Implementing Virtual Reality Interfaces for the Geosciences

Wes Bethel - *Information and Computing Sciences Division, Ernest Orlando Lawrence Berkeley National Laboratory*[1]

Janet Jacobsen - *Earth Sciences Division, Ernest Orlando Lawrence Berkeley National Laboratory*[2]

Andy Austin and **Mark Lederer** - *BP Exploration, Houston Texas, USA.*

Todd Little - *Landmark Graphics Corporation, Houston Texas, USA.*

1.0 Abstract

For the past few years, a multidisciplinary team of computer and earth scientists at Lawrence Berkeley National Laboratory has been exploring the use of advanced user interfaces, commonly called "Virtual Reality" (VR), coupled with visualization and scientific computing software. Working closely with industry, these efforts have resulted in an environment in which VR technology is coupled with existing visualization and computational tools.

VR technology may be thought of as a user interface. It is useful to think of a spectrum, ranging the gamut from command-line interfaces to completely immersive environments. In the former, one uses the keyboard to enter three- or six-dimensional parameters. In the latter, three- or six-dimensional information is provided by trackers contained either in hand-held devices or attached to the user in some fashion, e.g. attached to a head-mounted display. Rich, extensible and often complex languages are a vehicle whereby the user controls parameters to manipulate object position and location in a virtual world, but the keyboard is the obstacle in that typing is cumbersome, error-prone and typically slow. In the latter, the user can interact with these parameters by means of motor skills which are highly developed.

Two specific geoscience application areas will be highlighted. In the first, we have used VR technology to manipulate three-dimensional input parameters, such as the spatial location of injection or production wells in a reservoir simulator. In the second, we demonstrate how VR technology has been used to manipulate visualization tools, such as a tool for computing streamlines via manipulation of a "rake." The rake is presented to the user in the form of a "virtual well" icon, and provides parameters used by the streamlines algorithm.

We will discuss some issues to be considered when implementing VR and will describe our software development and user-working environments. Much of our work has been to create a software infrastructure to support VR. This infrastructure is a collection of software "building blocks" used in a visual programming environment for constructing a complete "program" for visualization. Our scientists can rapidly and easily interface several different VR input devices, such as a Spaceball or magnetic trackers, to a variety of computational or visualization tools.

Our work on interface and visualization issues led us into two technical areas which will receive elaboration in this paper. One of these is an optimized space-search algorithm which can be used by many scientific visualization tools, such as a streamlines solver. Another is our experience in making practical use of magnetic trackers as an input device.

We will conclude with a summary of the motivation for this work, and give an example of near-future extensions to this work.

---

1. Mail Stop 50F, Lawrence Berkeley National Laboratory, Berkeley CA, 94720, USA. EWBethel@lbl.gov.
2. Mail Stop 90-1116, Lawrence Berkeley National Laboratory, Berkeley CA, 94720, USA, JSJacobsen@lbl.gov.

2.0 Virtual Reality: A User Interface

The term “virtual reality” can be denoted as the conjunction of “being in essence or effect but not in fact” with “the quality or state of being real; agreement between what a thing seems to be and what it is.” The connotations of VR, however, are often associated with rather garish paraphernalia and as being useful only as entertainment [NEG93].

Despite a seemingly endless media perpetuation of entertainment-based uses of VR, as well as the public’s infatuation with “virtual worlds” and “avatars,” the basic concepts of what is referred to as VR by a wide cross-section of society are not new. The dream of computer scientists working in the field of VR dates back to the mid 1960’s with an idea put forth by Ivan Sutherland [SUT65]. Sutherland described an “Ultimate Display,” through which things look and act “real.” Decomposing Sutherland’s idea into its component pieces: “look” and “act” real, we can make the following observations:

1. “Looking real” is a function of rendering algorithms and display technology. There has been a good deal of research in the subject area of photorealistic rendering, hardware graphics accelerators, and so forth.
2. The virtual world seen through the Ultimate Display consists of geometric and procedural objects. We have witnessed a dramatic evolution in complexity of these objects beginning with the famous teapot through complex procedural objects rendered with startlingly realistic texturing and lighting effects. Furthermore, sometimes these objects are autonomous and interact with each other in unexpected and exciting ways.
3. User interface software provides an important bridge between the human and the machine.

To better understand the ways in which these components interact, consider the diagram in Figure 1. On the right-most side of the diagram, we have the elements of the traditional “graphics pipeline” - a group of objects which are then rendered into an image and presented to the user. In scientific visualization, we gain understanding about data by converting it from something abstract (e.g., velocity) into some other thing from which we can create a picture (e.g., connected piecewise vectors representing a streamline through the velocity data). The middle part of the diagram shows the relationship between the visualization processes and the traditional graphics pipeline. On the left-most side of the diagram is the user who must interact with all of these subsystems.

Focusing on the user interface, consider the many ways in which a user could specify the position and orientation of a viewer, or six dimensions of information to the renderer. One option is to ask the user to manually enter coordinates with a keyboard, a zero-dimensional device. Another option is to use a mouse, a two-dimensional device, which is better than a keyboard. Better yet, we could attach a six-dimensional device directly to the user and provide position and orientation to the renderer at 30hz. If the goal is to interactively explore a scene composed of three-dimensional objects, it is clear which option is the best choice.

