Runtime Visualization of the Human Arterial Tree

Joseph A. Insley*, Michael E. Papka*, Suchuan Dong†, George Karniadakis†, and Nicholas T. Karonis‡

*Computation Institute
Argonne National Laboratory
The University of Chicago

†Division of Applied Mathematics,
Brown University

‡Department of Computer Science
Northern Illinois University

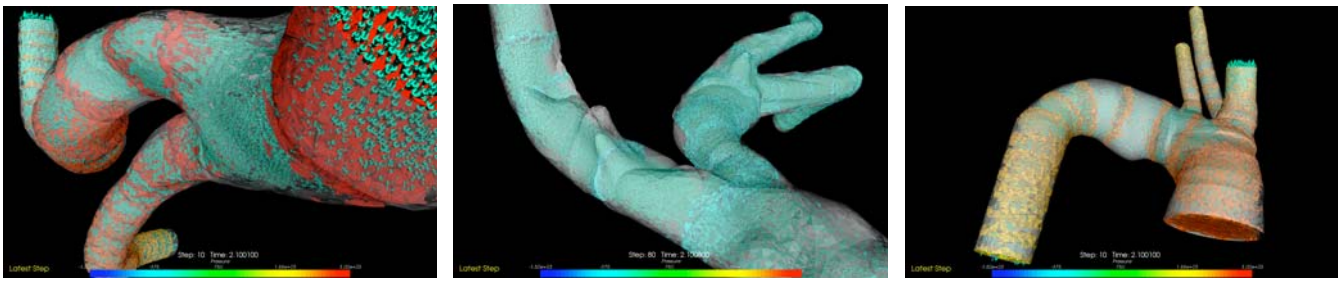

Figure 1: Three instances of the High Resolution Client, each displaying isosurfaces of the blood flow pressure and vectors of the velocity within a different 3D bifurcation mesh of the human arterial tree model.

Abstract

Large-scale simulation codes typically execute for extended periods of time, often on distributed computational resources. Because these simulations can run for hours, or even days, scientists would like to get feedback about the state of the computation and the validity of its results as it continues to run. It is also important that these capabilities be made available with little impact on the performance and stability of the simulation. Visualizing and exploring data in the early stages of the simulation can help scientists identify problems early, potentially avoiding a situation where a simulation runs for several days, only to discover an error with an input parameter caused both time and resources to be wasted.

We describe an application that aids in the monitoring and analysis of a simulation of the human arterial tree. The application provides researchers with high-level feedback about the state of the on-going simulation and enables them to investigate particular areas of interest in greater detail. The application also offers monitoring information about the amount of data produced and data transfer performance between the various components of the application.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems - Distributed/network graphics; I.6.6 [Simulation and Modelling]: Simulation Output Analysis;

Keywords: Real-time visualization, flow visualization

1 Introduction

Simulation-driven science is increasingly being used to initiate scientific discovery. As the accuracy and complexity of scientific simulations continue to rise, the computational resources required to execute these simulations also increase. Even with the advancement of Grid-enabling technologies [1, 2] allowing scientists to simultaneously utilize multiple distributed resources, large-scale simulation codes often run for days at a time. As the Department of Energy Office of Science Data-Management Challenge report points out "Long-running simulations can become vastly more productive if some information can be visualized in real time, allowing decisions to abort or steer the simulation. [3]" Enabling researchers both to monitor the progress of the running simulation and to validate its results can enable these types of decisions to be made, thereby saving valuable time and resources.

Equally important to providing these visualization capabilities is doing so with little or no impact on the performance and stability of the running simulation. Failures in the visualization, as a result of software or hardware malfunction for example, should not cause the simulation to fail as well. The ability to stop and start the visualization at arbitrary points while the simulation continues to run is also valuable.

We present here an application developed to provide visualization support for the Human Arterial Tree Simulation Project [4]. A distributed visualization pipeline was designed and developed that both archives data produced by these simulations and is capable of visualizing it in near-real time, with little impact on the performance and stability of the running simulation. Section 2 discusses related work performed in this area. Section 3 provides information about the human arterial tree simulation. Sections 4-8 give an overview of the visualization application and describe the various components, including the Data Archiver, the Low Resolution Client, and the High Resolution Client. Section 9 describes additional capabilities of the application. The paper concludes with a discussion of proposed future work.

2 Related Work

Particle Flurries, described in [5], is an example of synoptic 3D flow visualization, which gives viewers a synopsis of all flow features simultaneously. Motion-blurred linestrips surrounded by a black halo are used to represent particles as they follow pathlines from the inflow to the outflow of an artery. While this

*email: {insley,papka}@ci.uchicago.edu

method is well suited to the data produced in the human arterial tree simulation, it requires the precomputing of particle paths.

Forsberg et al. [6], have also developed an application that uses isosurfaces to help expose the gross features within the flow region. This application also follows a policy that a common data format is maintained between the simulation and the visualization software. This eliminates the burden of transforming the data between the two components.

Uintah [7] is a problem solving environment that enables the visualization and computational steering of complex simulations. It uses common component architecture to integrate the various components into an integrated environment and manage communication between them. It uses Nexus (now known as Globus XIO) for wide-area communication and MPI between colocated components.

3 THE HUMAN ARTERIAL TREE SIMULATION

Motivated by a grand-challenge problem in biomechanics, we are striving to simulate blood flow in the *entire* human arterial tree. The problem originates from the widely accepted causal relationship between blood flow and the formation of arterial disease such as atherosclerotic plaques. These disease conditions preferentially develop in separated and recirculating flow regions such as arterial branches and bifurcations. Modeling these types of interactions requires significant compute resources to calculate the three-dimensional unsteady fluid dynamics in the sites of interest. Waveform coupling between the bifurcations, however, can be reasonably modeled by a reduced set of one-dimensional equations that capture the cross-sectional area and sectional velocity properties [8]. One can therefore simulate the entire arterial tree using a hybrid approach based on a reduced set of one-dimensional equations for the overall system and detailed 3D Navier-Stokes equations at arterial branches and bifurcations.

