Implementing a Visualization Tool for Adaptive Mesh Refinement Data using VTK

Terry J. Ligocki[1]
Visualization Group
NERSC/LBNL

Abstract

We are currently working with several research groups who use adaptive mesh refinement, AMR, techniques for calculations in computational fluid dynamics and cosmology. The computational fluid dynamics group needed a visualization tool to visualize AMR data and to distribute with their AMR library. Since none of the widely available visualization systems directly supported visualizing AMR data, we chose to extend one, VTK, via its Tcl/Tk interface.

This AMR data consists of grids at different resolutions. Grids at the same resolution don't overlap. In our initial implementation we allow the user to select grids which are then processed for visualization independently. The final results are then rendered together. In this paper, we describe why we did this, how we did this, the tool that resulted, the advantages and limitations of this tool, and future work (both short term and long term).

1. Introduction

This paper describes a tool we developed to visualize adaptive mesh refinement, AMR, data. The tool was built using the Visualization Toolkit, VTK, [VTK] and its Tcl/Tk [Tcl/Tk] interface. First, we give some context, define the problem we faced, and discuss why it was a problem. Next, we present an overview of the implementation that describes the structure of the tool. Following that, we highlight some of the implementation details and results. Finally, we discuss future work and how if relates to the work done so far.

1.1 Context

The Applied Numerical Algorithms Group, ANAG, [ANAG] at the National Energy Research Scientific Computing Center, NERSC, [NERSC] have developed a library, Chombo, [CHOMBO] to support Adaptive Mesh Refinement, AMR, computations. They wanted to distribute a visualization tool with Chombo for visualizing AMR data sets. To meet the various needs for distribution (e.g. low cost, availability, source code access, expandability), we decided to use VTK and its Tcl/Tk interface.

1.2 Overall Problem

Before we describe the specific structure of the ANAG AMR data, it would be useful to look at a broader view of this type of data and the problems visualizing it. Basically, researchers are starting to use more sophisticated data structures in their computations. Instead of single grids/meshes they are using multiple grids/meshes which may overlap. In addition, a given region in the computational domain may be represented at a variety of resolutions, which introduces a data hierarchy. Finally, these grids/meshes usually change dynamically as the computations proceed.

Most widely available scientific visualization systems (e.g. AVS, IBM DX, and VTK) don't directly provide the functionality to visualize these types of data sets. These systems are designed to visualize a predetermined number of individual "units" of data (e.g. a uniform grid, an irregular mesh, unstructured cell data, scatter data). In some cases, AMR data can be represented in these systems in some way (e.g. as unstructured cell data) but there are several problems. Often the storage required for the data set increases dramatically when converted and/or the visualization tools cannot interactively handle even moderately large AMR data sizes in their converted form. There are two other problems. First, overlapping grids can only be handled independently. Thus, in the overlap region there is no possibility of using some combination of data values from each grid to get data values within the overlap. Second, all the information about grid hierarchies is lost. This means, for example, that there is no way to specify that data values from grids deeper in the hierarchy should preempt data values from grids "above" them.

We have been looking at three categories of visualization for hierarchical collections of grids. In each case, some selection mechanisms are applied before the data is visualized. For example, all grids above some level in the hierarchy are selected. Once this is done the resulting hierarchical collection of grids would be visualized as follows:

[1] 1 Cyclotron Road, M/S 50F, Berkeley, CA 94720, tjligocki@lbl.gov

1) Each grid would be processed independently. All the results would then rendered simultaneously. In this case, overlapping grids might have overlapping visualization results (e.g. isosurfaces). These overlapping results, in general, wouldn't coincide.

2) If a finer grid overlaps a coarser grid then the visualization results from the finer grid would supercede the results from the coarser grid in the region of overlap. Thus, there would be no overlapping results displayed. There still might be gaps/seams between the results generated from adjacent grids.

3) The entire hierarchical collection of grids could be processed in a holistic fashion. Here interpolation and extrapolation functions would be used to get smoothly varying data values over the entire computational domain. Thus, there would be only one result (not a collection as in the previous two cases) and this result wouldn't contain overlays, gaps, or seams due to the grid structure.

The first category is the easiest to implement since each grid is handled entirely independently and at the end all the results of all the visualizations are simply rendered together. Various forms of parallelism can be used if necessary and there are no synchronization issues until the results are rendered.

