The Render Button:

How a High-end Graphics Company Developed a Personal Graphics Product

Jon H. Pittman
Director of Software Development
Wavefront Technologies, Inc.
530 E. Montecito St.
Santa Barbara, CA 93103
(805) 962-8117

[decvax!]ucbvax!ucsbcsl!wavefro!jon

Abstract

The Personal Visualizer™ was developed to enable a casual user to create high-quality rendered images. Providing sophisticated rendering techniques to casual users forced us to confront a number of issues and to undergo a shift in values. These issues are outlined, and the ways we dealt with them are discussed. Things we would do differently as a result of our development experiences are also discussed.

Introduction

In July of 1989, Wavefront Technologies, Inc. introduced a product called the Wavefront Personal Visualizer™. Until that time, Wavefront had been known for high-end rendering and animation software targeted primarily toward entertainment and creative graphics. In other words, our products were directed toward people who used expensive graphics workstations to make "pretty pictures" for a living. The Personal Visualizer is oriented toward casual users who want to produce realistic images on low-cost engineering workstations. These people produce images as a means to an end rather than as an end in itself. They are typically engineers or scientists who use images as one tool in a set of tools to enable them to do their primary task. They use the product one or two hours a week rather than forty to eighty hours per week.

Developing the Personal Visualizer resulted in quite a culture shift at Wavefront. Where previously we focused on achieving ever-higher levels of photorealism and realistic motion, the Personal Visualizer forced us to learn about usability, information structuring, graphic user interface design, and simplicity. We had to expand our point of view beyond features and performance to make our product effective for the casual user. We found that the overriding design issue for our personal visualization product was one of conceptual integrity. To make visualization accessible for a casual user, we had to develop a clean, coherent product in which all of the components related logically to each other and acted as an integrated whole. We found the words of Fred Brooks, author of ***The Mythical Man Month*** [1], particularly apropos to our development effort:

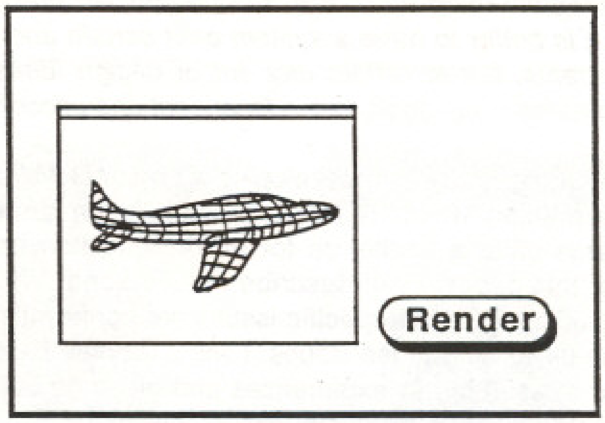
> *" ... conceptual integrity is **the** most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one which contains many good, but independent and uncoordinated ideas."*

For an organization such as ours, developing a personal visualization product was often a process of determining which features to omit in an attempt to achieve conceptual integrity. This was quite a challenge for an organization used to building products for expert users. In this paper, I will describe the Personal Visualizer project and product. I will then discuss some of the specific issues we confronted in developing the Personal Visualizer. While discussing these issues, I will speculate on some of the things we would do differently as a result of our experiences and on some of the directions we can expect to see personal visualization products take in the future.

The Personal Visualizer project

We originally began discussions with a graphics workstation vendor in January of 1988 on a rendering project. They wanted to take advantage of Wavefront's software rendering capability to make wireframe and hardware rendered images look "pretty." The request was for a magic "render button" that, when pressed, would turn a wireframe or hardware shaded image into a beautiful photorealistic image. When originally approached, we were somewhat skeptical of this endeavor. "Surely," we were told, "it can't be that difficult to add a button to the screen to render an image!" Of course, the request was somewhat naive. The problem was not one of adding a button to make the wireframe beautiful. It was one of underlying data representation and manipulation of all of the elements necessary to make a photorealistic picture.

