# A Geoscientist's Perspective on Immersive 3D Data Visualization

Author Names Withheld

Affiliations Withheld

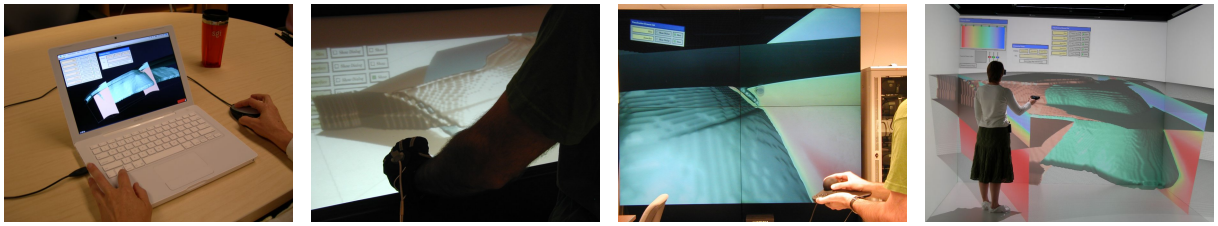

Figure 1: The Visualizer application in four different VR environments: desktop, responsive workbench, tiled display wall, and CAVE.

**ABSTRACT**

A growing need at the forefront of Earth science, and other domains, is the analysis of complex 3D volume data produced by observations or numerical models. Visualization, and especially interactive visualization in immersive VR environments, is a powerful method of exploring unknown data, and of identifying and quantifying unknown features in those data. We describe a visualization software, **Visualizer**, that was developed specifically for visual exploration in VR. Visualizer contains carefully optimized algorithms and data structures to support the high frame rates required for immersion, and the real-time feedback required for interactivity. As an application developed for VR from the ground up, Visualizer realizes benefits that usually can not be achieved by software initially developed for the desktop and later ported to VR. However, can also be used on desktop systems with a similar level of real-time interactivity, bridging the "software gap" between desktop and VR that has been an obstacle for the adoption of VR methods by the sciences.

To evaluate the effectiveness of Visualizer in both its native VR environment and on the desktop, we performed a user study comparing it to a widely used desktop visualization application. Our study's results show that scientists can receive substantial benefits from using VR methods for their research, and that carefully designed VR applications can be as effective on desktop systems as native desktop applications.

**CR Categories:** I.3.7 [Three-Dimensional Graphics and Realism]: Virtual Reality; J.2 [Physical Sciences and Engineering]: Earth and atmospheric sciences

**Keywords:** virtual reality, 3D data visualization, immersive visualization, interactive exploration

## 1 INTRODUCTION

Geoscientists work with diverse data sets ranging in spatial scales from nanometers to thousands of kilometers and varying on time scales from femtoseconds to billions of years. Typically, these observations or numerical models are 3D volume data sets: seismic tomography images of the Earth's interior, finite element models of plate tectonics and mantle convection, and neutron imaging of microbial communities preserved in ancient rocks. Analyzing and interpreting these large volumetric data sets with information at multiple scales poses a significant challenge in geoscience research – one that can be addressed by the scientific visualization community, particularly through innovative use of interactive and immersive virtual reality (VR) environments [13, 16, 12].

A substantial component of research in the geosciences involves identifying the most important processes in natural systems and developing computational models of key interactions. One efficient way to identify unknown processes is to look for correlations within and among data sets. Quantitative evaluation of correlations requires the scientist to have a detailed conceptual model of the underlying relationships; however, the fundamental processes are poorly constrained for many of the problems at the forefront of research. Thus, correlations can be extremely difficult to predict and extract mathematically. In these cases, relationships among data are best identified by examining data visually in a flexible, interactive environment. Such visual examination of data can lead to the conceptual framework necessary to develop quantitative methods for further analysis.

As a specific example of how visualization can aid geoscience research, which will be referred to again later, consider a fluid dynamics simulation for the deformation of a tectonic plate subducting into the Earth's mantle. Iterative solutions to finite element models of a subducting tectonic plate are strongly dependent on the smoothness of the viscosity and temperature field defining the plate and plate boundaries. The shapes of these structures are constrained by geological and geophysical observations and constitute complex 3D volumes. One of the challenges of developing new computational models of this kind is understanding why the numerical model fails to converge to a solution. Graphical representations of model results, in space and time, can quickly lead to identification of aliasing and other discontinuities in model data, which may violate both smoothness requirements and model fidelity to the geological structure.

Although we evaluate the usefulness of immersive visualization from a geoscientist's point of view, and our user study used a geoscience data set, we believe that our approaches and observations apply as well to other scientific domains where complex 3D data sets are analyzed on a regular basis.

**Visualizer: 3D Volume Visualization Software.** It has long been known that graphical representations of complicated data sets on 2D displays provide efficient and insightful ways of inter-

preting quantitative data [18], but similar analysis of 3D data sets has lagged behind. One reason may be that the design of 3D volume visualization software has traditionally focused on providing an environment for users to create a final image of a data set that is effective at communicating ideas and results [2, 3, 1]. This type of visualization software usually facilitates enhancing the appearance of structure of interests, or synthesizing various data types, e. g., isosurface of temperature, color-mapped slices of viscosity or streamtubes of fluid flow in a numerical fluid dynamics model. However, this design objective is focused on visualizing known structures, i. e., the value of the best iso-surface to display is already known, but does not support exploring a data set in which the features of interest are yet to be discovered. Exploring a data set visually in order to identify features or processes of scientific interest requires an interactive environment and real-time visualizations that can be modified on-the-fly.

