data you won’t need.
Providing metadata

- You must implement a GetMetaData callback function so your simulation can tell VisIt about the meshes and variables you wish to expose without actually having to provide them
  - Important because you don’t know which meshes and fields VisIt will ask for because that is determined by how the user drives the tool
  - Called when VisItTimestepChanged() is called when the simulation advances to a new time step
- Metadata helps populate VisIt’s menus
- Create a SimulationMetaData object
  - Add MeshMetaData objects
  - Add VariableMetaData objects
Simulation metadata

• Register the callback using the VisItSetGetMetaData function

```c
visit_handle
SimGetMetaData(void *cbdata)
{
    visit_handle md = VISIT_INVALID_HANDLE;
    simulation_data *sim = (simulation_data *)cbdata;

    /* Create metadata. */
    if (VisIt_SimulationMetaData_alloc(&md) == VISIT_OKAY)
    {
        visit_handle mm = VISIT_INVALID_HANDLE;
        visit_handle vm = VISIT_INVALID_HANDLE;
        visit_handle cm = VISIT_INVALID_HANDLE;

        /* Set the simulation state. */
        VisIt_SimulationMetaData_setMode(md, (sim->runMode == SIM_STOPPED) ?
            VISIT_SIMMODE_STOPPED : VISIT_SIMMODE_RUNNING);
        VisIt_SimulationMetaData_setCycleTime(md, sim->cycle, sim->time);

        return md;
    }
}
```
Mesh metadata

• Add a new MeshMetaData object for each mesh that you want to expose to VisIt

```c
/* Fill in the AMR metadata. */
visit_handle mmd = VISIT_INVLD_HANDLE;
if (VISIT_MeshMetaData Alloc(&mmd) == VISIT_OKAY)
{
    /* Set the mesh's properties */
    VISIT_MeshMetaData_setName(mmd, "AMR mesh");
    VISIT_MeshMetaData_setMeshType(mmd, VISIT_MESH_TYPE_AMR);
    VISIT_MeshMetaData_setTopologicalDimension(mmd, 2);
    VISIT_MeshMetaData_setSpatialDimension(mmd, 2);

    int ndoms = patch_num_patches(Gsim->patch);
    VISIT_MeshMetaData_setNumDomains(mmd, ndoms);
    VISIT_MeshMetaData_setDomainTitle(mmd, "patches");
    VISIT_MeshMetaData_setDomainPieceName(mmd, "patch");

    int nlevels = patch_num_levels(Gsim->patch);
    VISIT_MeshMetaData_setNumGroups(mmd, nlevels);
    VISIT_MeshMetaData_setGroupTitle(mmd, "levels");
    VISIT_MeshMetaData_setGroupPieceName(mmd, "level");
    patch_t **patches = patch flat array(Gsim->patch);
    int *pcount = (int *)malloc(nlevels * sizeof(int));
    memset(pcount, 0, nlevels * sizeof(int));
    for (int i = 0; i < ndoms; ++i)
    {
        char tmpName[100];
        sprintf(tmpName, "level%d.patch304d", patches[i]->level, pcount[patches[i]->level]++);
        VISIT_MeshMetaData_addDomainName(mmd, tmpName);
        VISIT_MeshMetaData_addGroup(mmd, patches[i]->level);
    }
    FREE(pcount);
    FREE(patches);

    VISIT_MeshMetaData_setUnits(mmd, "cm");
    VISIT_MeshMetaData_setUnits(mmd, "cm");
    VISIT_MeshMetaData_setLabel(mmd, "Real");
    VISIT_MeshMetaData_setLabel(mmd, "Imaginary");

    VISIT_SimulationMetaData_addMesh(mmd, mmd);
}
```
Variable metadata

• Add a new `VariableMetaData` object for each variable that you want to expose to VisIt

```c
/* Add a variable. */
visit_handle vmd = VISIT_INVALID_HANDLE;
if(Visit_VariableMetaData_alloc(&vmd) == VISIT_OKAY)
{
    Visit_VariableMetaData_setName(vmd, "mandelbrot");
    Visit_VariableMetaData_setMeshName(vmd, "AMR_mesh");
    Visit_VariableMetaData_setType(vmd, VISIT_VARTYPE_SCALAR);
    Visit_VariableMetaData_setCentering(vmd, VISIT_VARCENTERING_ZONE);
    Visit_SimulationMetaData_addVariable(md, vmd);
}
```
Providing a mesh