Currently, making a photorealistic picture or animation is very much like making a Hollywood film. Before creating the picture, you must carefully construct a scene which contains all of the elements in the picture. In addition to constructing the spaces and objects in the scene, you must carefully place lights to establish the mood or atmosphere, place cameras to establish the view, apply surfaces to the objects to give them texture and character, and place backdrops to establish an environmental context. Of course, producing an image with more realism requires more detail and accuracy in the data describing the scene. To further complicate things, many of the techniques used are not direct analogues to the real world but tricks of the "smoke and mirrors" variety that are used simply to produce the picture. They are similar to stage sets which provide the illusion of reality when seen from a particular viewpoint. In the case of the workstation vendor who wanted the render button, most of the wireframe and shaded images on their graphic workstations were of CAD or scientific data. The underlying data structures did not have all of the scene information necessary to construct a realistic image. They contained only enough information for simple display techniques. They did not contain information rich enough to construct a photorealistic image.



A render button to make the wireframe image photorealistic

The Personal Visualizer was developed to fulfill the promise of the render button. It is designed to enable a casual user to create a photorealistic image. The Personal Visualizer provides a data management system to assemble into a ***scene*** all of the components necessary to make a picture. Once the scene is described, you may generate a picture or sequence of pictures at various levels of realism. You need not create geometric data; data translators allow geometric data to be imported from a CAD system or modeling package.

The Personal Visualizer is organized into a core product that provides basic scene assembly, data management, and rendering capabilities; and a set of options that extend the functionality of the core. The options provide capabilities to create geometry, motion, or surfaces; output images to video; and more extensive import of geometric data from CAD systems. The core Personal Visualizer is bundled on graphic workstations and the options are sold by Wavefront. The options allow a user to configure a visualization tool to meet his or her specific needs.

The user interface was the primary focus of the Personal Visualizer development effort. We developed it around a graphic user interface to provide access to a wide range of users. Since the Personal Visualizer was intended for a broad audience, we tried to create a simple, concise product that was easy to understand and use instead of producing a feature-laden application. Whenever we were confronted with a choice between a powerful but complex feature and reduced but conceptually clear functionality, we chose conceptual clarity and integrity.

This focus on usability and conceptual integrity was difficult for many members of the Wavefront staff. We were accustomed to pushing the envelope of rendering and animation technology. Our customers were, by and large, interested in more and more sophisticated rendering and animation features. Our previous focus was on technology rather than designing for conceptual clarity. Our culture was one which was very typically UNIX-oriented in that we viewed products as sets of small, useful tools which were strung together to achieve a desired effect or result. Trying to build a product for the casual user forced us to confront a number of issues differently. However, the Personal Visualizer product is evidence that we were successful in shifting our focus from the expert to the casual user.

The Personal Visualizer product

The Personal Visualizer's purpose is to make pictures. The core product collects data from a variety of sources, assembles that data into a scene, enables you to arrange lights and cameras, and allows you to render an image of the scene. The scene can be rendered at various levels of realism. The more realism you require, of course, the more information you have to add to the scene and the more time the image will take to render. The information you assemble to form a scene consists of a set of resources. The Personal Visualizer works with the following kinds of resources:

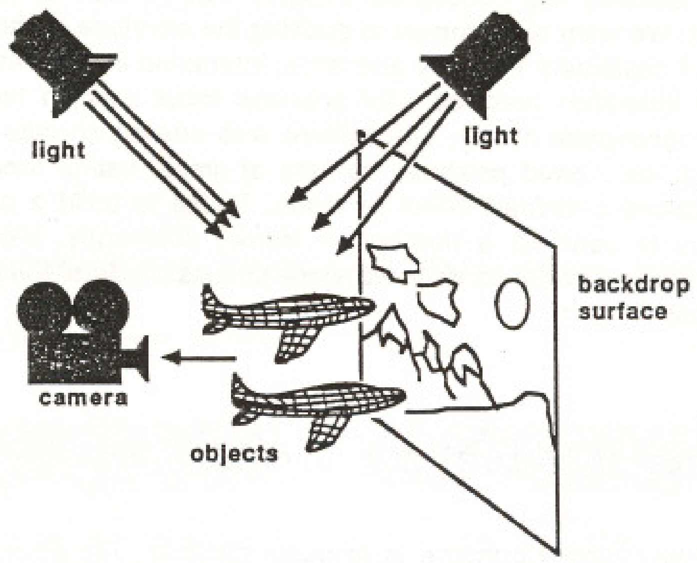
- **Objects** are geometric data. They provide the shapes that you will manipulate and place in space to create an image. Objects may be as simple as cubes or spheres or as complex as airplanes or buildings. Objects may be grouped together to form more complex objects.