A similar line of reasoning exists for display devices. On the lowest end of the scale, there are pen plotters and printers. On the highest end, there are immersive devices such as high-resolution head mounted displays as well as displays involving a configuration of multiple wall-sized stereoscopic projectors. From a practical standpoint, there is at least one order of magnitude in price difference between the “best” stereoscopic display device and the “best” six-dimensional input device.

Studies have shown [WAR96] that user comprehension of three dimensional structures increases by a factor of three with the addition of motion parallax and stereo, as compared to the comprehension level realized with just static images. Motion parallax alone accounts for more than a two-fold level of comprehension increase, while stereo viewing accounts for slightly less than a two-fold increase. Given the price difference of input and output devices, we have chosen to concentrate on input devices for practical reasons.

The scope of our work is to explore ways in which VR input devices can be effectively used for both scientific visualization and the control of a complex computational process. Work of a similar nature can be found in computational fluid dynamics [BRY91], general purpose direct manipulation interfaces for scientific visualization

[MEY93] and clinical medicine [STA96]. A complete survey of other work in which VR technology is used to perform scientific research is beyond the scope of this paper.

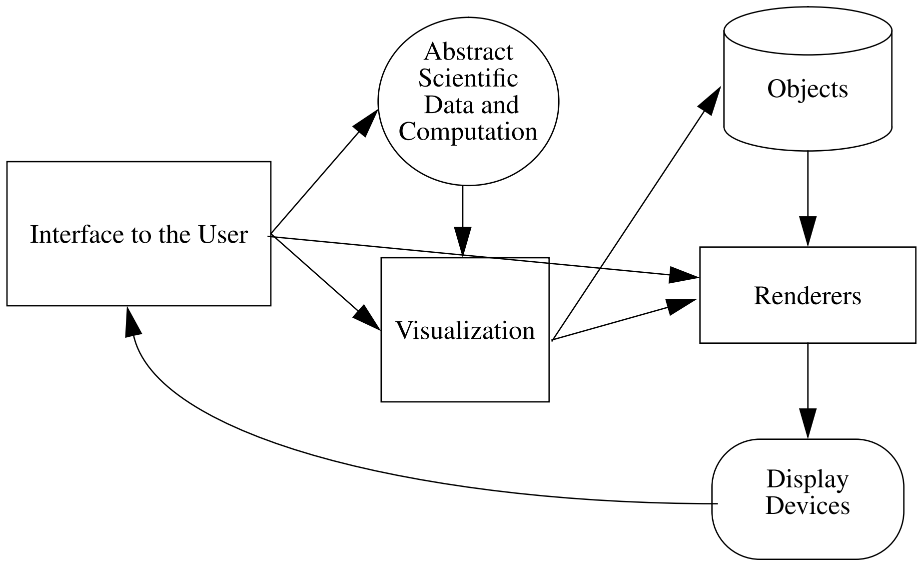

Figure 1. Relationships between User, Computational Processes (e.g, visualization) and the hardware.

3.0 VR Interface to a Numerical Simulator

Numerical reservoir simulators are used by reservoir engineers to simulate flow in petroleum reservoirs. This differs from static earth models in that the fluid distribution and compositions change with time. Understanding how the fluids move through the reservoir are paramount to designing an efficient depletion plan. Conventional reservoir simulation visualization techniques can only display the fluid distributions and compositions, and the changes in these distributions and compositions - not the path taken to achieve these changes. Flux visualization is utilized to display and understand the mechanism of these changes.

Our first usable prototype of closely coupling VR with a computational process is described in [JAC95]. The reservoir simulation, UTCHEM [DAT86], is an industry-standard code for solving multi-component flow problems. As input parameters, UTCHEM expects permeability and porosity distributions, component concentrations, miscellaneous physical and chemical parameters, together with such information about production and injection wells as spatial location, production and injection rates, perforation locations and so forth.

A typical way to use this code is to define a set of input parameters, run the code to a given simulation time, and then analyze the results. Analysis often indicates that changes in input parameters are necessary, due to errors or non-optimal results. A far more desirable way to use the code is to visualize the simulation output at intermediate time steps. This has two advantages. The first is that errors in input parameters or poorly formulated production/recovery strategies may be detected early in the simulation and the simulation can be stopped, thereby saving the user's time and compute cycles. A second advantage is that the user can see the simulation evolve. That is, the user is quickly presented with visual results and can proceed with modifying computation parameters while the results of the last run, and strategy for converging on an optimal solution, are still fresh in her mind. This cycle can be called the "visualization cycle" [MCC87].

In UTCHEM and other commercially available flow simulators, one of the key parameters to an optimal solution of a production or waterflood problem is the spatial location of producers and injectors. In order to demonstrate the feasibility of using VR at our facility for the purposes of enhancing scientific and computational productivity, we chose to implement an interactive VR facility that permits the user to quickly and easily edit the spatial location of well positions. When coupled with the capabilities of modern interactive visualization software the result is a powerful and easy-to-use system for reservoir analysis.

We consider our prototype a success, for it fulfills an important goal of a usable VR interface: the user is unencumbered from knowledge of grid coordinates, and is free to simply “grab and drag” the well from one location to another in the simulation grid.