There are several ways of implementing the second category. Basically, data in overlapping regions can be removed (based on the hierarchy) before any processing is done or the processing can be done independently on each grid and then superceded results in the overlapping regions can be removed. For example, in an isosurface computation, where a finer grid overlaps a coarser grid no isosurface is computed for the coarser grid or both isosurfaces are computed and the portion of the isosurface generated by the coarser grid in the overlapping region is removed. Much parallelism can be achieved in this case as long as the hierarchical structure of the data is respected.

The final category is the hardest to implement and will in most case require knowledge of the computation being preformed beyond simply possessing the data. The appropriate interpolation and extrapolation methods necessary to correctly blend the hierarchical data at all points in the computational domain are often defined very carefully by the researchers performing the computation. These methods are almost always specific to the computation being performed. These methods may also make it more difficult to parallelize the visualization processing because the grids are no longer treated independently. This would give a representation of the data that was the most consistent with the computation that generated it.

1.3 Specific Problem

In our case we had a much more specific type of hierarchical data and we chose to begin by implementing the first category of visualization described above. We chose the first category because we thought it was a good starting point. We would be able to learn more about Chombo and AMR data, VTK and its Tcl/Tk interface, and visualizing AMR data. Since each of the grids was handled independently, much of the specific structure of the data is unimportant so only a simple overview will be given in this paper. For more details see the Chombo WWW page [CHOMBO]. In this paper we will refer to the data Chombo generates as AMR data.

All the grids are axis aligned and all the cells in a given grid are the same size. There is a hierarchy of grids. At the coarsest level there is a grid which covers the entire computational domain. No grids go outside the extents of this grid since it coincides with the computational domain. From one level in the hierarchy to the next there is a fixed refinement ratio. Grids at any given level align with the corner of a cell in a grid at the next coarser level. Grids at any given level do not have to lie entirely inside any grid at the next coarser level. Thus, there is no required nesting or tree structure. Essentially, each level is a list of grids all with the same size cells and there is a list (or array) of levels.

This means that if any visualization system could handle lists of grids or, better yet, nested lists (i.e. lists of objects that might be lists) then we could directly perform the visualizations in the first category mentioned above. Unfortunately, this was not the case. At the time we were first planning to create this tool, we became aware of work being done at NCSA [NCSA] which involved parallelizing VTK and extending it to a new class of objects designed to address AMR data visualization. In fact, NCSA was helping cosmologists who were using a predecessor of the Chombo library. Thus, we began to collaborate with John Shalf and Matthew Hall at NCSA.

Since they were targeting very large data sets (gigabytes per time step) with a large number of levels of refinement (thirteen to sixteen), their immediate needs did not coincide with ours well enough for us to directly use they work. Specifically, we needed to be able to access their VTK extensions from Tcl/Tk so that we could more easily build a user interface. Unfortunately, this wasn't possible, although it may be addressed in the future. Also, we felt it might be difficult for the initial users of the Chombo library to get, build, and install VTK and then build and install the extensions correctly. We believe that in the future, the evolution of VTK and of these AMR extensions will make this easier. In the longer term, we are interesting in contributing to the efforts at NCSA and using the resulting extensions.

For these and other reasons we decided to attempt to extend VTK, via its Tcl/Tk interface to perform visualizations consistent with the first category of visualization described above. At the time we were unsure if we would be able to do this and end up with a useable visualization tool. We were pleasantly surprised that this was possible.

2. Overview of Implementation

In this section we will present the overall structure of the visualization tool we developed, ChomboVis [CHOMBOVIS], and place it in context with VTK, Tcl/Tk, and Chombo. The implementation breaks into three pieces. First, the data produced by Chombo needed to be read and put in some format compatible with VTK and Tcl/Tk. Next, all the data and visualization processing needed to be managed within Tcl/Tk. Finally, a user interface had to be built using these pieces.

There were also explicit several goals:

- The data would only be read as it was selected. In this way the user could quickly look at the coarser grids without waiting for the finer grids to be read.
- A core of visualization functions (e.g. 2D slices, isosurfaces) would be provided for AMR data.
- As much flexibility as possible would be retained so that future extensions would be relatively easy. We also wanted to structure the Tcl/Tk code and built the user interface using this as a guideline.