Limited computational resources are required for the 1D model; therefore it can run on a single compute node. In order to capture the flow dynamics in an artery bifurcation reasonably well, however, the grid resolution typically requires a mesh of 70,000 to 200,000 finite elements of high order; here spectral elements with a spectral polynomial order of 10 to 12 on each element are used [9]. These 3D models clearly require considerably more compute power to calculate.

The human arterial tree model used here contains the largest 55 arteries in the human body with 27 artery bifurcations. Because many of the bifurcations are close together, some of the 3D meshes contain multiple bifurcations. The primary compute environment for this simulation is the National Science Foundation's TeraGrid [10]. Several of the largest runs took place during demonstrations at SC|05. In order to reduce the simulation times during these demonstrations, the spectral elements were calculated with a polynomial order of 7, rather than 10 to 12. Depending on the complexity of the particular mesh, anywhere from 64 to 128 processors were used for each 3D mesh. These arterial tree model simulations utilized compute resources at four TeraGrid resource provider sites: National Center for Supercomputing Applications (NCSA), San Diego Supercomputer Center (SDSC), Pittsburgh Supercomputing Center (PCS), and Texas Advanced Computing Center (TACC), along with a site in the UK, Computer Services for Academic Research (CSAR). This Compute power was coupled with visualization resources at The University of Chicago/Argonne National Laboratory TeraGrid resource provider site.

The runs consisted of 17 bifurcations contained in 11 3D meshes distributed across the five-compute sites. The multi-job submission mechanism enabled with Globus and MPICH-G2 was used to submit the calculation of each of the 3D bifurcation meshes as a separate subjob of a single MPI application, to be executed across the five different resources. MPICH-G2 takes care of the details of submitting the subjobs to the separate resources and establishing the communication paths between them. Once all of the processes are running, they are part of the same MPI_Communicator and can exchange information via standard message passing mechanisms.

Several types of data are associated with the simulations of this arterial tree model. The input data comprises the three dimensional unstructured mesh that defines the arterial bifurcations. This data does not change over the course of the simulation. The output data is of three types. First is the data from the 1D simulation. This consists of 55 float values for each time step of the simulation, which represent the pressure at the inlet of each of the 55 arteries. Because this is a small amount of data, it is output at each time step of the simulation. The output data of the 3D simulations comprises one scalar value, representing the pressure, and three values, for the vector representing the blood flow velocity, for each point on the unstructured mesh. Since this can be a large amount of data, and because these values change relatively little from one time step to the next, it is not practical to output these values for every time step, as this would negatively impact the simulation. Instead they are output at regular intervals, typically every fifth time step. During the demonstrations at SC|05 this data was reduced to a polynomial order of 3 before being output. While performing this reduction did have some impact on the performance of the simulation, it was relatively minor. Also, the trade-off of having the ability to visualize the data in near-real time made the performance hit well worthwhile. In order to provide feedback about the simulation more frequently, another reduced set of data is output every third time step. This reduced set consists of the pressure scalars and the velocity vectors for only those points on the boundary of the mesh, typically a reduction on the order of 98%. While this is a greater reduction in the amount of data, it is less computationally intensive to perform and can therefore be done more often with less impact on the simulation.

4 VISUALIZATION OVERVIEW

The visualization application was designed with several goals in mind and builds from previous visualization work [11, 12]. The goals of the application were to:

- provide high-level feedback about the state of the on-going simulation in as close to real time as possible,
- enable users to investigate particular areas of interest in greater detail, and
- have little impact on the performance and stability of the running simulation, which is to say, any failure in the visualization should not cause the simulation to fail as well.

These three goals layout the major contributions as a visualization application, specifically:

- visual feedback of the state of a complex simulation distributed across many sites is extremely helpful,
- the use of multiple views of data is beneficial (i.e. rapid display of low-resolution representation with high-resolution version to follow), and
- the use of middleware solutions for the connection, management, and separation of the visualization application from the simulation.

The application consists of three separate, yet integrated components, as seen in Figure 2. First is the Data Archiver. This is the input to the visualization pipeline and is the only component directly tied to the running simulation. The Data Archiver receives