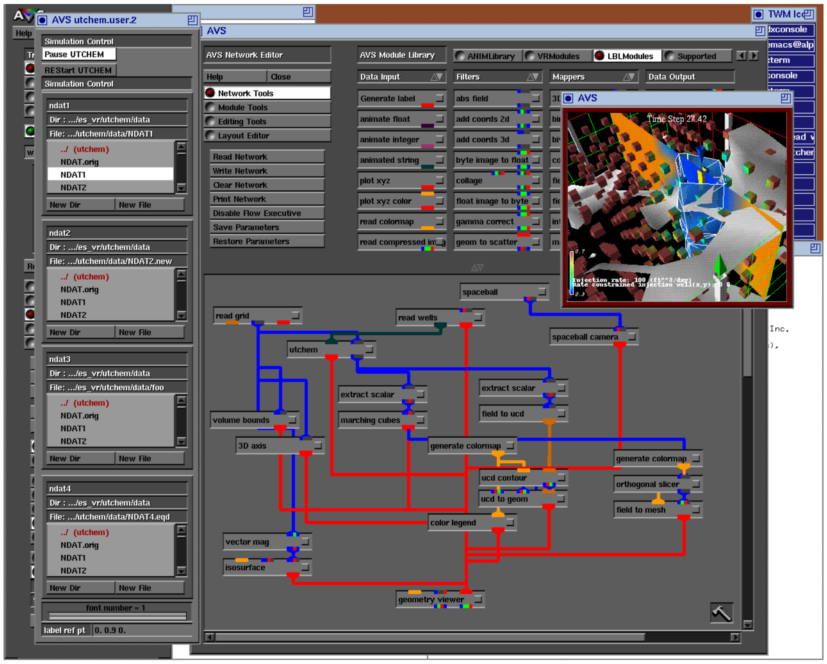

Figure 2.

AVS Network implementing a VR interface to UTCHEM.

Figure 2 shows a screendump of the AVS network used to implement the VR interface to the UTCHEM simulator. From within this network, the user may employ the VR input device to fly around within the model, or to alter the position of any of the wells (see Figures 3a and 3b). AVS is a commercially-available package in which software modules are linked together using a visual-programming interface to build a complete “program” for scientific visualization [UPS89]. UTCHEM has been integrated into the AVS environment, and its results can be visualized as they are computed, and some of its parameters may be manipulated using other VR tools which we have created for use in a visual programming environment. These tools will be described later in this paper.

4.0 VR Interface to Visualization Tools

Scientists from Lawrence Berkeley National Laboratory have been working closely with engineers from BP Exploration and Landmark Graphics Corporation on a collaborative research and development project. This project is to study visualization techniques applicable to flux visualization and to study ways to implement VR in a desktop environment useful to the reservoir engineering community.

Building upon the efforts in the aerospace industry for interacting with unsteady flow fields [BRY91], we implemented a tool for computing streamlines through a flux field (see Appendix B). The icon used for the streamlines “rake” is familiar to reservoir engineers: a well (Figure 4). The streamline is defined as a curve that is tangential to a flow field along it’s length. As a visualization tool, the streamline is useful as an aid in understanding the topology of structures within a flow field. A related algorithm called “particle advection” makes use of the same numerical solvers to compute the path of massless particles as they move through the flow. Both of these techniques have proven useful when used in interactive visualization environments.

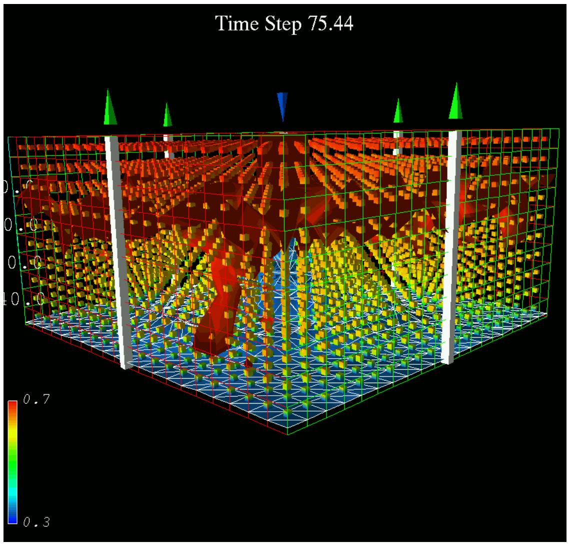

Figure 3a.

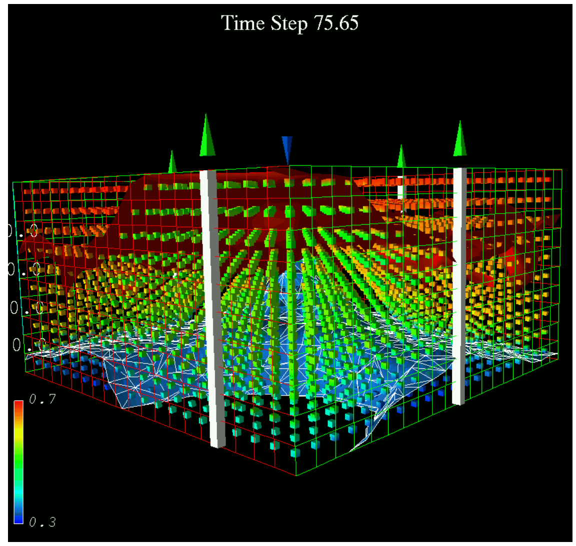

Figure 3b.