- **Surfaces** describe color, texture, reflectance, and transparency properties. Surfaces are applied to objects much like decals are applied to a model airplane. The surface that is applied to an object will dictate the level of realism of the rendered image. Alternatively, surfaces may be used as backdrops.

- **Lights** describe the direction, color, and falloff of the light sources in a scene. The lights help establish the mood of the scene and interact with the surfaces to produce the rendered images.

- **Cameras** describe the view of the scene. The actual camera resource describes attributes such as focal length. The position and direction of the camera dictates the actual view of the scene.

- **Pictures** are the end results of the rendering process. You can render a scene and save it as a picture.



A set of resources comprising a scene

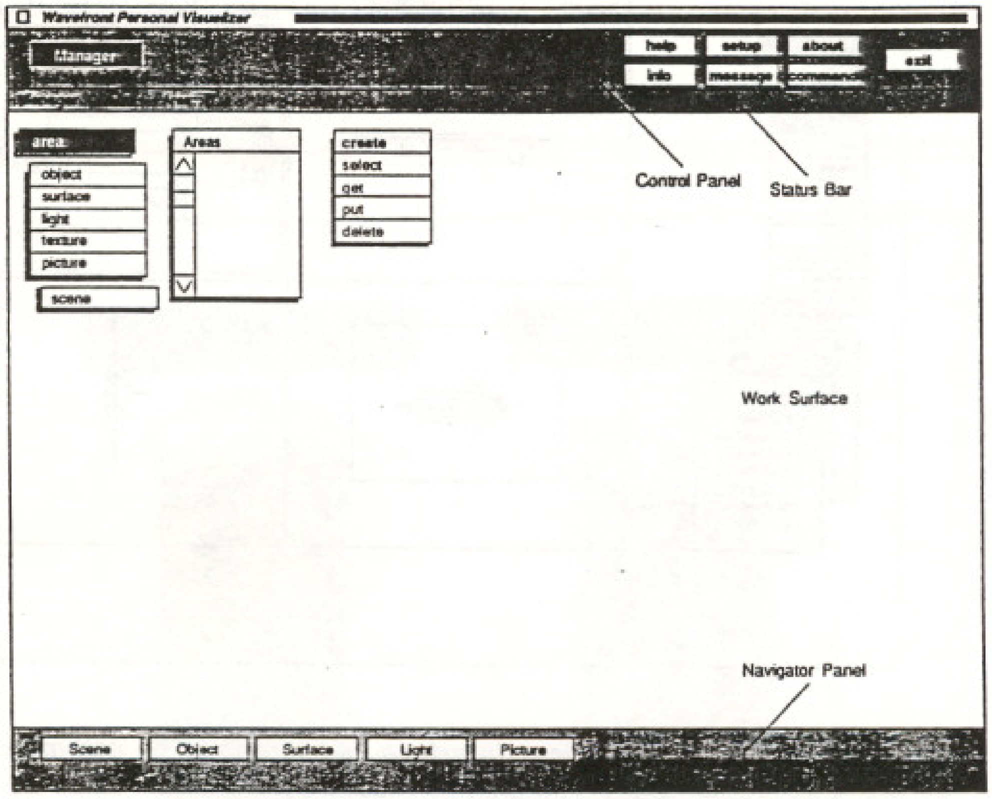
A number of these resources are shipped with the Personal Visualizer and are stored in a library that all users can access. The library contains several geometric primitives such as spheres, cones, cubes, and cylinders. In addition, it contains a model of a space shuttle and a T-45 jet fighter. It also contains a variety of lights, cameras, and surfaces. The surfaces include transparent surfaces, woods, rocks, metals, and various colors. You can use any of the resources in the library, import resources created by other applications, or add options to the Personal Visualizer that create and manipulate resources. The figure on the previous page shows an example scene with various types of resources.

The "face" of the Personal Visualizer consists of several different screens or ***editors.*** Each editor is focused on a particular task that the user performs. In the core Personal Visualizer, there are two editors:

- **The Manager** is the editor that you enter when you start the Personal Visualizer. It helps you organize and manage resources in your own private work areas. It is where data from other sources is imported to your work areas and data already in your work areas is exported for use by other programs or tools. The Manager also allows you to copy resources from the standard library and to share resources with others through a mechanism called the exchange.

- **The View Editor** is where you assemble scenes and render images. It allows you to add objects and lights to a scene. You can manipulate each of these resources using standard graphics transforms (translate, rotate, scale). You can also point lights or cameras at objects or associate surfaces with objects. Once a scene is assembled, you can test render an image to the screen or render a picture to save it for future use.