The 3D volume visualization software presented here, called Visualizer, has been specifically designed for highly interactive 3D VR environments and therefore follows different design principles than software that was originally developed for use in a 2D desktop environment [4]. For example: 1) navigating (picking up, rotating and translating a slice or isosurface) in a VR environment is simply done by moving a tracked input device, such as a wand, while pushing a button, and 2) all menus and dialog boxes are dynamic, i. e., they appear when and where the user presses a button on an input device, and 3) creating isosurfaces or slices occurs in real time as an input device is moved through a virtual data set. We present the underlying development philosophy and volume visualization implementation for Visualizer, which was developed through a collaboration of computer scientists and geoscientists, and present a user study aimed at assessing how well Visualizer aids scientists in analyzing 3D volume data. Although our study did not yield conclusive quantitative data, the subjective evaluations by the study participants indicate that the design principles built into Visualizer create an environment in which interacting with data is intuitive and data exploration is both effective and efficient.

## 2 Related Work

To put Visualizer into context, and compare it to other 3D visualization software for desktop and immersive environments, we first discuss several desktop applications, and then several applications developed for VR.

Tecplot [2] is a commercial visualization software for 3D volume data in a desktop environment, and is widely used in the Earth sciences. Its main goal is the production of publication-quality graphs and figures, but it also contains several features that make it applicable to visual data analysis. Its main volume *visualization elements* are color-mapped slices and isosurfaces, and the program allows one to change the set of elements interactively. However, interaction is limited to entering desired isovalues into dialog boxes or dragging axis-aligned slices through a data set using sliders in a dialog box. The program's response to changes is not in real time: especially changing an isovalue causes delays of tens of seconds before the display is updated. We found that directly observing the changing shape of an isosurface under varying isovalues is a very powerful analysis tool; the fact that Tecplot does not support this style of exploration is a major limitation. Additionally, Tecplot's navigation methods are limited when a user wants to explore a small feature in a larger data set in detail. Although Tecplot provides the usual virtual trackball interface, it can only rotate the data around its centroid, not around arbitrary 3D points. This, and the fact that Tecplot reduces the resolution of surfaces during navigation, severely limit detail analysis. Tecplot does contain a measurement tool to query the spatial position of and data values at arbitrary locations, but it is not intuitively clear how Tecplot translates a 2D mouse position into 3D space for measurement.

Vis5D [3] is an open-source visualization software aimed at time-varying, multivariate 3D volume data. It is often used in the Earth sciences, especially in atmospheric science. Its main goal is the production of figures and animations. Vis5D's main visualization elements are color-mapped slices and isosurfaces, but it also supports direct volume rendering. The level of interactivity of Vis5D is similar to Tecplot's, with the same limitations for visual data analysis, but Vis5D contains some improvements: slices can be dragged by direct manipulation with the mouse, and the virtual trackball for navigation always rotates around the screen center, improving the user's ability to examine small features in detail. Vis5D's volume rendering feature uses a simple slice-based algorithm, and is due to its long rendering times not applicable to interactive exploration.

CAVE5D [1] is a direct port of Vis5D to immersive environments based on the CAVE library [9]. It runs in VR environments compatible with CAVELib, and uses a CAVE wand to control the visualization. Even though CAVE5D was introduced in late 1995, it is still used for Earth science visualization in immersive environments, especially CAVEs. One reason might be that it allows scientists to visualize data on the desktop first using Vis5D, and then to import the visualizations into a CAVE.

The development of CAVE5D from Vis5D is a good example of the challenges posed by porting desktop software to immersive environments. The main benefits of VR, intuitive navigation and direct manipulation of 3D objects, are not realized because the original desktop program does not contain functionality to support them. For example, Vis5D allows a user to drag a slice by manipulating it with the mouse, and CAVE5D uses the same mechanism, but based on a 6-DOF input device. Instead of just moving the device to a position of interest and pressing a button to create a slice at that position (or drag an existing slice), the user has to aim the device at an "interaction box" at the corner of the slice to drag the slice along its axis. This makes it quite difficult to change a slice while zoomed-in to examine a feature, and is not the most appropriate way of using a 6-DOF input device to manipulate a 3D object. Interestingly, isosurfaces are still changed by numerically entering a desired isovalue; however, since VR environments have no keyboards, users have to use the wand to enter numbers via a virtual 3D keypad. This style of interaction is actually less effective in a VR environment than on a desktop. Navigation also does not take full advantage of interaction using a 6-DOF input device: rotating the wand causes the data set to rotate, but not around the current position of the wand. Instead, the model rotates either around its centroid or the user's head position, depending on the navigation mode. Both navigation modes are hard to get used to, and even experienced users sometimes have problems to move a model in the desired way. As a result, CAVE5D is mostly used to present previously created visualizations in a more impressive environment, and not to create or analyze visualizations by interactive exploration.