In a simulation of a waterflood, a finger of unswept oil is detected (3a) then removed by altering the location of one of the production wells and rerunning the simulation (3b).

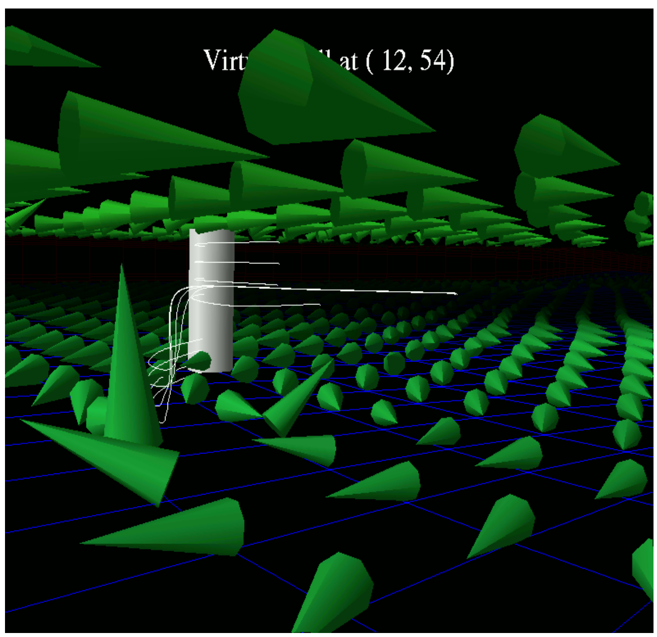

Figure 4.

Streamlines are computed using seed points that lie along the depth of a virtual well. Oil flux is visualized using a cone icon. Several simulation layers are averaged together along top and bottom layers to reduce visual clutter.

The streamlines algorithm operates by starting with a "seed point" in space, and then for some number of computational steps which is under user control, computes the location of a massless particle as it moves through the flow field. The computation is a numerical integration of velocity as the particle moves through the steady flow field. Intuitively, the streamlines show where particles would go if released into a field. Reservoir engineers also are concerned about where flow comes "from," in addition to where it goes. So, by changing the sign of the flux field and running the same streamlines code, the streamlines then show where the flow comes from. Figure 5 shows a virtual well, and the streamlines indicate flow *into* rather than *away from* the well. The markers along the streamline indicate time steps - markers close together indicate low flow magnitude along the streamline, while markers spaced farther apart indicate greater flow magnitude.

The inefficient placement of water and gas injectors not only is a waste of money, but can inhibit the efficient recovery of hydrocarbons. Reservoir simulation engineers can use these virtual wells to identify the source of extraneous water and gas influx caused by poorly placed water and gas injectors. Additionally, these virtual wells can be used to identify the source of water influx from an aquifer when the influx is via a tortuous path.

In our prototype system the interface to place the virtual well into the flux field is implemented using VR: the user can "grab and drag" the well to a new location using a six-dimensional input device. With this interface, the user is liberated from a cumbersome keyboard- or mouse-based interface.

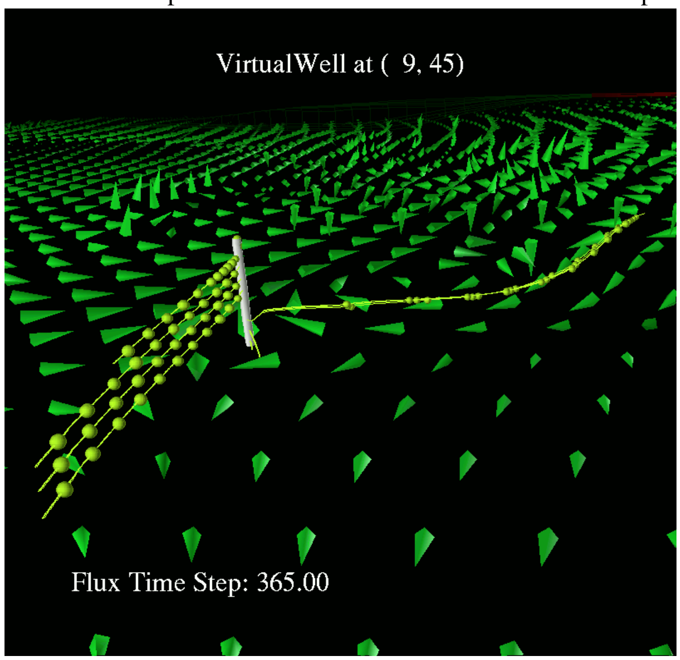

Figure 5.

"Backwards" streamlines with "markers" spaced uniformly in time along each streamline.

5.0 VRModules AVS Library

Much of the software we have developed to support desktop VR is available in source code form via anonymous ftp from `avs.ncsc.org`, or `http://avs.ncsc.org/`, and has been implemented as AVS modules. The modules themselves can be divided into three broad functional categories. For more detailed information about an earlier release of these tools, refer to [BET95].

- Device Input.

