SV

Scientific Visualization

1 Introduction

Large scale computer simulations require a special class of tools for calibration, debugging and interpretation of results. Computer graphics and scientific visualization are currently finding increasing use in many large scale modeling efforts. Visualization provides an effective means of dealing with many aspects of numerical models involving large scale computation. Applications of visualization in this area include data set preparation, model calibration, code debugging, code performance analysis, and interpretation and presentation of model results. Visualization is especially useful in allowing modelers to interrogate the data sets produced by their models. Often a modeler will need to recalibrate a computer model and recompute to expose certain details or features in the underlying physical problem. Visualization helps the modeler to ensure that all of the detail available from a particular set of computational results is exposed before undertaking the often expensive task of recomputing.

This chapter discusses some recent advances in the application of scientific visualization software tools in the area of computational science. Specifically, we illustrate how visualization tools are being used in three areas of mathematical simulation modeling

(1) data entry and data integrity

(2) code debugging and code performance analysis

(3) interpretation and display of final results.

In each of these areas, we will first outline some general concepts and techniques and then discuss their implementation. Following a general overview, some implementations of scientific visualization techniques will be illustrated with examples. We believe that, in the area of scientific visualization, it is especially important to strike a balance between visualization concepts and techniques and the operational issues surrounding their implementation on specific hardware using specific software. In moving from one graphics workstation to another, a computational scientist encounters a great deal of diversity in graphics hardware and software capabilities. Also, graphics hardware and software are advances that are following a very steep growth curve. To remain current, the user of scientific visualization must endeavor to track this growth curve. A computational scientist who becomes implementation oriented (recipe bound), rather than concept oriented, is sure to fall behind.

Having argued that visualization concepts and techniques are foremost, we acknowledge that no scientific visualization gets done without implementation. For purposes of this chapter, we will assume that the implementation or actual visualization activity, takes one of two forms:

(1) The computational scientist acts as an end user of an existing visualization software package.

(2) The computational scientist develops some new visualization software that is specifically designed to aid in the analysis of the problem under study.

To act as an end user of scientific visualization software on a UNIX workstation, requires the computational scientist to be able to

(i) produce a graphics file containing the results to be viewed

(ii) operate an appropriate visualization software package, and

(iii) navigate the operating system and user interfaces of a selected workstation.

Usually, a knowledge of FORTRAN or C, access to a graphics file generation library and manuals for the selected software package and workstation are all that are required to function as an end user of software. If one is to develop custom, problem specific visualization tools, then, in addition to capabilities (i)-(iii), a knowledge of the C language and the workstation's graphics library are required. In the tutorials of Part2 of this chapter, both end user and developmental codes will be illustrated.

Each workstation vendor offers its own windowing systems and graphics library. Even though skills and codes developed on one graphics workstation usually transfer to another fairly easily, this transfer can sometimes be tedious and time consuming. Unless one requires some hardware or software feature that is vendor specific, the issue of skill and code transfer can often be avoided. X-Windows provides a "standard" environment that is common to a broad range of UNIX workstations. To the software end user, the X-Window environment is portable and robust and provides access to a large number of high quality, public domain visualization software packages. In addition, the X11 library contains functions that enable a computational scientist to manage windows and to develop custom graphical interfaces.

In many universities and laboratories there are not enough seats at UNIX graphics workstations to meet everyone's needs. For those scientists who are developing custom visualization software, a PC (Macintosh or IBM clone) equipped with at least VGA quality graphics has proven to be a valuable tool in combating a shortage of workstations. Much of the early design and "proof of concept" testing can be done on the PC. With the advent of PCs with several megabytes of memory and with increased graphics resolution, such as the SVGA at 1024x768 pixels, the modeler can now use a DOS based PC to perform many of the tasks of the UNIX workstation.

2 Data Entry and Integrity

In the past twenty years, computing power has been increasing at an exponential rate. During that period speeds of high performance computers have increased from just over one megaflops (million floating point operations per second) to almost one gigaflops (billion floating point operations per second). Computer memory has shown a similar increase: around a million words twenty years ago and now rapidly approaching a billion words. In computational science a word usually refers to a 64 bit quantity. Thus a million words is 8 megabytes (MB) and a billion words is 8 gigabytes (GB).