The NASA virtual wind tunnel [6, 15, 5] is an even older application than CAVE5D, but it was directly developed for immersive environments and takes into account the particular benefits and constraints of VR. Its main purpose, as the name implies, is the analysis of computational fluid dynamics data, but it could be used for other 3D volume data as well. Its main visualization elements are streamlines/streaklines, particle traces, color-mapped slices, and isosurfaces. As opposed to CAVE5D, all visualization algorithms are optimized for direct 3D manipulation and real-time feedback. For example, streamlines are created by directly selecting their starting point in 3D space, and isosurfaces are created by growing them from a selected seed point in space, instead of specifying their isovalue. Isosurfaces are based on time-outs, i. e., the result of creating a surface will be visible in the display in less than 0.1 s, enabling direct observation of an isosurface's change as the seed point is

dragged. Navigation is also intuitive: users can "grab space" using a 6-DOF input device, and then reposition the data set by moving/rotating the input device. Overall, the virtual wind tunnel is an effective visual analysis application. Its main limitations are that it only supports a single grid format and that it is only portable to a very limited range of VR environments.

## 3 Interactive Exploration of 3D Volume Data

In the development of the Visualizer software, we followed many of the design principles first exhibited by the NASA virtual wind tunnel [6] and later described in more detail in [17]. The two main constraints of VR are the high frame rates upwards of 30 Hz required for head-tracked stereo viewing, and real-time response to any user interaction within 1/10 s [5, 11]. These constraints influence the implementation of all visualization algorithms, whose "standard implementations" typically do not observe them. A standard Marching Cubes [14] implementation, for example, might require several seconds or minutes to generate millions of triangles for larger data sets. An interactive and immersive implementation of this algorithm must ensure that the display does not "freeze" during that time, that at least intermediate results are presented to the user after at most 1/10 s, and that the generated triangle set can be rendered in less than 1/30 s. These goals require multithreaded programming, special algorithms such as seeded isosurfaces [15, 11], advanced rendering using compressed geometry or multiresolution methods [10], and careful optimization of the extraction and rendering algorithms.

The main benefits of VR are the direct 3D perception enabled by head-tracked stereoscopic displays and the ease with which users can select positions in 3D space using 6-DOF input devices. To fully exploit these benefits, visualization software has to follow a direct manipulation approach across the entire range of functionality, from navigation over creation of visualization elements to quantitative analysis. If users see a feature of interest in a data set, they must be able to quickly create appropriate visualization elements to explore the feature in more detail; it is not appropriate to first have to measure the position of or the data value at the feature, and then enter those numbers into text fields to create slices or isosurfaces. Instead, VR software should allow users to create elements by "point-and-click."

An additional important design goal was the ability to run the visualization application effectively on a wide range of VR environments with different sets of input devices, including desktop environments with only a mouse and keyboard. Our experience has shown that scientists are reluctant to use VR software because it forces them to use a (shared) VR environment for all their analytical work. Being able to use the same VR software on the desktop first for preview generation and initial quality assessment, and then only perform important and detailed explorations in VR, should alleviate these concerns and lead to a wider acceptance of VR methods in scientific domains. To achieve maximum portability, we developed Visualizer on top of the Vrui VR development toolkit [8]. Vrui supports highly interactive and high-performance VR applications that are completely independent of the underlying VR environment, including the type and number of available input devices. Our user study, discussed in Section 5, shows that Visualizer is very effective in a CAVE, and slightly more effective than a native desktop application in a desktop environment.

Finally, we designed Visualizer such that it can be applied to a wide range of data formats. The differences between data formats such as regular (Cartesian) grids, hexahedral (curvilinear) grids, simplical (tetrahedral) grids, and heterogeneous finite-element meshes are so fundamental that each normally requires its own implementation of all visualization algorithms. As VR software must be carefully optimized to satisfy VR's real-time constraints, a software architecture should allow easy experimentation with different algorithms and data structures. For example, developers might have to change the representation of isosurfaces from individual triangles to indexed triangles or triangle strips to evaluate which performs best on a given system. If there are separate implementations of the underlying algorithms for different data formats, applying such changes while keeping all versions working together is a major software engineering challenge. As a result, most visualization programs, especially VR visualization programs, support only a single data format. Visualizer is based on a separation between data formats and algorithms that allows one to develop visualization algorithms only once, and apply them to all supported data formats. In fact, Visualizer only contains a single piece of code implementing its isosurface extraction algorithm, and this code is applied to regular, hexahedral, and tetrahedral grids. This data format and algorithm abstraction uses the C++ template mechanism to create highly efficient code that performs on par with, and sometimes out-performs, other code developed directly for a specific data format.

## 4 System Architecture

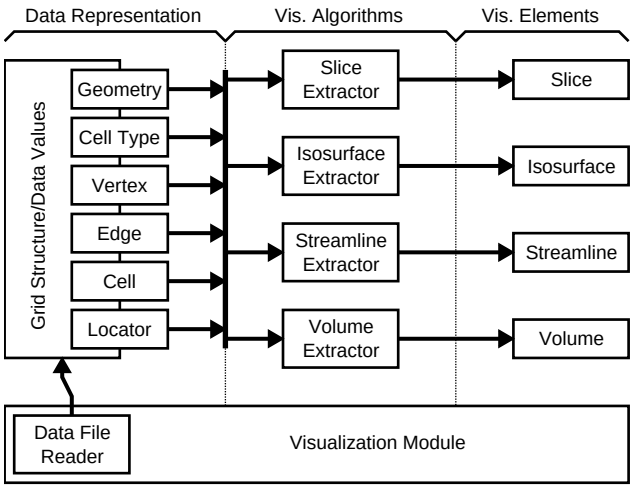


Figure 2: Visualizer's system architecture. Modules are plug-ins encapsulating the data structures and algorithms to visualize a particular data format, and are created by linking concrete instantiations of the underlying generic components.