  Support is provided for two input devices, the Spaceball 2003, manufactured by Space Tec Incorporated of Lowell, Massachusetts, and the Fastrak Tracker, manufactured by Polhemus Incorporated of Colchester, New Hampshire. Events from these devices consist of floating point values representing rotation and position changes, along with button pushes. The events are encoded into an AVS data structure, which may then be subsequently used by downstream modules in the data-flow network.

- Event Filtering.

Given a stream of VR device events, two useful types of filters are numerical and logical filters. A threshold operator can be used to implement a notch-type filter. The notch filter will zero out events in which values lie outside a certain user-specified range. The tracker device in particular is very noisy, producing many small values despite the user's best effort to stand completely still. Logical filters include those that act as pass-through filters, allowing events to propagate downstream into the network only if certain conditions are met, such as the occurrence of a button-push. The logical filters are very powerful in practice and can be used in a cascade fashion to quickly prototype user interfaces in which button pushes are "programmed" to cause certain system events to occur, such as a button push being mapped to the manipulation of an object while the push of a different button is mapped to the manipulation of the user's position in space.

- Manipulation of Objects and Viewpoints.

  This class of modules provides the "glue" layer between the generic VR device event and the AVS renderer.

All of the VR application examples in this paper were constructed using this suite of freely-available AVS modules.

6.0 Input from Magnetic Trackers

In this section, we discuss the characteristics of the Fastrak tracker and some of the implementation obstacles we encountered along with solutions.

The Fastrak tracker, manufactured by Polhemus Incorporated of Colchester, Vermont, USA, is a relatively inexpensive device that uses electromagnetic fields to produce six dimensions of information: position and orientation. The technology is based on generating near-field, low-frequency magnetic field vectors from a single assembly of three co-located, stationary antennas called a transmitter, and detecting the field vectors with a single assembly of three co-located remote sensing antennas called a receiver. The sensed signals are input to a mathematical algorithm that computes the receiver's absolute position and orientation in the coordinate system of the transmitter. The current product line supports up to four receivers with a single transmitter. Furthermore, the update rates of this technology have an upper limit of 120hz per transmitter. Thus, four receivers operating simultaneously (with a single transmitter) have a combined maximum update rate of 30hz. Communication with a workstation occurs through a serial port (RS-232 or IEEE-488). In practice at our site, we have found the maximum reliable baud rate for this communication channel to be 9600 baud.

The tracker system is programmable to return different types of information. For example, the unit has built-in noise reduction filters, commands to alter the alignment frame of the transmitter, and commands to request certain types of data from the receivers. A full discussion of all available parameters is beyond the scope of this discussion. For more information, refer to [FAS93].

The device, in general, returns the position and orientation of the receiver in the coordinate system of the transmitter. These are absolute coordinates and Euler angles. As discussed previously, our suite of VR software tools operates using relative position and orientation values.

Strictly speaking, it is possible to have the Fastrak device generate relative position and rotation values. By "relative", we mean "relative to the absolute position and orientation values taken at the last sample (with respect to the coordinate system of the transmitter)."

This is accomplished in the following way:

- Initialize the device, requesting that the device generate, at each time step:

  a. relative position changes;

  b. X, Y, and Z direction cosines.

- At each time step and for each receiver:

  a. Obtain relative position changes, and the direction cosines.

b. Reset the alignment reference frame for this receiver. In effect,we reset the origin and principal axes of the transmitter to be identical to that of the receiver. Thus, on the next event, the "absolute" Euler angles and positional values, which are given in the coordinate system of the transmitter, reflect relative changes in the reference frame of the receiver, regardless of the orientation of the receiver.

The above procedure yields the correct and desired results. However, there is a problem that prohibits use of such a tactic in practice. The problem is that we are communicating with this device via a half-duplex communication channel at 9600 baud.

In the above scenario we would incur the following communication costs for each event:

1. Receipt of relative position values and direction cosines:

(3 values * 4 bytes) + 3 * (3 values * 4 bytes) + (4 bytes of overhead) = 52 bytes.

2. Send command to reset alignment reference frame:

(9 values * 4 bytes) + (4 bytes of overhead) = 40 bytes.

Assuming no communication latency and a single receiver, we can expect about ten events per second. However, we typically use two receivers, one attached to the head and one to the hand, so the throughput drops to five events per second. Such an update rate is unacceptable.

If we instead request just relative position values, along with absolute Euler angles (note that the device does not directly support relative rotation values), we incur a communication cost of:

1. Receipt of relative position and absolute Euler angles:

6 values * 4 bytes + 4 bytes of overhead = 28 bytes.

With two trackers and a 9600 baud communication channel, we can reasonably expect about 25 events per second.

Note that the values reported are with respect to the coordinate system of the transmitter. We want values which are with respect to the (changing) coordinate system of the receiver, that is, the user.

The algorithm we use for computing relative angular values in the coordinate system of the user (the receiver) given absolute Euler angles in the coordinate system of the transmitter is as follows:

For each event:

1. Construct a composite rotation matrix using the absolute Euler angles giving the orientation of the receiver in the coordinate system of the transmitter. According to the Fastrak manual, the angles are computed in the order of azimuth, elevation then roll. Call this matrix B.