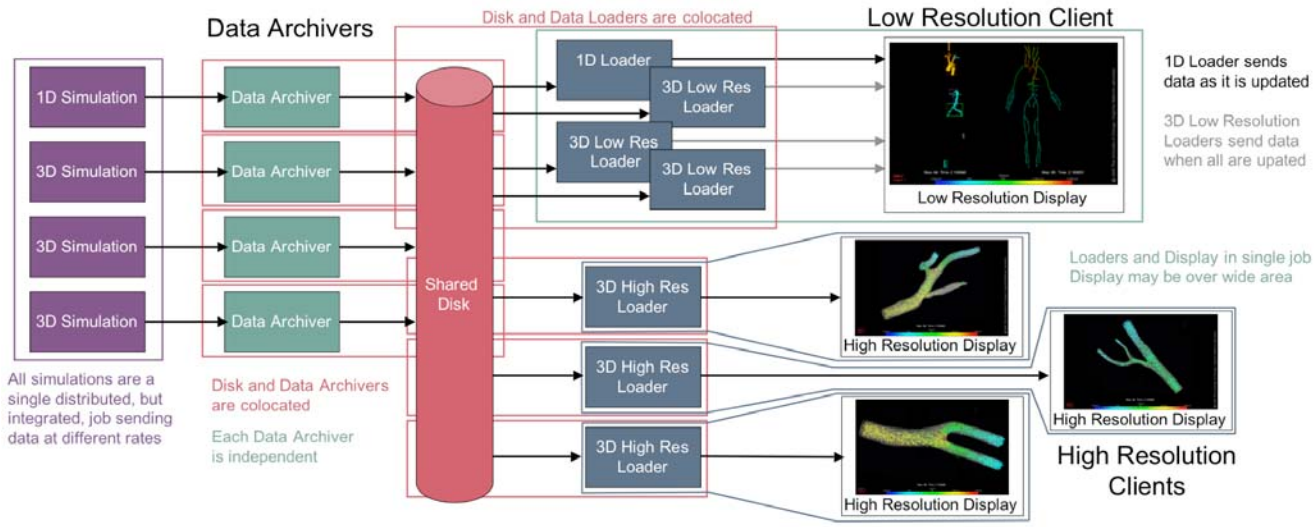

Figure 2: Overview of the components of the arterial tree simulation and visualization.

data from the simulation and is responsible for writing that data to disk, along with associated metadata. There are two visualization clients, one high resolution, the other low resolution. These clients have Data Loaders, which are colocated with the Data Archiver, and Displays, which can be at a remote location. The clients use the metadata written by the Data Archiver and interactions with each other to determine the data to be visualized. These three components are described in greater detail in the following sections.

5 DATA ARCHIVER

The Data Archiver communicates directly with the running simulations. All of the simulations, while distributed among several geographically distributed sites, are all part of a single integrated job. However, there is one instance of the Data Archiver for the 1D simulation and one for each of the 3D bifurcation simulations. The simulations connect to the Data Archivers via a connect/accept mechanism made available in the MPICH-G2 implementation. Each of the bifurcation meshes is given a numerical bifurcation ID. When each Data Archiver starts, it is given the ID of the bifurcation that it will be responsible for (0 is used for the 1D data simulation). Each Data Archiver then calls MPI_Accept and publishes the host and port where it is listening for a connection to a well-known location using a call to wget and a php script. Each simulation then uses a reciprocal call to wget and the php script to discover the host and port for the appropriate Data Archiver, and calls MPI_Connect to connect to it.

Once connected, the Data Archivers and the simulations are members of the same MPI_Communicator and use standard message passing routines to communicate with one another. After exchanging some initial setup information, the simulations send the 3D mesh information to the Data Archiver. This information consists of the coordinates of each of the elements in the mesh, which may be of varying resolutions. The communication is done for both the full 3D mesh and the reduced boundary mesh. The Data Archiver uses the Visualization Toolkit (VTK) [13] to create an unstructured grid object from each of the meshes and write them to a local shared file system. The data is written to a location determined by several variables provided to the Data Archiver at runtime and a letter that indicates whether the data is for the full mesh, “a” for artery, or the reduced mesh, “b” for boundary. These variables include a Base Directory, a Run ID (a unique identifier for the current run of the simulation), and the Bifurcation ID.

As the simulation proceeds and output data is produced, the simulation sends this data to the Data Archiver. As mentioned previously, the 1D data is quite small, 55 float values per time step. This data, along with its time-step information, is sent after each time step. The data for all time steps is written to a single binary file. In addition, a metadata file is also written. This file, named *`1D_latest.txt`*, stores the number of the latest time step that was written to the data file, along with the corresponding simulation time value.

Because the geometry of the mesh does not change over the course of the simulation, only the data values associated with the points on the mesh are transferred to the Data Archiver as the simulation runs. This approach eliminates the transfer of redundant data, thus reducing the impact on the performance of the simulation, especially when sending data over wide-area networks. Again, as indicated earlier, this data is sent at different intervals. The data for the boundary mesh is sent every third time step, and the data for the full 3D mesh is sent every fifth time step. The pressure scalar values and the velocity vector values for each mesh are written to separate binary files for each time step received. Several metadata files are also written for the full mesh and the boundary mesh for each bifurcation. Similar to the 1D metadata, two files, called *`a_latest.txt`* and *`b_latest.txt`* for the full and boundary meshes, respectively, contain information about the last time step that has been successfully written to disk for the corresponding mesh. Two other files, named *`a_list.txt`* and *`b_list.txt`*, consist of information about all of the time steps that have been written for that mesh. The clients then use this metadata to determine which time steps to display.

6 LOW RESOLUTION CLIENT

The Low Resolution Client comprises several components that work together to provide a high-level overview about the state of a running simulation in near-real time. It processes and visualizes the 1D data and the 3D boundary data as it is written to disk, providing the user with an overview of the current state of the simulation and the ability to control which bifurcations are examined in greater detail with the High Resolution Client. Because none of these components are directly connected to the simulation itself, or even to the Data Archiver, they cannot negatively impact the stability of the simulation. The Low

Resolution Client can start and stop, whether intentionally or as the result of a failure, without any effect on the simulation. This feature becomes increasingly useful as the expected run time of the calculation increases, enabling the scientist to periodically check on the running simulation.

The components include Data Loader processes and a Display process. Similar to the Data Archiver, there is one Data Loader for the 1D data and one for each of the 3D bifurcation meshes. Unlike the Data Archiver, however, these processes are all part of a single instance of the application. The Display process is also part of the same application instance. However, it is typically not colocated with the Data Loaders. This arrangement enables remote visualization of the data, allowing the researcher to monitor the simulation and investigate the results using modest local graphics hardware, without the need to transfer and store all of the data locally. Just as with the simulation, MPICH-G2 is used to submit the Data Readers and the Display as separate subjobs of a single MPI application, to be executed on two different resources. MPICH-G2 again handles the details of submitting the subjobs to the separate resources and establishing the communication paths between them.

