Editor: Theresa-Marie Rhyne

The Top Five Problems that Motivated My Work

Bill Hibbard

University of Wisconsin—Madison

Theresa-Marie Rhyne has lately been organizing efforts for a number of researchers to produce lists of top visualization problems, and I am flattered to be included. A recent Visualization Viewpoints column (see the July/August 2004 *IEEE CG&A*) featured Chris Johnson's excellent list, and now it's my turn. Since the May 1999 VisFiles column[1] already describes my list of the top ten problems that will drive future visualization work, this Visualization Viewpoints column is a look back at the top five problems that drove my own work developing Vis5D, Cave5D, and VisAD (see the "Vis5D, Cave5D, and VisAD" sidebar). Some of these problems are high-minded and some are grubby and gritty.

This is the mix of problems that I learned from Verner Suomi, the founder of the Space Science and Engineering Center where I do my visualization work. Suomi was literally the inventor of weather satellites. Among many other technical accomplishments, he held the patent on the spin-scan radiometer, which was the basis for the geostationary satellites that until recently produced the cloud animations we see on TV weather shows. He also lobbied the US congress for the funds to develop these satellites and was heavily involved in building the prototypes. He was a visionary, a politician, and what my father would have called a "dirty-fingered engineer." I hope these problems reflect this combination of concerns that lead to useful change. I start at number five, and count down to the most important problem.

Vis5D, Cave5D, and VisAD

Vis5D is widely used for visualizing the output of weather models and other time varying, multivariate, 3D gridded data. It was probably the first open source visualization system.

Cave5D is an adaptation of Vis5D to the CAVE and ImmersaDesk VR systems. It's probably the most popular VR visualization software. Both Vis5D and Cave5D have served as the starting points for numerous other software developments.

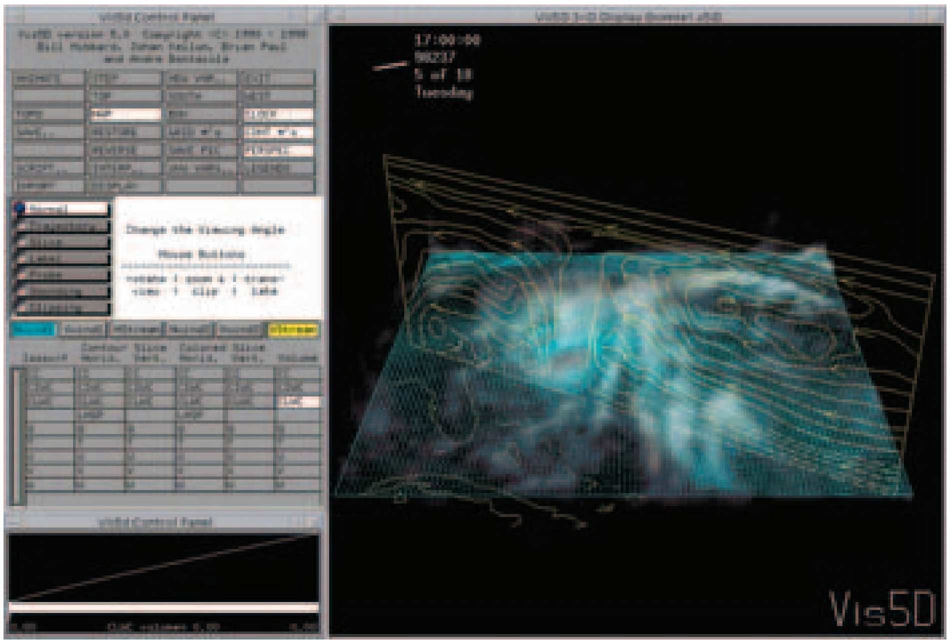
VisAD is a Java component library for interactive and collaborative visualization and analysis of numerical data. It's the basis for numerous domain-specific visualization systems. For more information about these systems, visit http://www.ssec.wisc.edu/~billh/vis.html.

Problem 5: Big data

Scientists have too much data to understand it as printed numbers, or even as simple 2D graphs. Every major weather modeling center has a large room whose walls are covered with hundreds of printed maps and charts showing the observations used to initialize their model and the data grids that come out of their model. When I started working on visualization in the late 1970s, some meteorologists were building 3D Plexiglas models depicting weather scenarios. Clearly they were anxious for a way to combine all their different vertical levels and different atmospheric fields into unified 3D pictures.