We knew that ChomboVis would probably have some limitations:

- It would not be able to be used directly with AMR data that did not come from computations using the Chombo library. Visualizing data generated with ancestors of the Chombo library might be possible by transforming the data into the current Chombo data format.
- It would not be able to handle large data sets. However, we didn't know what "large" would mean until we implemented ChomboVis. This will be discussed later.
- It would not do certain types of visualization correctly (e.g. volume visualization). Even in the case of 2D slicing we had to add additional functionality to handle slices through hierarchical data (see section 3.1.4).

With this in mind we began designing and implementing ChomboVis.

2.1 VTK and Tcl/Tk Interface Primer

At this point we need to digress a bit. It is important to outline some of the functionality of VTK and Tcl/Tk and how VTK integrates with Tcl/Tk. We needed to understand all these things in some detail to implement ChomboVis and we believe it will be of value to the reader in understanding what follows. Conversely, we have tried to present only what is necessary and not everything we learned in the process of implementing ChomboVis. We used VTK 2.3 and Tcl/Tk 8.0.4 for our initial implementation. All that follows is based on our experience with these specific versions but it is probably applicable to more recent versions with some modifications.

VTK itself is a C++ library that contains a great deal of functionality, it is freely available, and it can be a challenge to use effectively (we have found the VTK user group invaluable in addressing this challenge). Also of great benefit are the various interfaces between VTK and other object-oriented languages/packages (e.g. Tcl/Tk, Python, and Java). The Tcl/Tk interface is achieved by "wrapping" original C++ code with additional C++ code, compiling the result into Tcl/Tk libraries, and dynamically loading these libraries into Tcl/Tk.

Essentially, a Tcl/Tk command is added for most VTK classes. This command is called to create an object in that class whose name is given as an argument. This name is used to create a new Tcl/Tk command with that name. This Tcl/Tk command allows access to all the methods available to the object it represents. This is all done by dynamically adding commands to Tcl/Tk. All the underlying VTK data is passed around indirectly in Tcl/Tk. A unique string name is generated and this is used as a key into a hash table that contains pointers to the actual data. All that is visible in Tcl/Tk are the unique string names. We needed to use this technique to generate Tcl/Tk objects that represented the data and could be passed to VTK for processing.

In addition, we used similar techniques to extend Tcl/Tk and add commands for reading in AMR data, colormaps, etc. This capability was invaluable in adding functionality and getting around limitations in Tcl/Tk and VTK. This also allowed the different parts of ChomboVis (i.e. VTK, HDF5, our Tcl/Tk code, our Tcl/Tk extensions) to be developed and built separately and then dynamically brought together.

2.2 Accessing Chombo Data

The AMR data was stored using HDF5 [HDF5]. With the help of ANAG we implemented several new Tcl/Tk commands to read the AMR data. One command read and returned the header information of the current data set (e.g. number of levels, refinement ratios, physical dimensions). This was used to initialize various global variables and data structures within our Tcl/Tk code.

Another command read in a level of data and returned a Tcl/Tk list that contained each grid as a VTK object referred to by a unique string name and various other useful data items. To do this, code from VTK was duplicated in our Tcl/Tk extensions. Thus, this depended strongly on the way VTK interfaces to Tcl/Tk and was highly version dependent! Unfortunately, we were not able to develop any other efficient method to achieve this functionality.

2.3 Managing the Data and Visualization

We made extensive use of Tcl/Tk arrays to store and retrieve data and to manage the VTK visualization pipelines. Tcl/Tk arrays are indexed by strings and are essentially efficient hash tables. By using a few global arrays of this type we were able to get the functionality, structure, and efficiency we needed.

The data was read in a level at a time as it was selected. When a level was read in, all the grids for each data variable on that level were read using HDF5 and converted into VTK structured points. These were, in turn, converted to Tcl/Tk commands in a manner consistent with VTK. The commands that represented the AMR grids were then stored in a Tcl/Tk array indexed by level number, grid number, variable name, and a keyword (“data”).

Once the data was read in, a VTK pipeline was set up for each grid and visualization technique (e.g. 2D slicing, isosurfaces). These pipelines were set up regardless of whether a visualization technique was currently visible or not. As a result, for each grid approximately half a dozen VTK pipelines were created. These were also stored in the same Tcl/Tk array as the data, indexed by level number, grid number, and a keyword (e.g. “slice_x_actor”, “iso_actor”). Note: the pipelines were not indexed by the variable name because the only thing that changed when a different data variable was selected was the input to the pipeline, not the pipeline itself.