The increase in computing power available on our desktops has been even more dramatic. Twenty years ago, virtually all scientific computation was performed on a mainframe. Then, the standard mode of operation was batch and card decks were more common than computer terminals. Today, the computing power of an average PC exceeds that of most mainframes of twenty years ago. Moreover, the computing power of the PC has increased by an order of magnitude about every four years since its introduction in 1982.

Another recent addition to the desktop is the graphics workstation. Today, graphics workstations running UNIX offer five to ten times the performance of a typical PC. Like the PC, graphics workstations have exhibited impressive gains in price performance with a doubling of performance for the same price almost every year. The line between UNIX workstations and PCs is, however, becoming somewhat blurred. For several years, Macintosh computers have been capable of running the operating system AUX, which is Apple's version of UNIX. For some time, commercial UNIX operating systems have been available for IBM compatible PCs.

The advent of more powerful PC CPUs, such as the Intel 486, and the availability of a public domain UNIX operating system (LINUX) for this class of machine, signal a closing of the gap between the UNIX workstation class of machine and the increasingly powerful new breeds of PCs. The computational advances over the past twenty years have not been restricted to just the hardware arena. The increase in computational efficiency of many numerical analysis algorithms has been as impressive as increases in hardware computational efficiency. The combined effect of improvements in hardware and algorithms over the past twenty years is roughly a million fold increase in overall computational efficiency. Put another way, many computations that required weeks to perform in 1970 now require only seconds.

These impressive gains in hardware and algorithms have enabled scientists to numerically model physical phenomena that only a few years ago were beyond the limits of our computational resources. In the area of water resources research, for example, transient, three dimensional groundwater-contaminant models incorporating both saturated and partially saturated flow and multiple contaminants involving millions of variables over many thousands of time steps are today considered leading edge, but fairly routine.

From this increase in computing power and improved algorithms, a new generation of data hungry models has emerged. For example, in a contaminant transport model of a three dimensional nonhomogeneous aquifer involving a million finite difference blocks, the task of setting values of conductivity, specific yield, transmissivity, initial heads and initial concentrations presents the modeler with upwards of 15 million data values to set—a daunting task indeed! If the region to be modeled requires an irregular finite element or finite difference mesh, the data entry problem is compounded. If the model is to be calibrated against some observations, the data entry loop will have to be performed many times.

In many cases, a graphics screen provides the most efficient form of data entry. A scientist can manipulate graphics objects on the computer screen to create or modify data sets. Assume that a correspondence is to be made between the numerical value of some model parameter and the color of a pixel on a computer screen. The number of pixels on a workstation is roughly a million. Visualization software permits a modeler to "pan" the screen over even larger data sets or to "stack" data sets, one screen behind another. In this fashion a computational scientist is capable of setting many millions of data values by simply "painting" the screen. For example, with just one pass around the screen drawing the boundary of a section of the aquifer, the modeler can specify which finite difference cells lie within the model region and which cells are exterior. Similarly, by simply using a paint program, the modeler can assign colors to hydraulic properties and quickly characterize a million point planar region of the model. Section 5 of this chapter includes examples of such object oriented data base manipulation. Finally, before the model is run, the modeler can query the data base in a graphical manner. This often permits the modeler to spot errors or inconsistencies in massive data sets that would be difficult to inspect by any nongraphical method.

3 Code Debugging and Performance Analysis

In the traditional modeling exercise, once the modeler begins model execution the situation is pretty much one of "wait and see." Other than possibly a few diagnostics that the modeler may have chosen to print to the screen, the code continues to crunch to completion. By displaying a color map of some section of the model region, say after each time step, the modeler can gain important information about the behavior of the model. Then, if the simulation has appeared to go wrong, the execution can be halted, thus saving valuable computer and modeler time. Also, it is possible to steer a calculation and even do some intermediate recalibration. Additionally, the graphical information displayed to the screen can often yield valuable debugging information. By looking at an evolving raster map during the simulation, the modeler can find the cause of a problem such as a computational instability or a problem with a boundary condition.

