Fourteen Ways to Say Nothing with Scientific Visualization

Al Globus, Computer Sciences Corporation

Eric Raible, NASA Ames Research Center

Those not properly initiated into the mysteries of visualization research often seek to understand the images rather than appreciate their beauty. Such pernicious activity must be discouraged.

Upon reading David Bailey's seminal work, "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers,"[1] we were struck by the brilliant simplicity of the concept. Bailey ends with the admonition, "Conclude your technical presentation and roll the videotape. Audiences love razzle-dazzle color graphics, and this material often helps deflect attention from the substantive technical issues." Unfortunately, Bailey gives no guidance in the means and methods for producing such a result. This article humbly seeks to fill this void.

There are numerous time-tested scientific visualization techniques for producing pretty pictures while avoiding unnecessary illumination of the data. Our collection has been culled from the scientific visualization literature and numerous presentations we have given and attended.

1. Never include a color legend

Many visualization techniques involve assigning colors to scalar data values. In lesser sciences, a legend relating colors to values is provided. In our exalted art form, not only does a legend mar the beauty of an image, but it may also divert the viewer into idle contemplation of reality. (Note: Images can be particularly enthralling if the sequence of colors is chosen solely on aesthetic grounds. For optimal results, subtly use separate color mappings for different parts of the image.)

2. Avoid annotation

In dreary old-fashioned sciences like physics and biology, investigators have been known to annotate their images with arrows

0018-9162/94/$4.00 © 1994 IEEE

pointing out features of supposed interest along with explanatory text. This promotes clarity of understanding, undermining the sense of awe and confusion the best scientific visualization engenders.

3. Never mention error characteristics

If scientists using visualization software knew that visualization techniques might introduce error, they might not be properly impressed by our masterworks. Therefore, never imply by word or deed that your algorithm introduces any error whatsoever. After all, if the picture looks good, it must be correct.

4. When in doubt, smooth

Always strive for the smoothest possible surfaces, since they look so much better than a lot of ugly facets. For example, choose lighting normals to hide sharp edges in the data. Smoothing can also obscure errors and allow users to publish their results earlier.

5. Avoid providing performance data

When you are presenting a pretty picture, some stick-in-the-mud may ask how long it took to calculate. The fact that your ray-cast isosurface took hours of massively parallel supercomputer time to generate when nearly identical results could be achieved in seconds using marching cubes[2] on a workstation is irrelevant. In addition to being smoother (see rule 4), a ray-cast image can include some wispy stuff scattered around to give the image an ethereal quality.

6. Cunningly use stop-frame video techniques

Each frame of a scientific video usually takes seconds, minutes, or even hours to produce. To achieve smooth animation, it is usually necessary to generate video frames one at a time and transfer them separately to tape. They can then be played back at 30, or even 60, frames a second. Stop-frame techniques can dramatically improve perceived software performance. The magic is lost, however, if you are so foolish as to tell anyone what you're doing.

Faithful adherence to the rest of the rules will help avoid tedious debugging of software that already produces pretty pictures.

7. Never learn anything about the data or scientific discipline

Debugging scientific visualization software is much more difficult if you are worried about producing correct results. Irritating details like accurate interpolation techniques get in the way; in many cases, ad hoc interpolation techniques can produce much prettier pictures with significantly less work. Better yet, programming bugs can sometimes produce stunning images. If you don't know what to expect, you won't have to find and fix such bugs. As we know, beauty is the higher truth.

8. Never compare your results with those of other visualization techniques

Comparing results with those of other visualization techniques is fraught with danger. You may detect bugs in your code that will need to be fixed, a tedious chore. Much worse, other techniques may produce prettier pictures.

9. Avoid visualization systems

Visualization systems, such as FAST (Flow Analysis Software Toolkit)[3] and AVS (Application Visualization System),[4] provide mechanisms to add modules implementing new visualization techniques. There are two problem with these systems. First, users may violate rule 8 to your discomfort. Second, visualization systems are usually "not invented here."