Optional editors that allow you to do things such as create and modify surfaces, paint on or annotate pictures, create and modify objects, or animate scenes can be added to the core product.



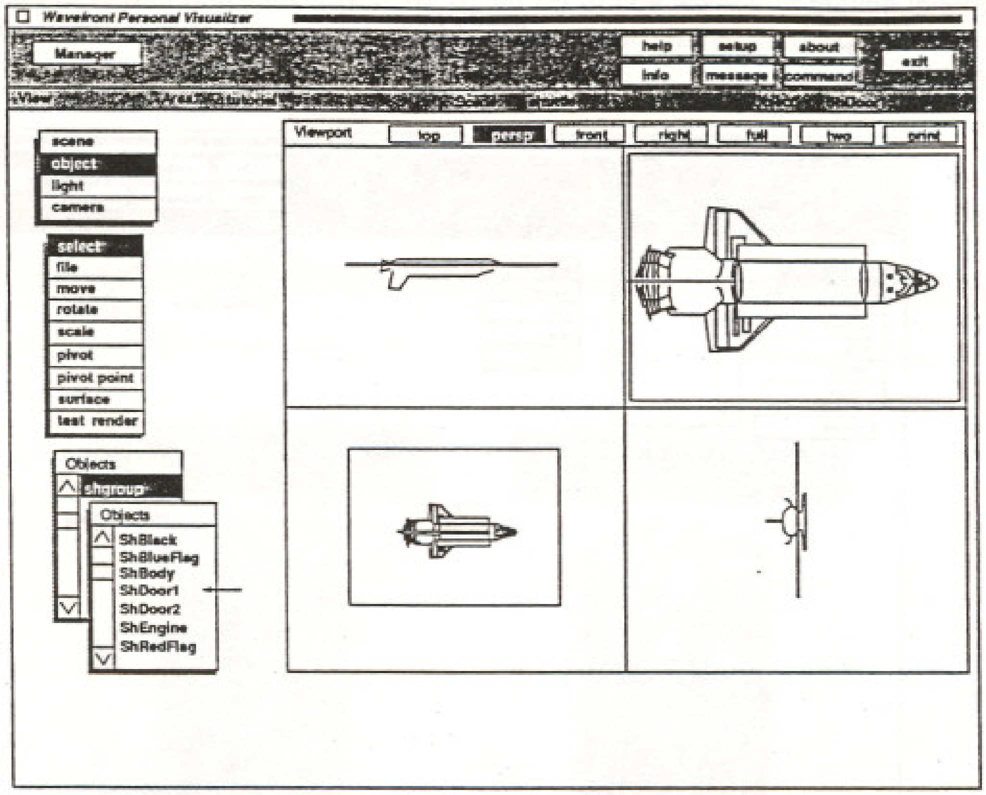
A visual framework common to all editors provides an identity for the Personal Visualizer

Although each editor has a unique function, all of the editors share a common look and feel. A set of elements appear in every editor and form a visual framework for the system. This framework establishes an identity or personality for the product. The elements make up this framework as shown in the figure above:

- **The control panel** -- allows you to get information about the state of the Personal Visualizer, modify preferences, invoke the command language, return to the Manager, or exit from the Personal Visualizer.

- **The status bar** -- indicates which items on the screen are currently selected.

- **The work surface** -- is the background area of the screen. It is the place where menus, dialog boxes, viewports, and other graphic elements appear.



The Personal Visualizer user interface is comprised of common graphic elements that form a consistent graphic vocabulary.

In addition to the common elements that appear in every editor, each editor is constructed from a standard set of graphic elements. These elements include menus, viewports, dialog boxes, and property sheets. The figure on the following page shows examples of some of these elements. We used these elements throughout the Personal Visualizer to ensure a consistent visual vocabulary and style. We tried to achieve conceptual integrity that was as effective as that of the Xerox Star™ [2,3] and the Apple Macintosh™ desktop.

To supplement the graphic user interface, we incorporated a programmable command language that provides access to all of the functionality of the product. This allows experienced users to work more quickly and to construct "programs" that provide higher-level functionality tailored to a particular use. We crafted the command language with the same view toward consistency and conceptual integrity that we used for the graphic user interface.

Issues - What decisions did we face?