6.1 Data Loaders

The Data Loaders are colocated with the Data Archivers and therefore can access data from the same shared file system. The 1D Data Loader reads from the 1D metadata file written by the Data Archiver, using file locks to ensure that memory access by the two processes is mutually exclusive. The loader keeps track of the last time step that it has read and compares this to the current value in the *`1D_latest.txt`* file. If the value in the file is newer, it reads the values for the corresponding time step from the 1D data file. Once read from disk, these new values are sent to the Display process, along with the time step and simulation time values, using MPI_Send.

The 3D Data Loaders for the Low Resolution Client are responsible for reading the boundary data for their designated bifurcation. Because the size and complexity of the 3D meshes vary from one to the next, the simulations are executed on a varying number of processors, in an attempt to keep their completion times synchronized. However, it is not uncommon for some bifurcation simulations to complete earlier than others. So, in order to keep the 3D Data Loaders synchronized, they all read the latest time step from the *`b_latest.txt`* file for their assigned bifurcation. All of these latest time step values are then compared, and the lowest one is determined. Collectively the loaders all keep track of the last time step that was read from disk. When the lowest value is greater than the previous time step that has been loaded, all of the 3D Data Loaders read the data for their respective bifurcations for this new time step. If this is the first time step to be loaded, as at the beginning of the simulation, the 3D Data Loaders first read in the geometry data for the boundary mesh for their designated bifurcation, creating a VTK unstructured grid object. For all subsequent time steps only the data values are read, and the unstructured grid object is updated to reflect these new values. As each 3D Data Loader finishes reading the data for the current time step it sends the updated unstructured grid object, along with the time step information, to the Display process. Because these boundary meshes are relatively small, just over 2 MB for the largest one, there is no performance penalty for sending the whole mesh, rather than just the data values.

6.2 Display

The Display process is responsible for providing visual representations of the 1D and low-resolution (boundary) 3D data produced by the simulations. Recall that the 1D data consists of a single value for each of the 55 major arteries in the human body, specifically the pressure on the arterial walls at the inlet of the artery. Even when only a subset of the 3D bifurcations is simulated, the 1D data is still calculated for all 55 arteries. To visually represent these values, we borrowed from a popular 2D diagram of the 55 major arteries. Using VTK objects, we constructed a 3D model of this diagram. The model allows each of the arteries to be easily addressed by name or by index. When the simulation is started, the researcher knows the range of pressure values expected to be produced by these 1D calculations. A color lookup table based on this value range is used to color the individual arteries in this model. When the Display process is started, it loads the artery model and creates the lookup table.

Another VTK object is used to represent the boundary data of each of the 3D bifurcations being simulated. This object, called the bifurcation group, is initially empty and is populated with the boundary meshes as they are received from the Data Loaders. The expected range of the pressure values over the entire set of bifurcations is similar to that of the 1D data. Therefore, the same color lookup table is used to color the points on each 3D mesh.

Once both of these VTK objects have been instantiated, a second thread is created. This thread is responsible for all communication with the Data Loaders. After exchanging some initial synchronization messages, it registers to receive data from the 1D and each of the 3D Data Loaders. It then loops, checking whether data has been received from any of the loaders.

The main thread of the Display process controls the graphics window where the objects are rendered; see Figure 3. It is laid out with two main regions, the artery model with the 1D data on the right and the bifurcation model with the 3D data on the left. Each region displays its respective models along with a label indicating the current time step and simulation time being shown. Because the models share a color lookup table, a scalar bar showing that table and its associated values is displayed across the bottom of the full window. Initially, before any data values have been received, all of the arteries in the 1D model are colored white, while no data objects get displayed on the 3D side, as all of the bifurcations are hidden until data has been received for them.

The two threads of the Display process use shared memory to exchange data and information. When the communication thread receives 1D data, a lock is put on the artery model and the color of each of the arteries is updated to reflect the new pressure values. The time-step information is also updated. Once the lock is released, the main thread updates the display window to reflect the newly colored arteries and the updated time information.

Each time through the execution loop the communication thread tests to see whether any new messages have been received from any of the 3D Data Loaders. If any new messages have been received, a lock is put on the bifurcation group object, and all new messages are processed. The receiving thread knows which bifurcation the data it received was intended for based on the ID of the Data Loader that sent the message. If this is the first data message received for a given bifurcation, the unstructured grid that was received is added to the bifurcation group object, and its status of “hidden” is changed to “display”. Otherwise, the unstructured grid that was received replaces the existing one for this bifurcation. If the time step information in this message is greater than what is currently being displayed for the bifurcation group, that information is updated as well. Once all new messages received since the previous pass through the loop are processed, the lock is released. The main thread can now update the display window to reflect any new data that was received. The 3D bifurcation objects are rendered on the left side of the display.