2. The composite orientation matrix from the previous event, A, is inverted. For rotation matrices (orthogonal matrices), the inverse is simply the transpose of the matrix.

3. Multiply the inverse of the last event's composite rotation matrix with this event's composite rotation matrix. This matrix can be thought of a "relative" rotation matrix that represents the change in receiver orientation from one event to the next. Call this matrix the "delta matrix", X. It is computed by

$$X = A^{-1}B$$

4. Compute the relative azimuth, elevation and roll using (1) (top of next page).

5. Compute the relative translation events. Let V_d represent the delta positional changes generated by the device. The change of position in the reference frame of the receiver, V_r, is computed as: $V_r = V_d B^{-1}$

$$X = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \quad \begin{aligned} Roll &= \operatorname{acos}(abc \bullet 1, 0, 0) \\ Pitch &= \operatorname{acos}(def \bullet 0, 1, 0) \\ Yaw &= \operatorname{acos}(ghi \bullet 0, 0, 1) \end{aligned} \tag{1}$$

In general, we will not obtain the original rotation matrix delta matrix X if we construct a composite rotation matrix from the computed Yaw, Pitch and Roll values. Our experience is that the above algorithm has produced very satisfactory results despite this shortcoming. As an alternative formulation for computing both the delta matrix and the relative rotation angles, one could use quaternions [SHO85].

7.0 Point-in-Space Problem

The streamlines computation, among others, requires the solution of two serial problems. In the first stage we are given an (x,y,z) point in space, and must determine which grid block contains that point. In the second, we compute the contributions at that (x,y,z) point from the surrounding data. Typically, these values are computed using velocity values at the nodes of a cell. The various streamline velocity-integration methods, Euler, Runge-Kutta, and so forth, vary in how this latter step is performed, and have been well explored in literature [SAD95]. In our implementation, we have chosen to use a second-order Runge-Kutta interpolation method.

This section outlines an efficient algorithm for solving the point-in-space problem given a structured grid over a three-dimensional computational domain. Such a grid is common in commercial petroleum reservoir simulators. With cosmetic changes, this algorithm will work with hierarchical grids, sometimes called Local Grid Refinement (LGR).

One of two strategies is typically used to solve this type of problem. [SUB90] uses *k-d trees* for accelerating volume rendering by partitioning space in such a way as to reduce the number of voxels to be intersection-tested in a ray-tracing rendering algorithm. Alternately, computational space can be subdivided, as is often done in many image processing algorithms [BAL82].

Our strategy implements a variation of the latter, based upon an observation about structured computational grids commonly used in reservoir engineering. These grids typically consist of three computational axes. The I and J computational dimensions correspond roughly to “plan view”, while the K dimension corresponds roughly to depth. The cells are hexahedral in shape but do not necessarily share nodes. In general, there are no restrictions on the relationship between the computational domain and overall grid geometry. The algorithm described below makes the assumption that the z-coordinates vary more than the x- or y-coordinates along the K computational axis. In other words, the projection of all grid blocks of varying K and constant I and J will tend to be very similar. Despite this assumption, the algorithm will still function properly, albeit less efficiently, even if this assumption does not hold.

Phase 1. Build the space subdividing tree.

- Step 1: For each [I,J], compute the bounding hull for the “column” of cells [I,J,k=1,K].
- Step 2: Recursively subdivide the computational grid into four parts using binary partitioning in computational space. When no further partitioning is possible, copy over the bounding hull for the appropriate cell “column” computed in Step 1. While ascending back through the recursion, combine the bounding hulls from lower levels.

In Figure 6 we see, in plan view, the spatial extents of the space-subdividing tree at varying tree depths superimposed over a small grid. Leaf nodes are shown with white lines. Figure 7 shows several levels of the same tree and grid, but in three dimensions.

Phase 2: Given an (x,y,z) location, find the [i,j,k] grid block containing that point.

Our search operates in two sequential phases. The first phase consists of a tree traversal algorithm to find those leaf nodes whose extents contain the given (x,y,z) point. Note that more than one leaf node (column of cells) may contain the given (x,y,z) point. The more that the x and y coordinates vary between steps of K in each column of cells, the more "column hits" will be generated in this phase of the algorithm. In the final analysis, only one cell will contain the point, so the "column hits" must be sequentially searched (this is a conscious algorithmic design choice).

The second step in the search consists of looking at the cells in the given "column" to determine which cell contains the point. This can be done quickly using a binary search on depth.