During the 1980s, visualization researchers strove to meet this need by developing techniques and software for making 3D depictions of data. But single 3D images lacked a vital feature of the 3D Plexiglas models: the ability to interactively rotate the viewpoint. In 1988, commercially available workstations appeared that could interactively rotate shaded 3D graphics. This produced a flurry of interactive 3D visualization systems including my project's Vis5D, NASA's FAST, Stellar's (originally called StellarVision) AVS, and others. These systems let scientists view their large data sets (or at least what were considered large data sets in 1989) and interactively change the data subsets they were seeing, and to rotate the 3D scene via a mouse.

Of course, the definition of *big data* is relative. I've worked closely with the European Centre for Medium-Range Weather Forecasts (ECMWF) since 1988, and I've always cautioned them that data sets produced on their modeling system, usually among the ten fastest computers in the world, could not be interactively explored in a single Vis5D session running on a normal desktop workstation. We put a lot of effort into optimizing memory use in Vis5D, including strategies for moving data between memory and disk in response to user interactions. And a few years ago, 3D graphics performance took a great leap forward with the popularity of 3D computer games, which helped visualization systems catch up a little with the large data sets produced by supercomputers. But none of this can really close the data capacity gap between supercomputers and workstations. Figure 1 is a scene from a Vis5D visualization of a 1998 ECMWF simulation of Hurricane Bonnie, showing winds on horizontal and vertical planes and a volume rendering of cloud liquid water, over a limited geographical region clipped out of the global simulation, and animating over a subset of the thousands of time steps produced by the simulation. For a large 1998 Silicon Graphics workstation this was a pretty big data set, and yet it's only a fraction of the complete model run.



1 **Image of an ECMWF Hurricane Bonnie simulation generated using Vis5D.**

One approach to big data is to combine numerous commodity display screens into wall displays with tens of millions of pixels. I tend to be skeptical of such expensive solutions, as they show more detail than the mind can grasp in one scene. Visual attention is focused in a fairly small region of the retina, so user needs can be more economically met by systems that enable users to interactively zoom out for an overview and zoom in quickly to see detail in specific regions. For applications that can justify spending large resources on visualization, I prefer putting the parallelism on the back end with the data, enabling users to interactively explore larger data spaces.

Problem 4: It's the response time, stupid

Inspired by Bill Clinton's 1992 motto "It's the economy, stupid," I put a little sign over my desk that said "It's the response time, stupid." For one thing, a high frame rate is necessary for creating the visual illusion of smooth motion for interactive rotation and animation. Perhaps just as important, smooth motion and fast responses to all kinds of interactions are emotionally satisfying to users and greatly enhance their enthusiasm for software. Every idle moment while users are waiting for a response is a moment when they are thinking that their software could be better.

Users are engaged in a thought process and visualization is merely a tool to help them think. Every wait for a response interrupts their flow of thoughts. For example, you can still get pads of maps with locations of weather stations printed on them. Meteorologists write observations, such as temperatures, next to each station and then manually draw isolines. This interactive drawing process helps them think about the distribution of temperatures and understand the overall pattern. Ideally, our visualization tools can help with similar thought processes, and with as little delay as a pencil.