The Visualizer software was designed with maximum modularity in mind, as as toolbox of generic interacting components, each encapsulating a particular functionality. The lower-level components of this architecture have been described previously [7], but have since been completely redesigned for higher performance, and the higher-level components and the module system have been added. The overall architecture of the new component toolbox is shown in Figure 2. Components fall into three basic categories: *Data representation*, *visualization algorithms*, and *visualization elements*. Visualizer uses the C++ template mechanism to combine concrete instantiations of all components required to visualize a particular data format, and a *data file reader* required to load individual data sets, into a *visualization module*. These visualization modules are packaged as external plug-ins, and loaded into the overall Visualizer application at run-time when a user requests loading a data file. A module advertises to the overall application the scalar and vector variables contained in the current data set, and the visualization algorithms that can be performed on it. Visualizer then creates a graphical user interface to allow users to select

variables and algorithms and assign them to input device buttons. The result of executing a visualization algorithm on a data set is a visualization element, e. g., a color-mapped slice or an isosurface. Elements are stored in a scene graph inside the Visualizer application and can be toggled on/off and deleted individually.

The C++ template mechanism is very powerful for creating component architectures. As opposed to run-time polymorphism, where descendants of the same base class can only differ in the implementation of virtual functions, templates allow one to additionally use different data types, supporting more powerful abstractions. Furthermore, templates are instantiated and linked at compile-time, allowing the compiler to perform full optimization on the generated code. As a result, generic code often performs as well as specialized code, and sometimes better than specialized C code due to the compiler's ability to optimize across function calls. To combine the benefits of high performance and run-time polymorphism, our module architecture links sets of closely interacting components at compile-time into larger polymorphic modules that only loosely interact with the overall application.

### 4.1 Data Representation

The core component of all visualization modules is the representation of the visualized data set. Visualizer currently supports regular (Cartesian), curvilinear (hexahedral) and unstructured (tetrahedral) grid structures. The interface between data representations and visualization algorithms is implemented as a set of utility classes giving access to the underlying geometry of a data set, i. e., the dimension and scalar type of its domain space, the type of its cells, currently simplices or n-dimensional cuboids, and the vertices, edges, and cells defining the data values and grid structure. A final accessor class, *Locator*, is a spatial iterator that makes it possible to query data values and local grid structure at arbitrary positions inside the data set's domain. Depending on the grid structure, data representation components contain acceleration structures to query the neighborhood relationships between cells, and to support the locator interface.

### 4.2 Visualization Algorithms

Visualization algorithms create visualization elements such as slices and isosurfaces based on the grid structure and data values of a data set, and the position/orientation of a Locator. Although Visualizer contains "global" algorithms such as isosurfaces specified by isovalue and slices specified by position and orientation, its user interface focuses on direct manipulation, i. e., the creation of elements based only on a point/orientation of interest. Visualizer currently supports color-mapped slices, isosurfaces, streamlines, and volume rendering. In accordance with the direct manipulation approach, and to provide more meaningful immediate feedback to users, most algorithms are *seeded* implementations. As illustrated in Figure 3, a seeded algorithm does not create visualization elements by processing an entire data set cell-by-cell, but instead starts element creation from the cell containing the point of interest, and traverses all other cells containing the same element radially outwards. This means that any intermediate results created by seeded algorithms provide local information in an area around the point of interest, and allow a user to explore a region of a data set by moving the point of interest while observing the change of the element's shape as it is dragged along. Once the user stops dragging, the partial element is created to its full extent, or to the maximum number of graphics primitives the display system can render in its alloted frame time.

The main benefit of a generic component architecture is that algorithms can be expressed independently of grid format. For example, Visualizer contains only a single implementation of isosurfaces, which is applied to all grid formats. By using the interfaces

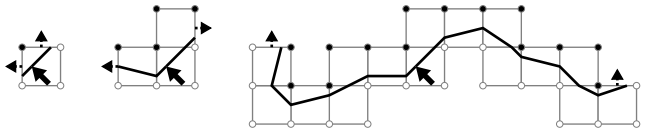

Figure 3: Creation of a seeded isosurface. Extraction starts with the cell containing the Locator. After each cell is processed, the algorithm determines into which neighboring cells the isosurface extends, and adds those to the queue of pending cells. Processing the queue in order causes the isosurface to grow outwards from the cell containing the Locator. The black and white dots denote grid vertices whose value is above and below the isovalue (the interpolated data value at the Locator's position), respectively.

described in the previous section, the isosurface algorithm only contains the logic of how to create isosurface fragments inside a single cell based on vertex values, how to traverse all cells containing the isosurface, and how to use timers to satisfy real-time constraints. Any additional required information, such as triangulation case tables, cell neighborhood information, and the formulas used to interpolate vertex positions/data values along cell edges, are provided by the data representation interface classes.

Another benefit is the ability to provide specialized implementations of components. For example, there are many different ways to volume-render 3D data, and some of the highest-performance ones only work on particular data types. In these cases it is possible to provide special-case components, and the C++ compiler will use them when possible. Visualizer's generic volume rendering algorithm is based on blending color-mapped slices; the specialization for regular (Cartesian) grids uses hardware-assisted volume rendering based on 3D textures to achieve frame rates high enough for immersive visualization.