This method of organizing the data and the visualization pipelines worked well. To extend ChomboVis to more types of visualization was straightforward and was done several times as ChomboVis was developed. All that was necessary was the creation of more VTK pipelines and the addition of entries in the Tcl/Tk array with new keywords.

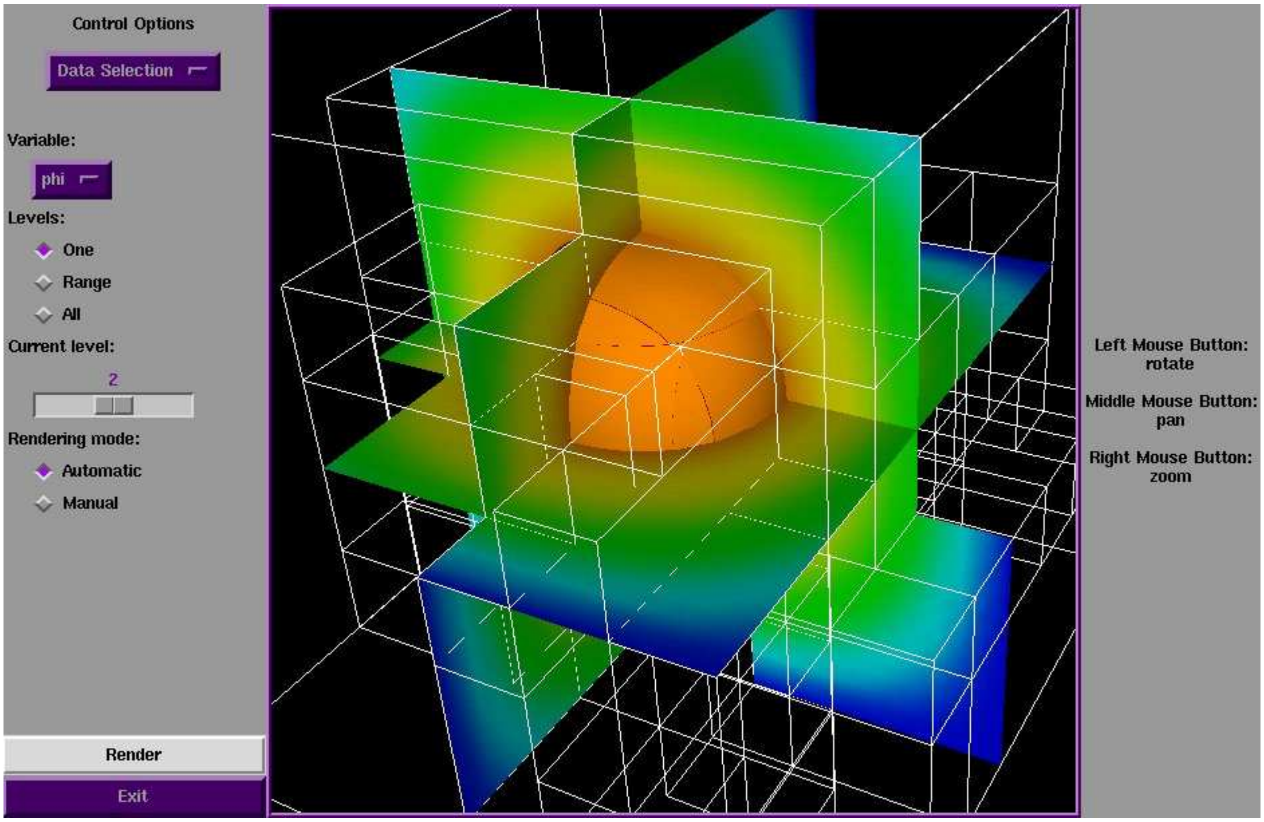

Figure 1: AMR visualization showing grid bounding boxes, 2D slices, and isosurfaces.

2.4 Organizing the User Interface

In order to make the user interface useable and extensible, we decided to have a portion of the ChomboVis window dedicated to controlling the visualization (see figure 1). Control was grouped by functionality (e.g. data selection, grid display, isosurfaces) and each group had its own control panel that was displayed when selected via a menu at the top of the left-hand side of the screen. Thus, to control the isosurfaces, the user would select "Isosurface" on the menu and that would make visible the isosurface controls. These controls could then be changed as needed. This made it easy to add new types of control and/or visualizations and still manage it all in a limited area.