In developing the Wavefront Personal Visualizer, we had to confront a number of tough design issues. These issues, in some cases, challenged the basic tenets upon which we had built our high-end products. In other cases, they were new concerns that arose from trying to address a broad class of users.

The major issues we confronted are discussed below. First, the issue is defined as a question or set of questions. Following this is a brief discussion of our approach to the issue. In most cases, the discussion does not completely answer the questions posed. It is intended that the questions be used as a point of departure for further discussion and thought, not that they be completely addressed in this paper. These issues are arranged into four categories: design, rendering, interaction, and infrastructure.

Design Issues deal with the way the actual features, look and feel, and configuration of the product is determined.

- ***How should personal visualization products be designed?*** *Personal visualization products pose a rigorous design challenge. How do we make something complex simple to use yet provide enough power for the high-end user? How do we layer capability to grow with the user's competence? Making things easy to use is different from pushing the high-end of technology. It requires different skills than devising a ray-tracing algorithm or radiosity technique. What skills and approaches to product design are required for personal visualization?*

  In designing the Personal Visualizer, we used a full-time product designer dedicated to the project. His job was to ensure that the conceptual integrity of the product was maintained. He worked extensively with the developers, technical writers, and marketing staff to ensure that the product was perceived as an integrated whole. Although the design effort was successful, in retrospect we would do some things differently:

  * **Build a user's conceptual model early in the process and maintain it.** We should have built an explicit model of how our user perceived the product and constantly tested the design against that model.

  * **Build and maintain a glossary.** We often found over the course of the project that we called something by many different names or, conversely, that we overloaded certain terms. We should have built a glossary of all of the terminology that the user saw and constantly evaluated the glossary against the user's conceptual model.

  * **Prototype.** We spent a lot of time discussing various design alternatives. However, most people could not evaluate and critique design decisions effectively until they became concrete through

representation in a prototype. We did most of our prototyping on paper. We should have developed more interactive prototypes and made rapid prototypes an integral part of our design and development process.

* **Usability test.** When the product got into the hands of real users, we often found that they had problems we did not anticipate or had good ideas that were too late to implement. We would do much more usability testing with real users early on in the design and development process.

- ***How do you build the correct user's conceptual model?*** *Who are the users of a personal visualization product and what characteristics do they have? Users may have a broad range of skills and computer and graphics literacy. How do you make a personal visualization product easy for the novice yet powerful enough to satisfy the proficient? Is it appropriate to try building a product that attempts to serve the needs of both the expert and the novice user? What is the user's mental model of the tool and how can it be reinforced by the product?*

  The best way to build a model of how the user views the system is to understand your user thoroughly and frequently test the model against the user and against the product. We had difficulty understanding our users, since most of Wavefront's traditional user base consisted of professional animators. It was difficult for them to place themselves into the shoes of a novice user. We got the most valuable information from users once we had the product out into their hands in the testing cycle.

- ***Is it better to provide one universal tool or lots of little tools?*** *To provide seamless access to visualization tools, two choices exist. You can create a "universal system" which does many things, or a collection of small, focused tools that talk to each other. Which is more effective?*

  To the user, this should not make a lot of difference. The structuring of applications into tools or editors is more an artifact of the way we design and implement user interfaces than the way someone would like to work. In our case, we provide a universal tool in that we provide a framework which is consistent throughout the Personal Visualizer, but we break functionality into separate editors that have limited scope and function. In effect, our editors are viewed as small, focused tools by the user. This kind of partitioning of functionality seems to be effective in keeping each tool simple. However, as the number of tools grows, it is difficult to keep navigation between the tools simple and to provide a clear picture of how total functionality is segmented between the tools.

- ***How do you avoid creeping featuritis?*** *Personal visualization is a new, exciting technology. Everyone has features and functions they would like to see incorporated in personal visualization products. The temptation is to become all things to all people. Unfortunately, this may result in a product that does a lot of* *things but none of them well. How do you develop a visualization tool with broad appeal, yet maintain a clear, clean, simple product?*

Time and market constraints help with this problem. We found it a constant battle to keep the product from becoming an accretion of interesting but unrelated ideas and features. We had to constantly focus on maintaining simplicity and integrity. Of course, as the end of development loomed closer and people realized that infinite time to develop was not available, it was much easier to find out what was important and what was not. In addition to time pressure, maintaining a clear focus on the product's purpose and intended user can help control random feature addition and maintain conceptual integrity.