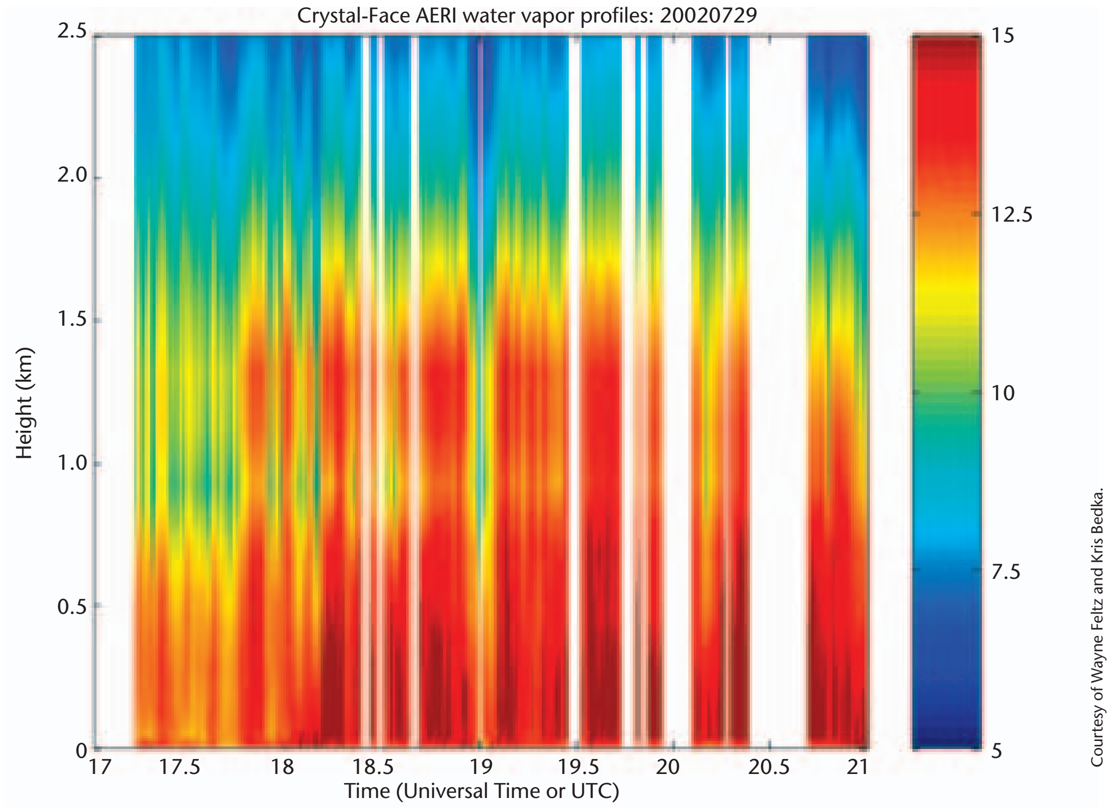
We took this problem seriously in the development of Vis5D, which probably has the fastest response times of any major 3D visualization system. This has partly been a matter of efficient algorithms and coding. But there are also natural tradeoffs in system designs. In the design of Vis5D we gave the big data and response time problems priority over the next two problems on my list: pretty pictures and abstraction.

Software evolution is guided by problems like the five discussed in this article, but you could also say that the evolution is driven by technological change. Such change is generally not gradual, but rather comes in the form of precipitating events. For 3D visualization, the major precipitating event was the ability of hardware to render 3D shaded scenes at reasonable animation frame rates. Our focus on response time in Vis5D reflected an understanding that the evolution of visualization software was being driven by this leap forward in hardware rendering performance.

Problem 3: Pretty pictures

A wonderful old friend in the US National Weather Service, Hank Schmidt, once told me I produced "Monty Python weather graphics." As a big Python fan I took it as a compliment, but the reality is I have always been a true believer in interactivity and fast responses and so regarded some users' desires for pretty pictures as a necessary evil. You could say that I was much more interested in OpenGL than Adobe PostScript. Nevertheless, image quality can be a show stopper for some organizations, especially when they need to produce images for publication.

Image quality has several dimensions. One important factor is including informative labels and rendering them using good fonts. Scientists also like cleanly rendered axis scales, and labeled color wedges embedded in images for variables depicted by pseudocolor. Information density should be neither cluttered nor sparse. In systems that support interactive zoom, this means progressively adjusting detail as users zoom. Also, fonts should remain at constant screen size with zoom. In many cases this requires moving text and other



2 **Publication-quality image generated by Kris Bedka and Wayne Feltz using Matlab.**

graphical elements so as to leave reasonable gaps between text and surrounding graphics, as font sizes change. We have devoted considerable effort in VisAD to the issues of axis scales, labels, fonts, and dynamic changes with zoom. Of course, antialiasing is another important quality factor that is fortunately built into OpenGL.

In addition to all these rendering details, scientists want to maintain high quality throughout the printing process. This largely translates into the ability of systems to output images as PostScript or PDF files, rather than in raster file formats like TIFF and JPEG. This is a real challenge for visualization systems based on OpenGL or similar 3D libraries. Figure 2 shows an image produced as an encapsulated PostScript file using MathWorks Matlab. Our Vis5D and VisAD systems can only produce raster images embedded in PostScript, which has always been a problem for some users. The GL2PS library, started in 2000, provides a better option for producing PostScript files from OpenGL programs.

I have to confess that among the top five problems that drove my work, pretty pictures has had the least personal appeal for me although I couldn't ignore it completely. The only interesting part of the problem was making image contents change in response to user interactions with viewpoints.

Problem 2: Abstraction

Before I got interested in visualization I was interested in mathematics, so I've put a lot of effort into the problem of abstraction. In fact, this is the only problem that I've included both in this list and in my earlier list of top ten problems that will drive future visualization work. An abstraction provides a framework for thinking about visualization and classifying the options available.