The ChomboVis Tcl/Tk code underlying this was also organized in a way that reflected the organization of the user interface. In addition to being organized by type of control and/or visualization, the Tcl/Tk code was organized into code that supported the user interface and code that supported the VTK pipelines. Thus, changes and additions to the user interface were made in one Tcl/Tk file and these often referenced Tcl/Tk functions in a different file (where the VTK pipeline was managed). This kept the individual Tcl/Tk file sizes reasonable and divided the files in a logical and consistent manner.

3. Details and Results

In this section we present some of the details of ChomboVis and the results of using it with some initial data sets. The goal is to give a solid idea of the functionality that has been implemented, any interesting issues that arose, how we addressed these issues, and our view of the overall result.

3.1 Functionality

The functionality implemented in ChomboVis was divided into six categories that correspond to the menu of "Control Options" in figure 1. The user could control data selection, colormaps, grid display, 2D slicing, isosurfaces, and volume visualization. Additional functionality we would like to add is described in section 4.1.

3.1.1 Data Selection

The data set being used is specified when ChomboVis is started (as a command line argument). It cannot be changed after that time. The user was allowed to select the scalar data variable of interest. In addition, a range of levels could be displayed varying from a single level to all levels. There was no way to select group of levels that were not contiguous (e.g. levels 1 and 3). Finally, immediate rendering could be turned on or off. By turning immediate rendering off, several parameters could be changed without waiting for the renderer to redisplay. Once this was done the user could turn immediate rendering back on or explicitly tell ChomboVis to render the visualization again.

As was stated above, once a level was selected all the grids on that level were read in. Based on the currently selected variable, VTK visualization pipelines were set up for all the new grids. These may or may not be added to the renderer depending on whether the given visualization was selected by the user. All the forms of visualization could be turned on or off. Once a level was read, the data associated with that level was never deleted nor were any of the VTK visualization pipelines. Thus, ChomboVis always grew in size and complexity. Initially, the coarsest grid was selected and read. A bounding box and three 2D slices through the center of the grid were displayed.

3.1.2 Colormaps

A very basic ability to select and read in colormaps was provided. A default grayscale and full color colormap could be selected or a user defined colormap could be read in. The user defined colormap had 256 entries, which specified a color (as RGB or HSV) and an opacity (or alpha) value. The scalar data values were mapped to colors linearly using the minimum and maximum values in the data. The opacity was only used for volume rendering. This could be extended to other types of visualization (e.g. isosurfaces) but this hasn't been done.

3.1.3 Grids

The user was permitted to view the bounding box of the entire domain at all times. For all the selected grids the user could view nothing, all the bounding boxes, or all the cells in all the grids. Viewing all the cells was not useful most of the time because there were too many cells but this could be helpful at times to see relationships between grids at different resolutions. To use this more effectively it would have helped to be able to select specific grids within the already selected levels (see section 4.1).

One thing that wasn't apparent to the user was that the geometry for the bounding boxes of the grids and for the cells were generated by the function that reads each level. This was done because doing this within VTK was too expensive, especially for the cell geometry. Basically, VTK generated line segments for each edge of each cell of each grid. For a grid in Chombo many fewer line segments could be generated and give the same result. This was a case where VTK provided the necessary functionality in too general of a form for our needs and, in this case, it proved too expensive for us to use. This understandably happens in most visualization systems and we were happy that it was possible to easily improve this by using our Tcl/Tk extensions.

3.1.4 2D Slices

The user could view 2D slices of the data that were aligned with the coordinate planes. These slices were rendered as they were oriented in the 3D data set. There were three slices available, each of which corresponded to one coordinate (x, y, or z) being held constant. The slices could be individually moved throughout the data set and made visible or invisible. For a single grid this was fairly basic functionality. For AMR data there were some subtleties.

If one level was selected and the grids on that level didn't cover the entire domain (which they usually didn't), what should be drawn outside the grids? We elected to draw nothing in this case (seen figure 1). This appeared to work well and it gave the user a good idea of how the grids at a given level were abutting.

A more difficult problem appeared when multiple levels were selected. If we held to the design choice that all the grids are processed independently then a problem arose where coarser grids overlapped finer grids. In the overlapping region, geometry for each slice was rendered and it coincided in 3D. Thus, the results were indeterminate and, in most cases, were impossible to interpret.

To address this problem, we chose introduce an additional parameter into the visualization. We called the parameter an offset and it was used as follows. If only one level was selected a slice through that level was generated (as before). If more than one level was selected then each level was sliced independently. The coarsest slice was rendered in the correct orientation and position. The next coarsest slice was offset the amount specified in the direction which was constant for that slice (e.g. a slice of constant x was moved in the x direction). This results in the slices from each level being layered and independently visible when the offset was large enough. If the offset was zero then the original problem discussed above occurred.