- You must provide a GetMesh callback function so your simulation can provide mesh data to VisIt.
- GetMesh callback returns $i^{th}$ domain.
- Allocate a mesh object and set its properties:
  - RectilinearMesh
  - CurvilinearMesh
  - UnstructuredMesh
  - CSGMesh
- Mesh objects ferry data to VisIt and then on to VTK objects.

```c
visit_handle
SimGetMesh(int domain, const char *name, void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;
    /* Get the patch with the appropriate domain id. */
    simulation_data *sim = (simulation_data *)cbdata;
    patch_t *patch = patch_get_patch(sim->patch, domain);
    if(strcmp(name, "EMK_mesh") == 0 & patch != NULL)
    {
        int i;
        if(VisIt_RectilinearMesh_alloc(&h) != VISIT_ERROR)
        {
            /* Initialize X, Y coords. */
            float *coordX = patch_get_coords(patch);
            float *coordY = patch_get_coords(patch);

            /* Give the mesh some coordinates it can use. */
            visit_handle xc, yc;
            VisIt_VariableData_alloc(&xc);
            VisIt_VariableData_alloc(&yc);
            VisIt_VariableData_setData(xc, VisIt_OWNER_VISIT, 1, patch->vox+1, coordX);
            VisIt_VariableData_setData(yc, VisIt_OWNER_VISIT, 1, patch->vox+1, coordY);
            VisIt_RectilinearMesh_setCoordXY(xc, xc, yc);
        }
    }
    return h;
}
```
Providing data

- You can provide a GetVariable callback function if your simulation needs to provide data on the mesh
- Create a VariableData object
  - Simple wrapper that describes the type and contents of the user-supplied data array
  - Array can be simulation-owned or VisIt-owned

```c
void SimGetVariable(int domain, const char *name, void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;

    /* Get the patch with the appropriate domain id. */
    simulation_data *sim = (simulation_data *)cbdata;
    patch_t *patch = patch_get_patch(&sim->patch, domain);

    if(strcmp(name, "mandelbrot") == 0 && patch != NULL)
    {
        VisIt_VariableData_alloc(&h);
        VisIt_VariableData_setDataC(h, VISIT_OWNER_SIM, 1,
                                  patch->nx * patch->ny, (char *)patch->data);
    }

    return h;
}
```
Updating plots

- Call `VisItTimeStepChanged` when you want to tell VisIt that the simulation time step has changed
  - This causes your `GetMetaData` callback to be called and new metadata to be sent to VisIt
- Call `VisItUpdatePlots` when you want to update VisIt’s plots
  - An update message gets sent to VisIt asking it to update its plots
  - The update message causes all of the plots to be reexecuted
  - This causes new data to be requested from the simulation via the data access code callbacks
Parallel considerations

- Libsim can instrument parallel simulations too
  - See updateplots.c example in VisIt source code
- More setup in your main function after `VisItSetupEnvironment`
  - Call `VisItSetParallel, VisItSetParallelRank`
  - Provide broadcast callbacks that let VisIt use your MPI implementation to broadcast integers and strings
    - `VisItSetBroadcastIntFunction`
    - `VisItSetBroadcastStringFunction`
- Only rank 0 can call `VisItInitializeSocketAndDumpSimFile`
- `VisItDetectInput` must be called only on rank 0 so its results must be broadcast to other processors to coordinate them
- When rank 0 calls `VisItProcessEngineCommand`, it needs to broadcast the results to the other processors so they can call it too
Domain list

- Provide a GetDomainList callback function that tells VisIt how domains are distributed among processors
  - In the most common case where each processor has 1 domain, the domain list contains a single integer: the processor’s MPI rank
  - VisIt’s load balancer uses the domain list to ensure that when VisIt asks for domain n, processor n gets the request for data
Exporting data back into the simulation

- VisIt can export processed data to several file formats.
- VisIt can also export processed data back to simulations that implement write callback functions in their data access code.
- Use VisIt to process simulation data and read the results back into the simulation for additional calculations:
  - Data could come from the simulation.
  - Data could come from files, etc.
Future work

- Complete our SimV2 Windows port
- Make plot setup from inside the simulation more convenient
  - Libsim provides VisItExecuteCommand to pass Python code strings to VisIt to let the simulation set up plots
  - Provide functions for creating plots and setting their attributes
  - Provide functions for adding operators
- Add Query API
- Complete work that lets simulations provide custom UI for VisIt’s Simulation window
Resources

- **Getting Data Into VisIt Manual**
  - [https://wci.llnl.gov/codes/visit/manuals.html](https://wci.llnl.gov/codes/visit/manuals.html)

- [http://www.visitusers.org](http://www.visitusers.org)
- VisIt User’s List
- VisIt Developer’s List