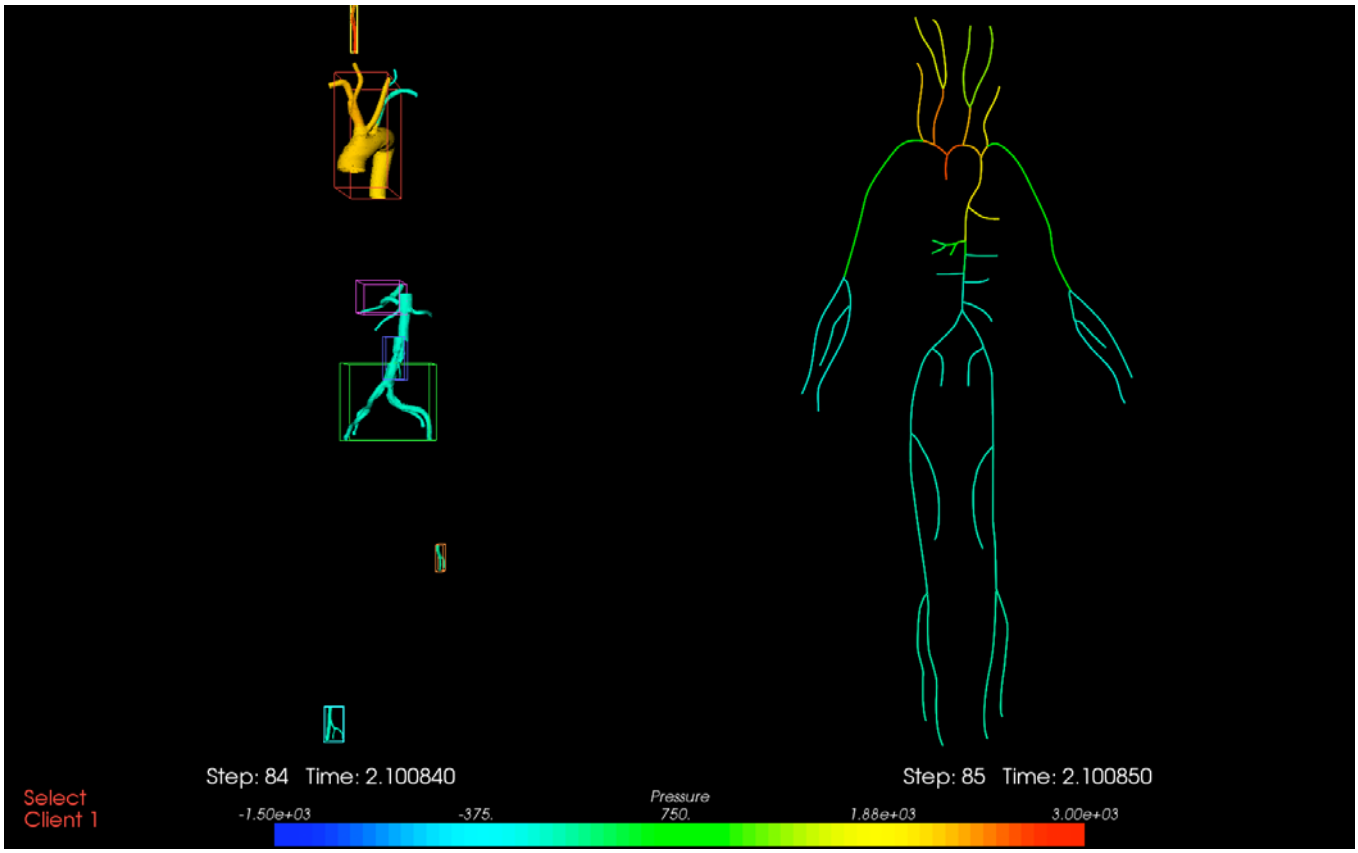

Figure 3: The Low Resolution Client displaying the 1D data for the 55 major arteries on the right, and the boundary data of 11 of the 3D bifurcation meshes on the left.

Their positions reflect where they would actually be located in the body, relative to one another.

Seeing the 3D bifurcations side by side with the 1D artery model, one can match the corresponding arteries from the two representations, especially if one has an intimate understanding of the arterial system. Having these two displays also gives the scientist a better idea of what is happening in the simulation. For instance, one can see that the pressure is initially greatest at the first artery, that closest to the heart. As the simulation progresses, this high pressure travels through the arterial tree toward the extremities. Seeing the results of the 3D simulations on these reduced boundary meshes both assures the scientist that the simulation is progressing as expected and enables the viewer to discover areas of interest to explore in greater detail. This in-depth investigation is done through the High Resolution Client, described in the following section.

7 High Resolution Client

The High Resolution Client enables exploration of a full 3D bifurcation mesh. Its architecture is similar to that of the Low Resolution Client, but with some notable differences. Like the Low Resolution Client, it comprises several components that work together to enable the visualization of data from a running simulation. There again are Data Loader processes and a Display process, which execute as a single application while on separate, usually distributed, resources, enabling remote visualization of the simulation data. Likewise, MPICH-G2 is used to start the processes and establish communication between them. They also have no direct interaction with the simulation or the Data Archiver, and thus cannot negatively impact the performance or stability. This feature again enables the Client to be stopped and started with no effect on the simulation.

Unlike the Low Resolution Client, the High Resolution Client visualizes only a single 3D mesh at a time. However, in addition to providing a simple view of the boundary data, it applies other visualization methods to the data of the full 3D mesh. Multiple instances of this client can be run simultaneously in order to view multiple bifurcations at the same time.

7.1 Data Loaders

The Data Loaders are again colocated with the Data Archivers and access data and metadata from the same shared file system, using file locks to ensure data integrity. Two methods are used for selecting which time step of the simulation data to display. In "Latest Step" mode the Data Loader looks in the `a_latest.txt` file to discover the last time step that has been successfully written to disk. If this is a later time step than the one currently being displayed, the data for this new time step is loaded. In "Playback" mode the Data Loader reads through the `a_list.txt` file and sequentially loads the data for each time step, one at a time. When the end of file is reached, it rewinds back to the beginning and starts over.