### 4.3 Visualization Elements

Visualization elements are produced by visualization algorithms, and stored in a scene graph managed by the overall Visualizer application. Element components share lower-level implementations, such as triangulated surfaces optimized for high-performance rendering, and provide an interface for algorithms to create those in a streaming fashion. This additional separation of algorithms and their resulting data allows developers to optimize them separately. For example, changing the initial implementation of isosurfaces from unordered triangle sets to indexed triangle sets increased rendering performance substantially, and only required changing a single type definition inside the Isosurface component, and no changes in the Isosurface Extractor component. Once the superior performance of indexed triangle sets was established, we changed the Isosurface Extractor to generate indexed triangles internally, increasing extraction speed by a large factor. We expect that implementing even higher-performance surface representations will be not much more difficult.

### 4.4 Visualization Modules

A visualization module ties together all components required to visualize a particular data format, and a file reader to load concrete data sets from external storage. The actual code of a visualization module is usually very short. It only contains "glue code," i. e., type definitions to describe the internal structure of the data and the required components, and the code to read grid structures and data values from an input file and store them in the data representation.

The module concept is the incorporation of an approach to data handling that differs from many other visualization applications. Many applications, including Tecplot and Vis5D, define a "native"

data format, and users have to convert their data to this data format in a pre-processing step. Although these conversions are generally simple, having to keep several versions of the same data in several formats wastes storage space, and conversions can lose precision, especially since most interchange formats are plain ASCII tables. More importantly, conversion means that it becomes impossible or inefficient to directly stream intermediate results from a running simulation code into a visualization application to monitor the process of an ongoing simulation, and potentially even manipulate simulation state on-the-fly. Our ultimate goal for Visualizer is to have it used in such a context, hence we decided not to enforce native data formats for each basic grid structure, but to give programmers the ability to make the simulation's data format native to Visualizer by coding a visualization module. From a user's point of view, Visualizer does not have one native data format, but as many as there are visualization modules. Our approach is also different from providing file reader plug-ins that convert data file formats into an internal format: all data representation, visualization algorithm, and visualization element components related to a particular data format are compiled specifically for the data format, giving the compiler the option to optimize across component boundaries. We believe that our approach yields higher extraction and rendering performance.

### 4.5 Overall System Architecture

The Visualizer application itself is responsible for managing all loaded visualization module plug-ins, all loaded data sets, the graphical user interface that lets users select variables and algorithms, and the scene graph of created visualization elements. Another important component is the DataLocator module responsible for translating input device interactions to extracting visualization elements, and for displaying intermediate extraction results for real-time feedback. Additionally, Visualizer contains modules to add interactive clipping planes to a visualization, to measure the position of and data value at arbitrary locations, and to edit the color maps applied to all data variables individually.

### 4.6 Tying It All Together

Judging by the descriptions provided in this section, it might seem that Visualizer is more of a programming toolkit for visualization software than an actual application aimed at end users. And this impression is partially true; our experience shows that VR visualization requires fine-tuned algorithms and data structures that sometimes depend on the particular data set to be visualized, and sometimes a particular scientific question requires custom analysis tools that need to be implemented at a low level to perform in real time. Visualizer's component toolkit enables programmers to quickly add such custom algorithms, and experiment until their optimal implementation is found. Finally, reading a particular data set requires writing a data file reader, and the "glue" code that binds all required components into a visualization module.

However, from a user's point of view, Visualizer is a complete application for visual exploration. If users happen to use a data format that is already supported by a visualization module, they can read it directly without the need for any scripting or programming; if there is no module, they can initially convert their data to a supported format in the same way they previously did for other visualization software, or find a programmer to write a custom module for their data.

## 5 Evaluation

### 5.1 Visualizer Performance

As Visualizer is designed for immersive VR environments, it is important to ask whether its implementation satisfies the VR real-time constraints. To evaluate this, we measured its performance on the data set shown in Figure 4, on a desktop PC with a 2.4 GHz AMD Athlon 64 X2 CPU, 1 GB of RAM, and an Nvidia Geforce 7800GS graphics card. The test data set is defined on a curvilinear hexahedral grid of $271 \times 81 \times 201$ vertices, with two variables (temperature and viscosity) given for each vertex. The data set is stored in the native ASCII format written by the used simulation code, and occupies 353.5 MB of disk space.

On loading the data set, Visualizer needed 12.4 s to parse the input file, convert all vertex positions from spherical to Cartesian coordinates, calculate the decadic logarithm of the viscosity values (viscosities are best visualized logarithmically), and create a kd-tree containing all cell centers that is later used to quickly find the cell containing a given position. Most of this time is spent parsing the ASCII input file (creating the kd-tree of 4.4 million vertices takes about 3 s); storing input data as binary files reduces load times substantially. Afterwards, we measured how long it takes to extract the isosurface shown in Figure 4, b), using smooth shading with vertex normal vectors computed as data value gradients. Creating a seeded isosurface from the center point of the feature shown in Figure 4, c), took 304 ms and generated 339,722 triangles. For comparison, creating a global isosurface of the same isovalue took 744 ms and generated 339,854 triangles (the isosurface has a small disconnected component not extracted by the seeded algorithm). After extraction, Visualizer was able to render either isosurface at a frame rate of 146.8 frames/second, or 49.9 million triangles per second. During dragging, the extraction algorithm was able to create about 100,000 triangles before it had to stop and display the intermediate result due to the 0.1 s time-out; in other words, it was able to visualize a large region of the isosurface around the point of interest in real time.

Performance is similar in immersive environments. Startup in our CAVE takes a few seconds longer since Visualizer itself and the input data set have to be replicated to all six cluster nodes, but isosurface extraction times are about the same. The rendering performance in the CAVE is about a factor of two lower, since all surfaces have to be rendered twice in each frame (once for the left eye and once for the right eye).