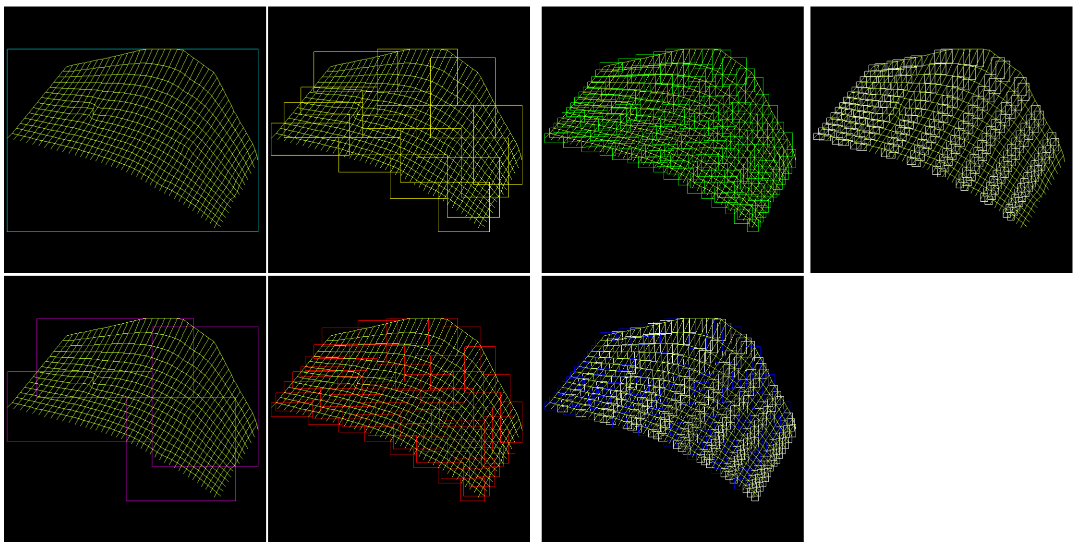

Figure 6.

The bottom-left image contains four overlapping bounding boxes which represent the spatial extents of the four regions resulting from subdividing the entire grid shown in the upper-left image. The remaining images are further refinements, with the final two images showing the extents of cell columns.

8.0 Conclusion

Virtual Reality technology provides a valuable component to an effective user interface. A primary design goal of the VR interface is to enable scientists to make more effective use of computing technology by leveraging upon their intuition through increased interactivity realized through intuitive interfaces. Many scientific problems involve some component of optimization. Even with today's near-teraflop machines, it is still unrealistic to expect a direct analytic solution for many problems. Reservoir engineers, for example, are constantly asked to formulate production strategies given changing conditions. The VR interface provides one means to simplify interacting with complex computational processes in order to quickly answer "what if" questions.

An intriguing use of this technology is in the area of well design and infill drilling. Using VR input devices the engineer or geologist could easily design a wellbore path. Then by using the streamline calculation the well's drainage area could be readily visualized. A limit of the streamline calculation is that it is based on an instantaneous steady state approximation. However, it still provides significant information to the engineer. Once a good starting point is designed, the full reservoir simulation could be restarted accounting for the new well(s), and a new simulation case could be evaluated. We are investigating visualization techniques to help identify areas in