One early abstraction for visualization and computer graphics in general was the data flow model of programming. This modeled the flow of information from raw data to final rendered images, through a series of information transformation processes such as resampling, isosurface generation, projection from 3D to 2D, clipping, and so on. One nice feature of this abstraction was its natural interface to visual programming, where users could design visualization processes by drawing networks of primitive processes. Data flow visual programming was the basis for a number of successful systems, including AVS and IBM Data Explorer. I resisted the pressure to adopt this abstraction as the basis for my own systems, for reasons explained in the August 2000 VisFiles column.[2]

I was much more enthusiastic about data modeling abstractions. These abstractions have long been the focus of database research, progressing through the hierarchical, network, and relational data models. While scientific visualization data models primarily resemble the hierarchical data model, they are distinguished by their focus on continuous data spaces and sampling topologies. Some early visualization efforts suffered from the lack of information for locating data in time and space, and systematic data models provided an abstraction for designing file formats and visualization systems that overcame these problems.

Our VisAD system was designed around a comprehensive data model, and introduced another abstraction for the visualization process factored into a set of mappings from primitive data variables to primitive display variables. Figure 3 shows a user-interface component from the VisAD SpreadSheet based on this abstraction. A data schema for a time sequence of earth-navigated satellite images is shown in the top two text areas, the list of primitive data variables is shown in the "Map from" area, the list of primitive display variables is shown in the "Map to" area, and a possible set of mappings is listed in the "Current maps" area.

The VisAD design was also driven by a major precipitating technology event: the physical connection of almost all computers in the world via the Internet. This event drove the software evolutions of distributed object technology and Sun Microsystems Java, which as far as possible unifies the syntax and semantics of local and remote objects (if the network is the computer, then Java is the language designed for programming that computer). Distributed objects constitute another important abstraction for visualization and other computer applications, and are fundamental to the VisAD design.

The final problem listed in Chris Johnson's recent Visualization Viewpoints column is a call for a theory of visualization. One approach to such a theory is an abstraction, defining various kinds of mathematical structures on sets of data and sets of depictions, and treating visualization as a mapping between these sets that preserves mathematical structure.[3,4] That is, to view visualization as a sort of isomorphism between a data space and a depiction space. Some examples of mathematical structures include:

- equality (requiring that visualization be a one-to-one mapping from data to depictions),
- topology (requiring that visualization be a continuous mapping),
- metric (requiring that visualization mappings preserve distance),
- lattice (these are lattices of whole data spaces, rather than, say, a lattice of samples in one data grid), and
- vector (requiring that visualization be a linear mapping).

While abstractions are interesting in their own right, they can also be used as the basis for flexible and general systems. A good abstraction defines a framework that can accommodate new and unpredicted needs, giving systems longer, useful lifetimes.

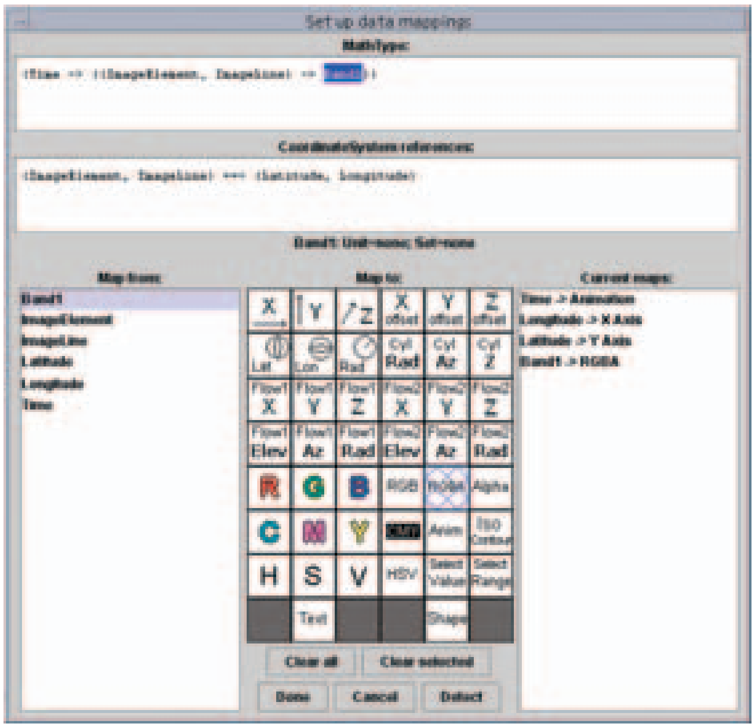
Problem 1: Find funding

This may sound cynical, but the visualization researchers I know spend as much time and effort on this problem as any. My strategy for solving it was to build large communities of individual and institutional users of my systems, who could lobby funding agencies on my behalf or provide funds directly. So I'll rephrase this problem as Problem 1: Find users.