There are two Data Loader processes. Both processes read the same bifurcation mesh. The bifurcation ID for this mesh is initially given as an input parameter but can later be changed, as described in the following section. Both of the Data Loader processes read the pressure scalar values and the velocity vector data. Once read, the unstructured grid is updated with these new values. One process then generates ten isosurfaces using values evenly distributed across the range of pressure values in the

Figure 4: Components of the arterial tree visualization application running on a tiled display at SC|05.

current time step. The geometry for these isosurfaces is then transferred to the Display process by using a call to MPI_Send. The other Data Loader process uses a glyph filter to generate arrows that depict the direction and magnitude of the velocity of the blood flow over a random sampling of the elements in the mesh. These arrows may be colored by either the pressures values, or the magnitude of the velocity vectors. The geometry of the arrows is then transferred to the Display process, again by using a call to MPI_Send. Both processes also send information about the current time step along with their data. While each of the Data Loaders is currently only a single process, each could be parallelized to take advantage of multiple processors for calculating the glyphs and isosurfaces.

7.2 Display

As in the Low Resolution Client, the Display process for the High Resolution Client is responsible for rendering the visual representations of the data. When the Display process is started, it is given the ID of the initial bifurcation that it will visualize. It reads a local copy of the boundary data for this bifurcation, without any data values, and renders it in the graphics window. The boundary data is given a neutral grey translucent color and is used to give a frame of reference for the shape of the bifurcation being simulated. VTK objects that will be used to display the isosurfaces and glyphs generated by the Data Loaders are also created. Also added to the display are the color lookup table and its associated values, and a label indicating the current time step and simulation time being shown, initially set to zero.

As with the Low Resolution Client, here again a second thread is created that is responsible for communicating with the Data Loaders. Again the threads use shared memory to exchange data and locks to ensure mutually exclusive access to the data. The communication thread registers to receive data from each of the Data Loaders. Because both of the Data Loaders will be sending data for the same time step, the communication thread waits until it receives the data from both before processing the data. Once all of the data for the time step is received, the VTK objects are locked, and the isosurface and glyph objects are updated with the new data. When the lock is released, the main thread updates the display with the newly received data.

Users can interact with the Display to change the position and orientation of the bifurcation data. They can also zoom in to get a closer look at a particular region of interest, or out to see the entire bifurcation. By pressing a key on the keyboard the users can also easily switch between "Latest Step" mode, where the display is updated to show the latest time step of the simulation to be completed, and "Playback" mode, which animates through all of the time steps that have been completed so far. When the key is pressed, the main thread captures this event and determines whether a change of mode should be triggered. If so, it communicates this change to the communication thread via shared memory. The communication thread then uses MPI_Send calls to notify the Data Loaders of the change in mode. The Data Loaders can then switch from reading the `a_latest.txt` file to the `a_list.txt` file, or vice versa, depending on the current mode.

8 TYING IT ALL TOGETHER

As mentioned earlier, the Low Resolution and High Resolution Clients work together to both provide a high-level overview and enable detailed exploration of the arterial tree simulation data. The Display components of the clients are intended to run on the same resource. For the SC|05 demonstrations a tiled display was used to display the Low Resolution Client and multiple instances of the High Resolution Client, each on a different tile of the display; see Figure 4. In order to simplify the startup process of all of these clients, including the Data Loader components running on the UC/ANL TeraGrid visualization resource, a script and configuration file are used. The configuration file includes information such as what resource manager each subjob should be submitted to, the path to the application executables on the different resources, directory paths to where the data should be written, the bifurcation IDs that will be included in the current run, and the number of High Resolution Clients that should be started. MPICH-G2 makes use of RSL (Resource Specification Language) [14] to describe the job requests that it submits. The startup script reads the configuration file, creates the appropriate RSL expression, and calls MPICH-G2's mpirun to submit a job request for the each of the clients to be started.

When each of the High Resolution Clients is started, it is given the initial bifurcation ID that it will be responsible for. The Low Resolution Client is also given this information, so it knows how many High Resolution Clients are running and which bifurcation each will be displaying. Because the Display components of all of the clients are running on the same resource, in this case a 9-node cluster, they share a common file system. Thus the clients can communicate through the use of shared files, again using locks to ensure exclusive access. In the display of the Low Resolution Client some of the 3D bifurcations have colored bounding boxes around them. The bounding boxes indicate that these bifurcations are being displayed in one of the High Resolution Clients. The color of the bounding box matches the color of a label that is drawn in the display of the corresponding High Resolution Client. The user can select a different bifurcation to be displayed in one of the High Resolution Clients by simply picking one of the bifurcations in the Low Resolution Client that does not have a bounding box. Keys on the keyboard are used to select the High Resolution Client for which picking is currently active. The index and color of the active High Resolution Client are displayed in the lower left corner of the Low Resolution Client. When a new bifurcation is picked, the Low Resolution Client writes its bifurcation ID to a file for the active High Resolution Client. When the High Resolution Client checks the file and discovers a new bifurcation ID, it uses MPI_Send to notify its Data Loaders of the new value. The loaders then discard the unstructured grid that they are currently using, load that of the new bifurcation ID, and start reading data and metadata from the corresponding location.