Rendering Issues deal with the way a personal visualization product deals with graphic display and rendering.

- ***What level of realism is appropriate?*** *In the entertainment market we assumed photorealism was the goal. However, to achieve photorealistic effects, we often have to go to great pains to set up data. Are users willing to pay the costs of photorealism? Can we make it easy for "Joe Public" to achieve realism? Does "Joe Public" want to achieve photorealism? Does photorealism obscure or enhance the message? Are there ethical questions concerned with producing photorealistic images?*

  We provide a range of levels of realism in the Personal Visualizer. In fact, we make extensive use of hardware shading, and the user can interact with a scene using a very high level of realism without resorting to software rendering. It is clear that many users do not need or want a high level of realism. However, we feel that the standard of realism that users expect to see will rise as the capability to produce such realism becomes accessible and the cost becomes reasonable. We must keep in mind, however, that most people who use computer graphics are trying to do useful work, not just produce pictures. Pictures enable them to do something. Realism may or may not help that person get their job done. It should be viewed simply as another tool in their repertoire.

- ***What happens when graphics become ubiquitous?*** *Workstation vendors are all interested in providing graphics capability. Hardware shading and intrinsic graphics libraries are becoming standard on many workstation products. How does this affect the development of personal visualization products? Will this cause a shift of focus from rendering techniques to interaction techniques?*

  We designed the Personal Visualizer to take advantage of hardware rendering capabilities that exist on various graphic workstations. It is structured such that the transition from hardware rendering to software rendering is seamless, although there is a discontinuity from interactive performance to waiting for a picture to render. It is clear that more and more rendering capabilities will be built into hardware or into low-level system software. Software techniques, however, will be able to provide more realism for some time to come. The real problem, though, is not one of rendering or realism. The real problem is how to use sophisticated rendering technology that comes with the hardware. Harnessing all of the "gee-whiz" rendering techniques to do something useful is a much different problem from rendering. Doing useful work with hardware rendering requires good interaction techniques. When realistic rendering becomes ubiquitous, we will be able to turn our attention to the real problem at hand, how to use visualization techniques to accomplish useful work.

Interaction Issues deal with the dialog between the human user and the personal visualization product.

- ***What style of interaction is appropriate?*** *Two traditional alternatives are graphic interaction and command-based interaction. Some people prefer to use typed commands or scripting languages to interact with an application. Others prefer graphic user interfaces. Which is more appropriate? Are the two approaches mutually exclusive?*

  Although the current trend is toward graphic user interfaces, there are still a lot of reasons to use commands that you type at the keyboard. Commands can be more succinct and precise than graphic interactions, can be grouped into scripts for repeated execution, and may be combined with programming constructs to allow extension of the base functionality of the product. We are starting to see scripting languages incorporated into graphic user interfaces. HyperTalk, HyperCard's scripting language, is a good example of this. Combining a graphic user interface and a programmable command language results in more power and flexibility than either approach taken separately. Providing both allows users to select the interaction mode that best suits their needs. We provide both in the Personal Visualizer. In our implementation, the command language and graphic user interface are relatively independent of each other. You have to switch back and forth between the two styles of interaction. A superior implementation would closely integrate the two means of interaction so they worked together.

- ***How should a personal visualization product deal with multimedia?*** *Media such as sound, video, kinesthesia, and tactile are becoming more pervasive. Soon they will be integrated into hardware and software environments. How can a personal visualization product exploit these capabilities? For example, now video is expensive and messy. How can we make it easy and inexpensive for the user to access?*

  In the Personal Visualizer, we used one medium (the graphic display), three channels (mouse and keyboard in and graphic display out), and three dimensions. However, this is only a modest taste of the possibilities in a personal visualization tool. There is no question that personal visualization tools should be integrated with multimedia technology. We interact with the real world using all of our senses. It is imperative that we interact with our visualization tools in a similar fashion. It will be necessary to build mechanisms into the infrastructure needed for personal visualization tools to support multimedia, multi-channel, and multi-dimensional interaction between the user and computer. This will substantially increase the communication bandwidth between the user and the computer. Currently, UNIX workstations don't even deal well with video output, let alone other types of media. Thus, we still have a long way to go before dealing effectively with multimedia.

Infrastructure issues deal with the ways personal visualization tools relate to their operating environment (hardware, UNIX, graphics libraries) and other visualization tools.