This was not a perfect solution to the problem. It could be hard to see between the slices for each level, only the coarsest and finest level's slice could be viewed parallel to the screen, and if slices in more than one direction were visible the result was confusing.

3.1.5 Isosurfaces

Of all the types of visualization, the generation of isosurfaces was the most straightforward in this context. Basically, each selected grid was handled independently and all the geometry generated was rendered simultaneously. This, of course, resulted in overlapping isosurfaces when multiple levels of grids were selected. In addition, when only one level was selected there were sometimes visible artifacts between adjacent grids. In some cases these were gaps but in other cases the surface was continuous but the surface normals didn't match at the boundary (see figure 1). One way to correct this would be to extend the grids one cell (i.e. use ghost cells) and only generate the isosurface and normals within in bounds of the original grid.

In addition, isosurfaces could be made visible or invisible, the isosurface value could be set, and the color could be white or could be based on the current colormap.

3.1.6 Volume Rendering

This was the most problematic type of visualization. Superficially, it was very similar to the generation of isosurfaces and was implemented in much the same way. Unfortunately, in most cases, superimposing the results of many independently generated volume rendering was not the same as volume rendering the original data.

When only one level was selected, because the grids don't overlap, it was possible to get consistent results. Volume rendering the grids separately and then compositing them correctly (i.e. in the right order) sometimes matched the results of volume rendering all the data present as a whole. Even in this case, it wasn't clear that the results we were getting were correct. This may have been due to a misunderstanding of the volume rendering in VTK, a problem with the compositing order, or a problem with volume rendering in VTK 2.3 (especially in the case of multiple volumes).

When there were overlapping grids (i.e. when more than one level was selected) the results were definitely incorrect. Deciding what "correct" was wasn't obvious if the grids were to be treated independently. This was analogous to the case of 2D slices where multiple grids and multiple levels overlapped. Viewed in this way, the only solution might be to select one level at a time when the grids are being treated independently. The alternative would be to remove the coarser data in the regions of overlap but this would change the type of visualization being done from the first to the second category discussed above. As a result, this would be difficult to implement in the current framework of ChomboVis.

3.2 Results

As we implemented ChomboVis, we tested it on a number of data sets generated by software built on top of Chombo. In particular, a Poisson solver was used to generate data where the coarsest grid was 8x8x8, the refinement ratio was 2, and there were 5 levels of refinement. Thus, at the finest level, a grid that was 128x128x128 would cover the entire computational domain (no such grid was present in the data). There were a total of 175 grids in this data set and the size of data set was approximately 7 Mbytes. At the coarsest level of resolution there was only one grid and it covered the entire domain. This would be considered a medium size data set which could be (and was) computed on workstation.

In this case, ChomboVis started in a few seconds on several of our workstations (e.g. SGI Indigo 2 with Solid Impact graphics). This time was spent starting VTK, running some initial Tcl/Tk code, reading in the coarsest grid, and producing three 2D slices. The next coarsest level also contains 1 grid and that level could be selected, read, and displayed in about a second. The next three levels contained between 40 and 70 grids. It took between 4 and 7 seconds to select, read, and display three 2D slices. Once levels had been read in, the selection and redisplay time went down by a factor of about two. With this data set, at all levels of resolution, it was reasonable to interactively move slices and change isosurface values (i.e. changes were redisplayed in less than a second).

We were pleasantly surprised by these results. Once all the data was read in there were in excess of 1200 VTK pipelines/actors being managed via Tcl/Tk. All these actors were being added and removed from a single VTK renderer as levels were selected and various forms of visualization were made visible and invisible. This happened fairly efficiently and without any of the software crashing.

3.3 Comments

Although our initial results were surprisingly good, we know that the entire design had some fundamental problems. First, for data sets that are large (e.g. containing thousands to tens of thousand of grids), handling all the grids independently in Tcl/Tk will most likely make ChomboVis too inefficient to use even if it doesn't crash. Second, as has been pointed out throughout this paper, treating all the selected grids independently lead to artifacts in the visualization that may not be acceptable. Finally, none of the visualization computations were done so that parallel processing could be used to speed up ChomboVis.