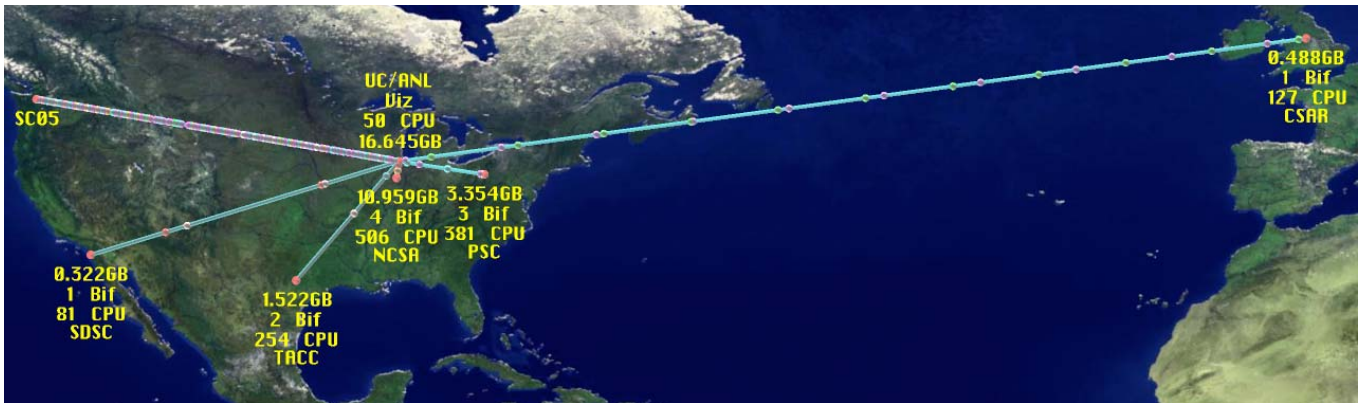

Figure 5: Image of geeViz displaying monitoring information of human arterial tree simulation and visualization demonstration during SC|05.

9 Additional Features

In addition to enabling the real-time remote visualization of the simulation data, this application provides several other features including performance monitoring, playback, and improved network performance.

9.1 Performance Monitoring

As the compute environment that simulations are executed in becomes more complex, providing users with feedback about the progress of long-running computations becomes increasingly important. To enable the users of our application to monitor its performance, as well as that of some aspects of the simulation, we have instrumented several of the components. Specifically, we used geeViz [15], a previously developed system for visualizing Grid-enabled environments. It provides an API that can be used to easily log events of interest, and an application that is used to visualize those events.

The first component to be instrumented is the Data Archiver. Whenever it receives data from the simulation, it calculates the bandwidth between the simulation and itself. It does so using a timestamp that was sent with the data, the local time when the data was received, synchronization information that was exchanged when the two processes first connected, and the size of the data buffer that was received. The Data Archiver then logs a transfer event, which includes the source and destination of the transfer, the bandwidth, and the amount of data received. The geeViz application displays an image of a world map. It plots the location of the source and destination of the transfer event on the map and draws a link between them. Spheres are moved along the link in the direction of the transfer, at a speed based on the bandwidth. Also displayed are labels of the locations of the endpoints and the amount of data that was either sent or received. Other events are used to indicate the number of bifurcations being calculated at a given site. This information is added to the display.

The SC|05 demonstration involved multiple compute sites, some simulating multiple bifurcations. Figure 5 shows the geeViz display with the locations of compute sites (NCSA, SDSC, TACC, PSC, and CSAR) plotted on the map. Each is labeled and indicates how many bifurcations it is simulating, as well as how much data it has produced so far. Links connect each of these sites with UC/ANL, where the Data Archivers are running. Displayed here is the total amount of data received, for all five computation sites combined. When multiple transfers are sent along the same link, such as the boundary data and the full 3D mesh data coming from the same site, they are represented by different colored spheres moving along the link.

The High Resolution and Low Resolution Clients are also instrumented. Using a formula similar to the one in the Data Archiver, the Display processes calculate the bandwidth from the Data Loaders whenever they receive data. Transfer events are again logged and visualized in the geeViz display. Figure 5 shows a link between UC/ANL, where all of the Data Loaders are running, and the SC|05 exhibit floor in Seattle, WA, where all of the Displays are running.

Placing the mouse over any of the sites on the map will bring up a graph showing the bandwidth performance into and out of that site over time. The user can also interact with the geeViz display to pan and zoom to focus on particular areas of interest. Using this information on the geeViz display, the scientist can monitor the progress of the simulation. This feature proved to be particularly useful at the beginning of the simulation, to see when sites were starting to send data. The fact that some sites were not yet reporting results, while others had been for some time, indicated a potential problem and prompted further investigation.

9.2 Playback of Archived Data

In addition to visualizing the results of the arterial tree simulations in real time, the data is also archived and can be viewed at a later time. Because the clients do not communicate with the simulation directly, they are not dependent on it. They determine which time steps to load and visualize based on the metadata in the `*_latest.txt` files. Rather than relying on the Data Archivers to update these files, a script can be used to update them instead. The script takes a configuration file, which indicates the rate at which each of the files should be updated. This should be the same rate at which the simulation output its data. In the examples we've discussed, the 1D was updated every step, the 3D boundary data every three steps, and the full 3D artery data every five steps, but these rates may vary. For instance, if the elements in the mesh are calculated to a polynomial order of 12, it may make more sense to output the data less often. The configuration file can also contain a delay value, which is used to determine how long the script should wait between simulation time steps.

9.3 Network Performance Boost

Poor network performance when communicating over wide-area networks, such as when the Data Loaders of the High Resolution Client send data to the Display process, has been identified as a major bottleneck in the performance of the visualization application. By default MPICH-G2 uses TCP over the wide area.

The high bandwidth and high latency found in the networks used here result in the TCP protocol entering congestion avoidance mode, where the sender-side congestion window becomes too small. In an attempt to overcome this obstacle, MPICH-G2 employs a technique that uses UDT [16], a reliable UDP-based protocol that does automatic optimization, including congestion avoidance. By setting a few attributes on both the sending and receiving sides of a transfer, MPI_Send can be configured to use the UDT-based protocol. This was done in the High Resolution Client, for data transfers between the Data Loaders and the Display process. While monitoring transfers between these two components using the information provided by geeViz, network bandwidth performance was observed to increase by an order of magnitude. Thus, enabling UDT clearly eliminated the network as the performance bottleneck.