### 5.2 Effectiveness of Interactive Visual Exploration

Throughout the development of Visualizer the goal has been to create a flexible, easy-to-use and intuitive environment for interactive, scientific analysis of 3D volume data sets. However, the verification of whether the software achieves this goal must come from the scientists using the software. To this end, we have conducted a user study aimed at assessing how well Visualizer aids researchers in evaluating unknown data sets, by comparing it to Tecplot [2], described in Section 2, a software package commonly used by geoscientists. The study was driven by two main questions:

1. Does enabling a high-level of interactivity in a volume visualization system aid scientists in efficiently and effectively analyzing data?

2. Does the enhanced interactivity and immersion provided by a 3D environment play an important role in scientific data exploration?

In the first part of the study both visualization systems were compared in their native environments (Visualizer in a CAVE and Tecplot on a desktop computer). In the second part of the study, Visualizer was also used on a desktop computer and the results were compared to the those from the CAVE; Tecplot does not run in VR environments.

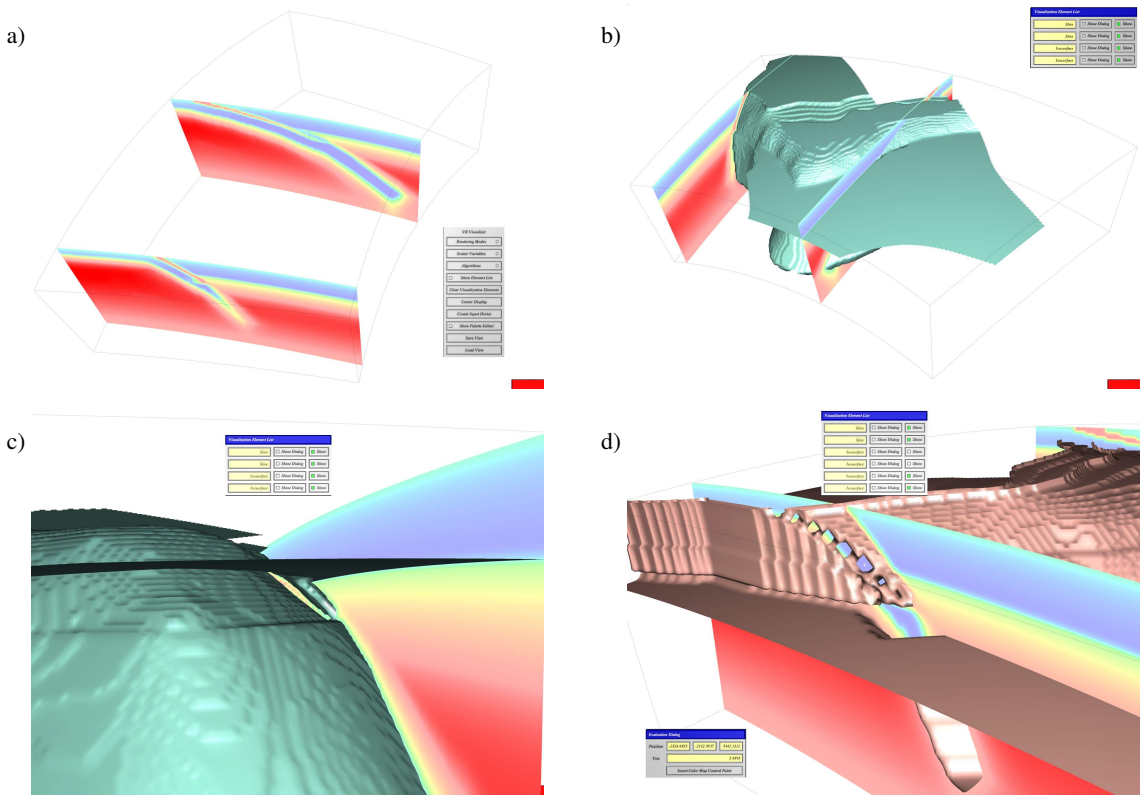


Figure 4: Snapshots from target data set exploration. a) Color-mapped slices used to locate initial problem features. b) Isosurface providing a 3D view of subducting tectonic plate structure. c) Close-up view of "problem feature" protruding from slab surface. d) Close-up view of a second problem feature: holes in the isosurface.

**User Study Format.** In each software/environment setting, the study participants were asked to explore a data set and to identify features that did not meet specified criteria: 1) features must be smooth and continuous, without steps or faceted surfaces, and 2) features must be continuous, without holes or protruding structures. The data used in all parts of the study, shown in Figure 4, was an actual finite element (FE) model input structure of a subducting tectonic plate, that had been used in a deformation calculation, but suffered from grid-aliasing problems and mis-aligned fields that caused holes in and protrusion from the tectonic plate surface. These problems were only discovered after the calculation had failed to converge to a solution. The tasks performed by the study participants were based on the same steps used by geoscientists in analyzing the appropriateness of a model input data set for a FE model calculation.

Before beginning the prescribed tasks, the study participants were taught how to use the software in each environment using an idealized input data structure. Afterwards, the participants were then given 30 minutes to explore the target data set and instructed to write down the coordinates and data value for features they found that did not meet the specified criteria. After completing the analysis, the participants were asked about the tools within the software that they found most and least useful for navigating through the data set and identifying features. They were also asked to note any tasks that they found to be particularly easy or difficult to complete. Finally, after completing the tasks in all three software/environment settings, the participants rated the software on a scale of easy to difficult for the tasks of navigating, identifying features, locating features, learning the software, and overall use.