10. Never cite references for the data

If you cite a reference describing the data used to generate images, someone may read the paper and discover that your visualization bears no relationship to the key elements the original experiment was meant to elucidate. Some viewers may consider this a flaw that detracts from your picture's appeal.

11. Claim generality but show results from a single data set

It can be difficult to write visualization algorithms that function properly on a variety of data. Much effort may be saved by running your software on one (small) data set and using viewing angle and color-map manipulations to make the images look different, as if from multiple data sets. Follow rule 10 so that no one will know what you're doing.

12. Use viewing angle to hide blemishes

Many otherwise excellent algorithms produce 3D objects containing unsightly blemishes. Avoid carelessly choosing viewing angles that expose such flaws. If a suitable angle cannot be found, try another data set. If another data set is too much trouble, then:

13. If viewing angle fails, try specularity or shadows

Sometimes every possible viewing angle is marred by some small ugliness. In these cases, try adding shadows or brilliant highlights in appropriate places. However, never resort to using a paint program for touching up your image; that wouldn't be scientific.

14. 'This is easily extended to 3D'

Three-dimensional algorithms are almost always much more difficult than 2D, and the effort of generalizing a promising 2D algorithm to 3D can detract from producing pretty pictures. To both impress your colleagues and avoid much tedious work, simply claim that your algorithm "is easily extended to three or more dimensions." Only the real pros will know you're lying, but they won't challenge you, since we all make identical claims.

As Bailey pointed out, "It is often necessary for us to adopt some advanced techniques in order to deflect attention from possibly unfavorable facts." Follow these rules faithfully and you'll never need to compromise your pretty pictures with the mundane realities of science. May your images be accepted by Siggraph. ■

Acknowledgments

We acknowledge helpful contributions and comments by David Bailey, Dan Asimov, Michael Gerald-Yamasaki, Creon Levit, and Sam Uselton. Nahum Gershon's panel session[5] and Wayne Lytle's brilliant video[6] provided inspiration.

The work of Al Globus is supported through NASA Contract No. NAS2-12961.

References

1. D. Bailey, "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers," *Supercomputer Rev.*, Vol. 4, No. 8, Aug. 1991, pp. 54-55.
2. W.E. Lorenson and H.E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics* (Proc. Siggraph 87), Vol. 21, No. 4, July 1987, pp. 163-169.
3. G. Bancroft et al., "FAST: A Multiprocessing Environment for Visualization of CFD," *Proc. Visualization 90*, IEEE CS Press, Los Alamitos, Calif., Order No. 2083 (microfiche only), 1990, pp. 14-27.
4. C. Upson et al., "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, July 1989, pp. 30-41.
5. N.D. Gershon, chair; J.M. Coggins, P.R. Edholm, A. Globus, V.S. Ramchandran, panelists, "How to Lie and Confuse with Visualization," *Computer Graphics* (Proc. Siggraph 93), Vol. 27, No. 1, 1993, pp. 387-388.
6. W. Lytle, "The Dangers of Glitziness and Other Visualization Faux Pas," videotape described in *Visual Proc: The Art and Interdisciplinary Programs of Siggraph 93*, Vol. 27, No. 1, 1993, p. 64.

Al Globus is a senior computer scientist with the Computer Sciences Corporation at the NASA Ames Research Center. His research interests include scientific visualization, space colonization, and computer-network-based education.

Globus received a BA in information science from the University of California at Santa Cruz in 1979 and has taken graduate-level courses at Stanford University. He is a member of the IEEE Computer Society and the American Institute of Aeronautics and Astronautics.

Eric Raible is a computer scientist in the Applied Research Branch at the NASA Ames Research Center. His research interests include scientific visualization, programming languages and environments, and user interfaces.

Raible received a BS degree in computer science from the Massachusetts Institute of Technology in 1983.

Readers can contact the authors at NASA Ames Research Center, MS 27A, Moffett Field, CA 94035-1000, e-mail {globus, raible}@nas.nasa.gov.