10 Conclusions and Future Work

We have articulated the value of visualizing simulation data early and often, while the simulation is still running. In particular, the productivity of long-running simulations can benefit from this capability. Equally important is the ability to do this with little impact on the performance and stability of the simulation. We described an application we developed to provide these capabilities to a human arterial tree simulation. The various components of the application, including the Data Archivers and the High Resolution and Low Resolution Clients and their elements, the Data Loaders and Displays, were described in great detail. This effort, while focused on a single application, has illustrated the benefits that a flexible, low-impact runtime visualization application can have on demonstrating, monitoring, and debugging a large distributed simulation. The benefit of the use of low- and high-resolution renderings for immediate feedback to users was also demonstrated. Finally, it showed that the visualization community can greatly benefit from advances within the Grid community for the integration of visualization into running simulations and as part of the analysis pipeline.

We have identified the following areas in which future work can increase the usefulness and robustness of the application. The way that the simulations currently discover the contact information of the Data Archivers is through the use of wget and a php script. While this method works, there are more robust ways in which this information discovery could be done. The use of a services-oriented architecture with an index service may be a more appropriate approach.

In the previous section we described performance gains that were obtained by enabling the use of UDT over wide area networks. Adding this functionality to transfers between the components of the Low Resolution Client, and between the simulation and the Data Archiver, could be done with little effort and would likely result in similar increased performance.

While the visualization capabilities provided by the High Resolution Client are valuable, they are just a start. Additional control over the parameters used in the visualizations is needed. The ability to manage the values used in creating the isosurfaces, for instance, would be helpful. Including more visualization algorithms, such as cutting plane and streamlines, would also be useful. As mentioned in an earlier section, parallelizing these visualization algorithms in the Data Loaders would provide enhanced performance.

Acknowledgments

We wish to thank Peter Schmitt for his work on the 3D artery model used for the 1D data visualization and Kelly Gaither for her insightful comments on draft versions of this document. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38, and in part by NSF under Grant OCI-0504086.

References

[1] I. Foster and C. Kesselman, "Globus: A Toolkit-Based Grid Architecture," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds.: Morgan Kaufmann, 1999, pp. 259-278.

[2] N. Karonis, B. Toonan, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *Journal of Parallel and Distributed Computing*, vol. 63, 2003, pp. 551-563.

[3] "The Department of Energy Office of Science Data-Management Challenge," November 2004.

[4] S. Dong, L. Grinberg, A. Yakhot, S. Sherwin, and G. E. Karniadakis, "Simulation of blood flow in human arterial tree on the TeraGrid," in *SIAM Conference on Parallel Processing for Scientific Computing*. San Francisco, CA, 2006

[5] J. Sobel, A. Forsberg, D. H. Laidlaw, R. Zeleznik, D. Keefe, I. Pivkin, G. Karniadakis, P. Richardson, and S. Swartz, "Particle Flurries: Synoptic 3D Pulsatile Flow Visualization," *IEEE Computer Graphics and Applications*, vol. 24, 2004, pp. 76-85.

[6] A. S. Forsberg, D. H. Laidlaw, A. vanDam, R. M. Kirby, G. E. Karniadakis, and J. L. Elion, "Immersive Virtual Reality for Visualizing Flow Through an Artery," in *IEEE Visualization*. Salt Lake City, Utah, 2000, pp. 457-460.

[7] J. D. de St. Germain, J. McCorquodale, S. G. Parker, and C. R. Johnson, "Uintah: A Massively Parallel Problem Solving Environment," in *Ninth IEEE Internalional Symposium on High Performance Distributed Computing (HPDC'00)*. Pittsburgh, PA: IEEE Computer Society, 2000, pp. 33-41.

[8] S. J. Sherwin, L. Formaggia, J. Peiro, and V. Franke, "Computational Modeling of 1D Blood Flow with Variable Mechanical Properties in the Human Arterial System," *International Journal for Numerical Methods in Fluids*, vol. 43, 2003, pp. 673-700.

[9] G. E. Karniadakis and S. J. Sherwin, *Spectral/HP Element Methods for CFD*: Oxford University Press, 1999.

[10] C. Catlett, The TeraGrid: A Primer, www.teragrid.org.

[11] G. von Laszewski, J. A. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M. L. Rivers, S. Wang, B. Tieman, and I. McNulty, "Real-time Analysis, Visualization and Steering of Microtomography Experiments at Photon Sources," in *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[12] N. Karonis, M. E. Papka, J. Binns, J. Bresnahan, J. A. Insley, D. Jones, and J. Link, "High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment," *Future Generation of Computer Systems*, pp. 909-917, 2003.

[13] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit, An Object Oriented Approach to 3D Graphics*: Kitware, Inc., 2004.

[14] Resource Specification Language (RSL) www.globus.org/gram/rsl_spec1.html.

[15] W. A. Allcock, J. Bester, J. Bresnahan, I. Foster, J. Gawor, J. A. Insley, J. M. Link, and M. E. Papka, "A Tool for Visualizing the Behavior of Large-Scale Distributed Systems," in *IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 179-187.

[16] Y. Gu and R. L. Grossman, "UDT: An Application Level Transport Protocol for Grid Computing," presented at the Second International Workshop on Protocols for Fast Long-Distance Networks, Argonne, IL, 2004.