The study included 19 participants – eleven graduate students, five under-graduate students, and three faculty – engaged in geoscience study and/or research. Of the 19 participants, two had minimal previous experience using a CAVE and one had previously used Tecplot.

**User Study Results.** The results of the user study, shown in Figure 5, demonstrate that the high level of interactivity provided by Visualizer in the CAVE makes analysis of 3D volume data both easy and effective. Participant responses indicate that not only is data exploration (navigating, identifying and locating features) easier to do in the CAVE than using Tecplot, Visualizer is also easier to learn how to use and easier to use overall. In addition, on average more features were located using Visualizer in the CAVE than using Tecplot or Visualizer on the desktop, although this part of the results is not conclusive and demands further investigation. The participants also found that data exploration is easier on the desktop using Visualizer than using Tecplot, but they also found it more difficult to learn how to use, which probably affected both their overall impression of the software and ability to locate features.

Participant feed-back on the individual software provided additional information on why Visualizer is easy to learn and use in the CAVE and more difficult to learn on the desktop. First, nine participants stated that they did not find any of the tasks to be difficult in the CAVE and found it particularly easy to identify and locate features. As one user reported, "the Visualizer-CAVE was by far the easiest to work with. All the features stood out very clearly and it was incredibly easy to navigate." In contrast, 13 participants stated that identifying and locating features in Tecplot was the most difficult task.

Second, 13 users stated that assigning tools in Visualizer on the desktop was the most difficult task. In any environment, tool assignment in Visualizer requires first choosing what one would like the program to do (e. g., create isosurfaces) from the main menu and

then assigning this action to a button by selecting a locator tool from the tool selection menu using the desired button. On the desktop, however, two things make this process more cumbersome. First, buttons on the mouse are used to navigate (e. g., left button for rotating and right button for translating the data) and they are also used to create color-mapped slices or isosurfaces by using a modifier key on the keyboard (this caused confusion for new users who had to remember the button-key assignments). Second, the tool selection process on the desktop includes an extra step of creating a "virtual input device" to map 2D mouse positions into 3D model coordinates. User responses indicate that remembering the button-key combinations made tool assignment and use more difficult; however, they also stated that, once this was overcome, Visualizer was better for identifying and locating features. As one user stated, "the greatest difficulty is probably the initial complexity of click and key combination, but even in 30 minutes I became pretty efficient with what I learned and I can see getting used to it very quickly." Based on the responses and suggestions of the study participants, we plan to add an optional, fixed "tool bar" for tool assignment that can be used to introduce new users to Visualizer, but can be turned off once button-key combinations, which are faster to use, are learned.

The user study ratings and responses show that the interactivity provided by using Visualizer in a CAVE is intuitive and allows users to focus their attention on exploring the data set. In addition, they indicate that while interacting with a 3D data set in a 2D environment can be made effective in an interactive visualization system, it requires practice, while an immersive 3D VR environment provides immediate benefits by allowing the user to interact with data in a natural and intuitive manner.

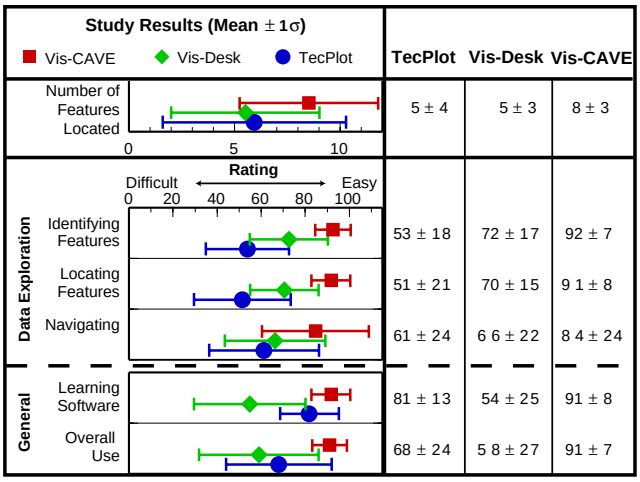


Figure 5: User study results. Rating of data exploration (navigating, identifying and locating features) and general use (learning software and overall usability) displayed as the mean and standard deviation. Results show that Visualizer used in the CAVE made data exploration very easy and the software was easy to learn and use overall. Visualizer used on a desktop was also better for data exploration than Tecplot, but was more difficult to learn since the tool assignment process is somewhat cumbersome on the desktop.

**Not Just a Pretty Picture.** Although our user study demonstrates that Visualizer can aid scientists in exploring and analyzing 3D volume data, the question may still remain: will Visualizer actually aid geoscientists in doing real research? As stated earlier, the data set used in the user study is a real model input for an FE model of deformation of a subducting tectonic plate. Before the model calculation was run this data set had been checked in a standard way, by using color-mapped slices in a 2D desktop visualization system, and deemed satisfactory to use as input. Problems in the data set were only discovered after the calculation had failed to converge to a solution. The data set was then viewed in a CAVE using Visualizer, which revealed a range of grid-aliasing problems and unintended holes in and protrusions from the tectonic plate. Therefore, unlike many non-interactive visualization systems, which are viewed as providing merely a "pretty picture," but little in terms of scientific understanding, Visualizer has already demonstrated its utility in facilitating scientific analysis and discovery.