Another area in which visualization has great potential is that of code performance analysis and tuning. A modeler can obtain a good understanding of the relative importance of various subroutines by monitoring a subroutine histogram during the execution of the program. The total area under such a histogram represents the total execution time, and the area under each subroutine bar gives a measure of the percentage of time spent executing it. In complicated programs the flow of logic can be displayed by a network diagram that shows the various subroutines as nodes on the network and during execution displays vectors from the calling program to the called program, with the active node highlighted. This kind of information is especially valuable when working on advanced architectures, such as a massively parallel machine on which the goal is to balance the computational load across a large number of processors.

4 Interpretation and Presentation of Results

Graphical interfaces and modern visualization methods provide a vehicle to help modelers and decision makers assimilate the massive data sets produced by leading edge numerical models. The human brain possesses a "narrow bandwidth" for processing raw numbers, but a surprisingly "wide bandwidth" for processing visual data. However, visualization is more than just a method for increasing the rate at which we absorb data. Visualization facilitates the discovery of new science by revealing hidden structures and behaviors in model output. As R. W. Hamming so aptly expresses it, "The purpose of computing is insight, not numbers." It is in the areas of insight and understanding that visualization plays a central role.

Scientific visualization provides an effective means for the presentation and communication of results of numerical models. Creative uses of color, texture and translucency allow subtle aspects of complicated, large scale models to be displayed in a form that can be easily understood. For example, two dimensional gridded field data that might be typically displayed as line drawn contours, can be visualized in a number of different ways, each offering special advantages over traditional contouring methods. The most common method for viewing the 2-D data is to produce a raster image of the data. In this case, the spatial dimensions of the model data are represented by the computer screen and the third dimension is represented by different colored pixels. Many visualization software packages are capable of accepting such (x, y, z) data sets and producing a color raster map output. One additional tool in most of these packages is the ability to interactively manipulate the color palette. This allows the researcher to make changes in the data representation akin to changing the contour interval in an interactive and nearly instantaneous manner. This activity often exposes important structures and trends in data sets.

Visualization is especially useful in allowing modelers to interrogate, in great detail, the data sets produced by their models. Three dimensional, time dependent data sets are particularly difficult to digest. The computational resources required to produce such data sets are often very large. In research mode, a computational scientist will frequently have to recompute a set of results, using different input parameters, initial conditions and boundary conditions to expose certain details or features in the mathematical model. Visualization helps the modeler ensure that all of the detail available from a particular set of computational results is exposed before undertaking the often expensive task of recomputing.

Visualization tools are available for displaying two dimensional field data of the form $(x, y, f(x, y))$ as traditional contour maps, color coded raster maps, and as a three dimensional surface, $z = f(x, y)$. Traditional graphics tools have allowed for this type of data presentation for a long time. Modern visualization tools add some new capabilities for this type of data display. Computer rendering techniques that are used to prepare photo-realistic images of scene descriptions are now in wide use for architecture and engineering. Textures can be applied to the surface renderings to enhance the perception of the three dimensional nature of the surface. Tools are now available to allow the overlay of the color contour data described above onto a surface plot of a second parameter, allowing the simultaneous display of two spatial variables and two field values.

Three dimensional field data of the form $(x, y, z, f(x, y, z))$ can be rendered in a variety of ways. Typically, the entire field of spatial values and the function value at each node can be assimilated by the software tool and a variety of output forms can be chosen interactively by the user. Examples of the output choices are color raster maps of one or many slices along a plane through the 3-D field, three dimensional dices where color contours are rendered on each surface of the subregion, and isosurface renderings where the surface formed by a particular isovalue is displayed. The isosurface displays are again enhanced by ray-tracing and other techniques to make the three dimensional shape clear to the observer. Even more sophisticated renderings of isosurfaces allow multiple isovalues to be displayed by controlling the apparent translucency of the outer isosurfaces.

5 Examples

This section discusses several visualization applications software packages that are available on the Internet. The applications discussed in some detail in this section are either in the public domain or copyrighted but free. Most of the applications discussed have an accompanying users guide that explains the use of the software in quite a bit of detail. The purpose of this section is to present very brief introductions to several packages. These introductory treatments are intended to serve as a 'fast start' mechanism for computational scientists who want to use visualization tools, not as a replacement for the various user guides.

The visualization tools introduced by way of example in this section provide the following capabilities.