Problem 1 redux: Find users

Solving this problem is the real test of the utility of visualization techniques and systems, and ignoring this problem runs the risk of wasting your time and resources. The first problem in Chris Johnson's list was a call for visualization researchers to become more engaged with their users. I heartily agree.

The first step in solving this problem is being open to


Set up data mappings

MathType:

(Time -> ((ImageElement, ImageLine) -> Band1))

CoordinateSystem references:

(ImageElement, ImageLine) <-> (Latitude, Longitude)

Band1: Unit=none; Set=none

Map from:

Band1
ImageElement
ImageLine
Latitude
Longitude
Time

Current maps:

Time -> Animation
Longitude -> X Axis
Latitude -> Y Axis
Band1 -> RGBA

3 **VisAD SpreadSheet user interface component for specifying mappings from primitive data variables to primitive display variables.**

users wherever they come from. It's tough when you have to entice specific people to use your work, since they may be among the vast majority who don't want to go to the effort to learn new tools. But there's usually someone out there who is looking for something new, and they can lead the way for the rest. The second step is to have open communications with your users, which means taking their suggestions seriously and constructively, and letting them know what you can and cannot do for them. The first version of VisAD, written in C, had only a couple users and they didn't like it much. So I thought long and hard about their complaints and was open to suggestions when we redesigned the system in Java. Similarly, Vis5D benefited greatly by users' suggestions. Many were against my instincts but turned out to be right in practice.

One great thing about open source is that your user communities also become your developer communities. Enthusiastic users are always looking for ways to make the system better, and with open source they often do it themselves. Furthermore, many institutions won't give you funding but they are happy to provide their own programmers to help with development. In fact, nothing seems to motivate institutions better than letting them take over development of successful systems. All three of my systems, Vis5D, Cave5D, and VisAD, are now in the care of others. Vis5D is being developed as the Vis5d+ system on SourceForge, and as D3D by the National Oceanic and Atmospheric Administration's Forecast Systems Lab. Cave5D was first taken over by Old Dominion University, and now by Argonne National Lab. VisAD development is now focused in the Integrated Data Viewer system at the Unidata Program Center, the

AIFS system at the Australian Bureau of Meteorology, the new Man–Computer Interactive Data Access System (McIDAS) V version at the University of Wisconsin, and other systems. The use of McIDAS is especially gratifying, since my systems grew out of McIDAS in the early 1980s.

There was an interesting difference between the development of the Vis5D and VisAD user communities. Large institutions resisted using Vis5D in the early days, in many cases feeling they could develop their own equivalent. So the user community started with individuals. Institutions signed on after they learned it wasn't so easy to rewrite the system (many of these efforts were based on data flow systems, which have a hard time replicating certain features of Vis5D). However, the VisAD community included some important institutions early on. I suspect this was based on the credibility gained from Vis5D. It takes awhile for these systems to mature to the point where they are really useful, and our institutional partners in VisAD development have experienced some pain waiting for that maturity.

Conclusion

It's the combination of different problems that makes research, or life, interesting. The really hard problem is to create a system based on general abstractions that provides fast response times and pretty pictures with big data sets, and appeals to users and funding agencies.

Many readers will want to think about what's left to accomplish on these problems, or what other problems will motivate the future of visualization. My own thoughts were laid out in the May 1999 VisFiles column. And you can read the thoughts of Chris Johnson and others who have written Visualization Viewpoints articles and review an IEEE Visualization 2004 panel discussion organized by Theresa-Marie Rhyne.[5] ■

References

1. W. Hibbard, "Top Ten Visualization Problems," *Computer Graphics*, May 1999, pp. 21-22.
2. W. Hibbard, "Confessions of a Visualization Skeptic," *Computer Graphics*, Aug. 2000, pp. 11-13.
3. W. Hibbard, C.R. Dyer, and B. Paul, "Toward a Systematic Analysis for Designing Visualizations," *Scientific Visualization: Overviews, Methodologies, and Techniques*, G.M. Nielson, H. Hagen, and H. Mueller, eds., IEEE CS Press, 1997, pp. 229-251.
4. M. Hutchins, *Modelling Visualisation Using Formal Algebra*, doctoral dissertation, Dept. of Computer Science, Australian Nat'l Univ., 1999.
5. T.-M. Rhyne et al., "Can We Determine the Top Unresolved Problems of Visualization?" to be published in *Proc. IEEE Visualization 2004 Conf.*, IEEE CS Press.

Contact Bill Hibbard at billh@ssec.wisc.edu.

Contact Theresa-Marie Rhyne at tmrhyne@ncsu.edu.