## 6 Conclusions and Future Work

To address a common problem in Earth science, the analysis of 3D volume data, we have developed a 3D visualization software, Visualizer, in a collaboration of computer and Earth scientists. In contrast to many other visualization tools, whose primary goal is to create polished images to communicate already known results, Visualizer is a highly interactive immersive application aimed at the intuitive exploration of unknown data and the identification and quantification of newly discovered features in the data. Interactive exploration is a natural application of immersive visualization technology, and adds to the value of visualization: observing changes in a 3D display in response to user actions yields more insight into the underlying data than simply observing the 3D display alone.

We compare Visualizer to other existing desktop and VR visualization software. Many scientists are sceptical of VR, in part due to a perceived lack of benefits of using VR stemming from the exposure to non-native software. Programs initially developed for the desktop and then ported to VR cannot fully utilize that richer environment, because desktop systems lack the interactivity of VR environments, and hence desktop programs do not contain the functionality needed to exploit it. Because Visualizer was directly designed for use in immersive environments, it exploits the full capabilities of VR.

As an application aimed at immersive VR environments, Visualizer has to satisfy two real-time contraints: it has to maintain a high frame rate upwards of 30 Hz to support immersion, and it has to provide real-time feedback to any user interaction within about 1/10 s to enable interactivity. We described how these constraints influenced the software architecture of Visualizer and the implementation of the offered visualization algorithms. Since we started applying Visualizer to more problems from Earth science and other domains, we have learned that its design as a "programmer's toolkit" allows us to implement additional problem-specific analysis methods that would have been very hard to realize using other, more user-centric, software.

To evaluate the effectiveness of Visualizer compared to the widely used Tecplot software, and to evaluate the effectiveness of immersive VR environments compared to regular desktop systems, we performed a user study involving geoscience students and faculty. The study shows that immersive environments, when combined with highly interactive applications, offer substantial benefits over desktop systems, and that software originally developed for immersive VR can be as effective on the desktop as "native" desktop software. We briefly described how the Vrui VR development toolkit [8] supports creating such portable applications, and thus addresses a second common complaint about using VR for scientific purposes: the separation between desktop and VR software forces scientists to either perform all their analysis tasks in expensive or shared VR environments, or to use two different programs on the desktop and in VR. This separation, and the perceived lack of benefits of using VR, leads most scientists to forego the use of VR methods for their research.

Our user study also provided valuable feedback on how further to improve Visualizer. We learned that effective exploration requires

"transparent" user interfaces, such as intuitive navigation and "point and click" creation of visualization elements, and we will continue developing Visualizer, and the Vrui toolkit itself, to provide a more efficient, and easier to learn, user interface especially on the desktop. We are continuously improving Visualizer by developing more visualization algorithms, and by optimizing the existing algorithms and elements for faster creation and rendering. Another, separate, goal is to create communications modules to couple Visualizer with remote on-going simulation processes, to be able to visualize the progress of a simulation in real-time, and potentially make changes to the simulation's current state to steer it in a desired direction.

## REFERENCES


[1] CAVE5D. http://www-unix.mcs.anl.gov/~mickelso/CAVE2.0.html.

[2] Tecplot. http://www.tecplot.com.

[3] Vis5D. http://www.ssec.wisc.edu/~billh/vis5d.html.

[4] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 2004.

[5] S. Bryson. Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71, May 1996.

[6] S. Bryson and C. Levit. The Virtual Windtunnel: An environment for the exploration of three-dimensional unsteady flows. In *Proc. of Visualization '91*, pages 17–24, Los Alamitos, CA, 1991. IEEE Computer Society Press.

[7] Co-author. Previous paper about low-level component architecture, 2001.

[8] Co-author. Environment-independent VR development. Parallel submission to IEEE VR 2007, 2007.

[9] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proc. of SIGGRAPH '93*, pages 135–142, Anaheim, CA, 1993. ACM Press.

[10] Klaus Engel, Rüdiger Westermann, and Thomas Ertl. Isosurface extraction techniques for web-based volume visualization. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 139–146, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[11] O. Kreylos, E. W. Bethel, T. J. Ligocki, and B. Hamann. Virtual-reality based interactive exploration of multiresolution data. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical Approximation and Geometrical Methods for Scientific Visualization*, pages 205–224. Springer-Verlag, Heidelberg, Germany, 2001.

[12] Oliver Kreylos, Tony Bernardin, Magali I. Billen, Eric S. Cowgill, Ryan D. Gold, Bernd Hamann, Margarete Jadamec, Louise Kellogg, Oliver G. Staadt, and Dawn Y. Sumner. Enabling scientic worksflows in virtual reality. In *Procòf the ACM SIGGRAPH Internaional Conference on Virtual Reality Continuum and Its Applications (VRCIA) 2006*. ACM SIGGRAPH, ACM SIGGRAPH, 2006.

[13] Ching-Rong Lin and R. Bowen Loftin. Application of virtual reality in the interpretation of geoscience data. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 187–194, New York, NY, 1998. ACM Press.

[14] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. of SIGGRAPH '87*, pages 163–169. ACM, 1987.

[15] T. Meyer and A. Globus. Direct manipulation of isosurfaces and cutting planes in virtual environments. Technical Report CS–93–54, Brown University, Providence, RI, 1993.

[16] Theresa Marie Rhyne and Alan MacEachren. Visualizing geospatial data. In *SIGGRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, page 31, New York, NY, 2004. ACM Press.

[17] William R. Sherman and Alan B. Craig. *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[18] Edward Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.