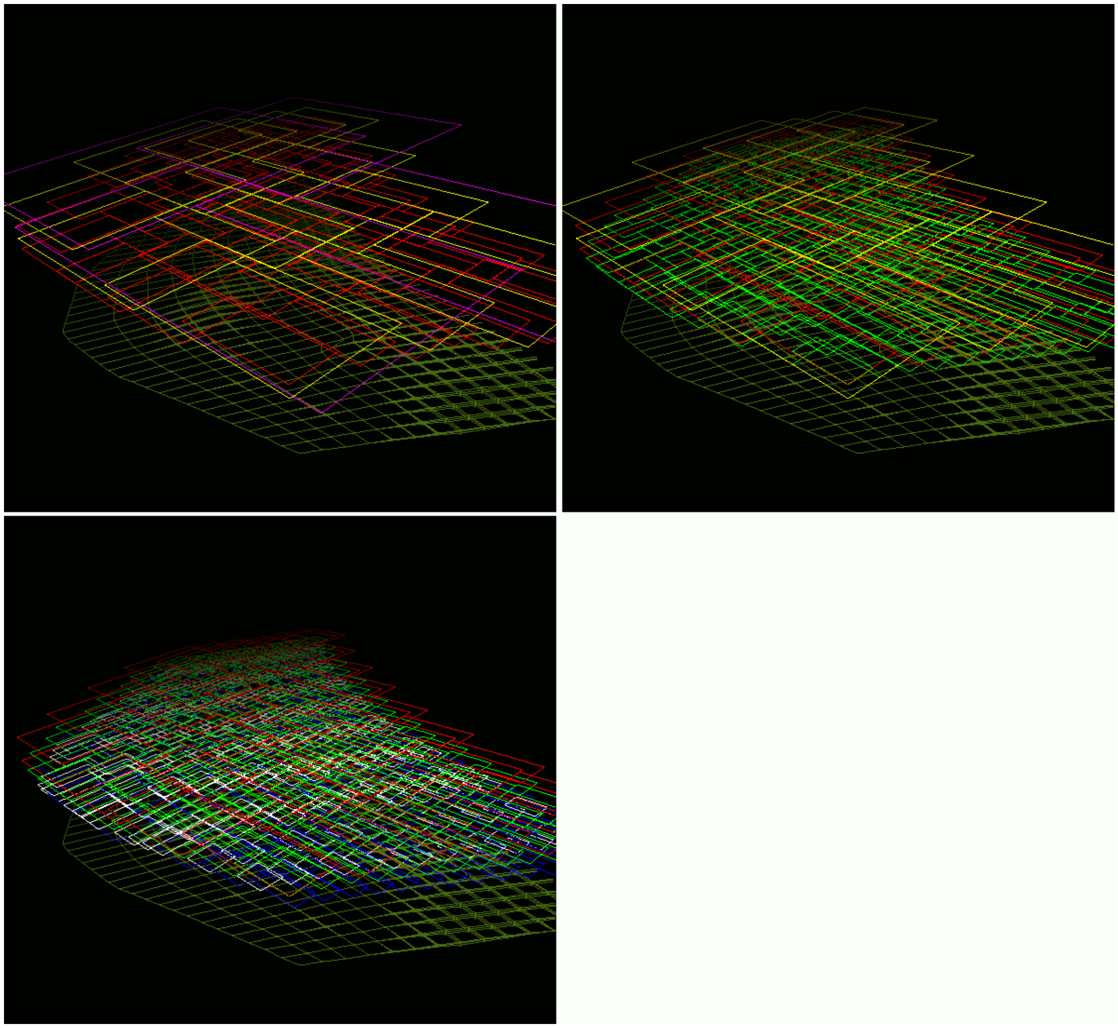

Figure 7.

Three dimensional view of reservoir and the projected [I,J] extents of several depths of the space-subdividing tree.

the reservoir with low flow and high hydrocarbon mobility in order to aid the engineer in designing recovery strategies.

We anticipate that VR interfaces will become more commonplace, beginning with desktop implementations using inexpensive hardware, not unlike the prototype systems we have been using in our facility. One of the largest obstacles faced when considering the subject of VR in general is that of expensive display devices, along with the expense of the graphics power required to compute high resolution images, in stereo, at 30 frames per second. On the other hand, the practical notion of a simple user interface for data input, has the potential to change scientific computing to the same degree which a windows-based interface changed personal computing. The underlying software infrastructure to support such an interface is still in its infancy, but will grow tremendously in the years to come.

9.0 Acknowledgment

The authors wish to thank Harvard Holmes of Lawrence Berkeley National Laboratory for his suggestions and critical comments. This work is supported by the U. S. Department of Energy, Office of Energy Research, Office of Computational and Technology Research, Laboratory Technology Research Division under contract number DE-AC03-76SF00098 between the U. S. Department of Energy and the University of California.

10.0 References

[BAL82] Ballard, D. and Brown, C., Computer Vision, Prentice-Hall, 1982.

[BET95] W. Bethel, ‘Modular Virtual Reality Visualization Tools,” LBL Technical Report Number 36693, UC 405, Lawrence Berkeley Laboratory, 1995. http://blondie.lbl.gov/publications/avs95/avs95.html.

[BRY91] S. Bryson and C. Levitt, “The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows,” in Visualization ‘91, pp17-24, 1991.

[DAT86] Datta-Gupta, A., Pope, G.A., Sephernoori, L. and Thrasher, R.L., “A Symmetric Positive Definite Formulation of a Three-dimensional Micellar/Polymer Simulator,” SPE Res. Eng, 1(6), pp. 622-632, 1986.

[FAS93] Fastrak 3Space User’s Manual, Revision F, December 1993.

[JAC95] Jacobsen, J., Bethel, E., Datta Gupta, A., and Holland, P., “Petroleum Reservoir Simulation in a Virtual Environment,” Proceedings of the 13th Symposium on Reservoir Simulation, Society of Petroleum Engineers, San Antonio, Texas, February 12-15, pp. 233-247, 1995.

[NEG93] Negromonte, N., “Virtual Reality: Oxymoron or Pleonasm?,” Wired, Volume 1, Number 6, December 1993.

[MCC87] McCormick, B., DeFanti, T. and Brown, M. (eds.), “Visualization in Scientific Computing,” Computer Graphics, Volume 21, Number 6, November 1987.

[MEY93] T. Meyer and A. Globus, “Direct Manipulation of Isosurfaces and Cutting Planes in Virtual Environments,” NASA Technical Report RNR-93-019, 21 December 1993. http://www.nas.nasa.gov/.

[SAD95] I.A. Sadarjoen, T. van Walsum, A.J.S. Hin, F.H. Post, “Particle Tracing Algorithms for 3D Curvilinear Grids,” in Proceedings of the Second Dagstuhl Seminar on Scientific Visualization, G.M. Nielson, N. Hagen, H. Mueller (eds.) IEEE Computer Society Press, Los Alamitos, in press.

[SHO85] Shoemake, K., “Animating Rotation with Quaternion Curves,” Computer Graphics, Volume 19, Number 3, Proceedings of Siggraph 1985.

[STA96] State, A., Livingston, M., Hirota, G., Garrett, W., Whitton, M., and Pisano, E. and Fuchs, H., “Technologies for Augmented-Reality: Realizing Ultrasound-Guided Needle Biopsy.” To appear in Computer Graphics Annual Conference Series, SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996).

[SUB91] Subramanian, K., and Fussell, D., “Applying Space Subdivision Techniques to Volume Rendering,” in Proceedings of Visualization ‘90, IEEE Computer Society Press, October 1990.

[SUT65] Sutherland, I., “The Ultimate Display,” in Proceedings of the IFIP Congress, 2, 1965, pp 506-508.

[UPS89] Upson, C., et. al., “The Application Visualization System: A Computational Environment for Scientific Visualization,” IEEE Computer Graphics and Applications, Volume 9, Number 4, July 1989.

[WAR96] Ware, C., and Franck, G., “Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions,” ACM Transactions on Graphics, Volume 15, Number 2, April 1996.