On the other hand, we built an initial version of a visualization tool that could be used to look at data sets of interest to ANAG and, hopefully, other users of Chombo. It has allowed us to explore VTK and its Tcl/Tk interface and we have learned a great deal in the process. Without doing this we would have been unaware of many of the issues described in this paper.

4. Future Work

There are many ways in which we would like to extend and improve ChomboVis. There are also many ways the users would like to see ChomboVis extended and improved! Luckily, these two sets of extensions and improvements have a lot in common.

4.1 Extending Functionality

As soon as ANAG began looking at ChomboVis and using it they had suggestions for extending its functionality. We viewed this as a good sign as it meant they were using the tool. In general, they wanted to see more information shown in a textual form somewhere in the ChomboVis window (e.g. the number of levels, the size of the computational domain, the number of grids currently selected). This should not be hard to address and was important.

Another extension they wanted was the ability to view 2D slices in a separate window shown as a colored image. In this case they wanted data from finer grids to overwrite data from overlapping coarser grids and grid boundaries and/or cell boundaries to be shown. They wanted the ability to do contour plots of the 2D slices and select regions of data to see in some type of spreadsheet form (i.e. view the data values and/or data structures directly).

Similarly, they wanted be able to select grids and possibly cells directly from the visualization. They would like to use this as a way to modify the set of grids that were being visualized (e.g. visualize just this grid, remove this grid from the set of grids being visualized). In addition, they wanted to get more information about the selected grid (e.g. what were its physical dimensions and placement in space, what was the size of the data array).

Finally, they wanted to be able to view time varying data. A first step in this direction would be to extend ChomboVis so that the input data file could be changed from within the tool. Currently, the input data file can only be chosen when ChomboVis is started. This limitation made it easier to write the initial version of ChomboVis but will clearly have to be addressed. The next step would be to iterate through a list of data files under the control of the user.

4.2 Migrating Functionality

As the visualization tasks become more complicated and the data becomes larger it will be necessary to migrate functionality from the Tcl/Tk interface directly into VTK. In particular, if VTK was extended to work with lists of grids and/or lists of lists then much of what was being done in Tcl/Tk could be done directly with VTK. An additional benefit would be that various types of optimizations could then be done. For example, at each step in a VTK visualization pipeline, each grid could be processed in parallel.

Selection, picking, and interrogation of the data could be done more simply and made more powerful. By removing the additional layer between VTK and Tcl/Tk, we could have direct access to the data and data structures via C or C++. In addition, removing one level of indirection would make all the computations more efficient and hide less. In Tcl/Tk it may be fairly difficult to interpret what was picked and to what part of the data structure it belongs.

4.3 Other Types of Visualization

Once some of the functionality has been migrated from Tcl/Tk to VTK (see section 4.2), it would be possible to implement the second and third category of visualization (see section 1.2). Both categories do not treat all the grids as entirely independent data objects. Thus, representing, passing, and processing all the selected grids, as a single data object, would allow more sophisticated visualizations and it would still be possible to do all the visualization done by ChomboVis.

5. Conclusions

We were able to extend VTK, via its Tcl/Tk interface, to visualize AMR data sets. We did this in approximately one month and we were able to structure the Tcl/Tk code so that it was both modular and extensible. In the process, we learned a great deal about VTK, Tcl/Tk, and visualizing this type of data. This resulted in a useful tool, ChomboVis, and many ideas for extending it.

Acknowledgments

This work was supported by the Directory, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. In addition, we would like to thank ANAG and the NERSC/LBNL Visualization Group for their help during the initial development of ChomboVis. Specifically, we would like to thank Brian Van Straalen, Dan Graves, Wes Bethel, John Shalf, and Nancy Johnston.

References

[ANAG] http://seesar.lbl.gov/anag/

[CHOMBO] http://seesar.lbl.gov/anag/software/chombo.html

[CHOMBOVIS] http://seesar.lbl.gov/anag/software/chombovis.html

[HDF5] http://hdf.ncsa.uiuc.edu/HDF5/

[NCSA] http://zeus.ncsa.uiuc.edu/~jshalf/VTK/vtkSMP/

[NERSC] http://www.nersc.gov/

[Tcl/Tk] "Tcl and the Tk Toolkit", John K. Ousterhout, Addison-Wesley, 1994.

[VTK] "The Visualization Toolkit, 2nd Edition", Will Schroeder, Ken Martin, Bill Lorensen, Prentice-Hall Inc., 1998.