- ***What is the role of UNIX?*** *UNIX has many features that allow clean integration of tools and provide power to the expert user. However, UNIX can be quite daunting to the casual user. How can a personal visualization product coexist peacefully with UNIX? Should the two ignore each other or is there a graceful way to integrate the best of both worlds?*

  Our charter for the Personal Visualizer was to make visualization as accessible as possible for the casual user. One of the most difficult learning problems for new Wavefront users is learning about and dealing with UNIX. In addition, there is currently a lot of work in the UNIX community aimed at hiding the raw UNIX user interface and providing a more accessible interface such as Silicon Graphics' WorkSpace [4], NextStep, and OpenLook to make UNIX appealing to a broader range of users. Thus, we chose to ignore UNIX once a user has invoked the Personal Visualizer. As the work on making UNIX more accessible progresses, we expect to tie the Personal Visualizer more closely to standard UNIX graphic user interface techniques.

- ***Where does the data go?*** *Graphics applications require lots of data and various data elements may be closely related. The UNIX file system and a byte-stream data model may not be appropriate for a personal visualization product. The complexity of graphic data may not lend itself to such an approach. How do we organize data and present that organization to a casual user in a way that makes sense? How can we make the management of data seamless and effortless for the casual user?*

  A scene may contain a lot of data and there may be complex relationships between data elements. If we kept all the data for each scene in one file, the file would be big and cumbersome and it would be difficult to share data elements between scenes. If we kept the scene data in a lot of small files, a user might inadvertently corrupt the scene by removing or renaming some of the files. Ideally we wanted a file system that let us control user access. Since we did not have such a scheme available, we decided to hide the database from the user. The database exists in a hidden subdirectory below the user's home directory. We provide data manipulation capability in the manager. The user is discouraged from directly manipulating the database. This was (and still is) a fairly controversial issue within Wavefront. The other alternative is to let the user use the UNIX file system directly. Given our approach of hiding UNIX from the user, letting the user deal with the file system directly did not seem appropriate.

- ***How do we fit personal visualization tools together?*** *To allow seamless integration of tools, a number of standards are needed. These include data interchange, look and feel, portability, networking, live links, and peripheral device standards.*

  UNIX has a tradition that supports building a lot of little, focused tools, and stringing them together to do useful work. The operating environment supports such a model through the file system and mechanisms like pipes. It works very well for byte-stream data such as text. It does not work so well for complex graphic data. The intent behind UNIX is a good one. The models it supports, however, are not appropriate for the kinds of data that must be supported in a personal graphics system. Supporting personal graphics requires a new infrastructure that deals with graphic data and the interaction of graphic software components. Lacking such an infrastructure, we had to build our own. I have to admit that, although the intent was to build an infrastructure that supported the linking together of a lot of small tools, the one we developed was a bit too monolithic. We are now working to rectify that problem.

- ***How does a personal visualization product adapt to changing environments?*** *The hardware and operating environments in which personal visualization products operate is changing very rapidly. New hardware, applications, peripheral devices, communications technology, etc. are continually evolving. How can a personal visualization product adapt to these changes?*

  Eventually, the underlying hardware and operating environment will be of little concern to the developers of visualization tools. We are already beginning to see less reliance on hardware environments because of the proliferation of UNIX. This is the result of the use of standard hardware components and the standardization of the UNIX operating system. Unfortunately, no clearly established standards exist for graphic user interfaces, graphics display libraries, or rendering. As these standards become established, visualization tools will adapt to new environments easily. In effect, they will be tied to an infrastructure which abstracts them from the details of hardware and operating environment.

One common theme throughout many of these issues is that we need an infrastructure to support personal visualization tools. In the absence of such an infrastructure, we had to invent one for the Personal Visualizer. However, it was a massive effort. It is not practical for everyone who wants to build a personal visualization tool to build such an infrastructure. We needed something that would abstract hardware, peripheral devices, operating environment, and graphics display and interaction in such a way that anomalies are avoided. UNIX was designed as such an infrastructure for text-based systems and has proven to be a very robust infrastructure for such systems. However, it is beginning to show signs of stress when dealing with personal graphics systems. What is needed is a new infrastructure (or extensions of the old infrastructure) that supports the needs of personal visualization tools.

Conclusion - What does the future hold for personal visualization?

The Wavefront Personal Visualizer is the first step toward personal visualization tools that will be commonplace in the future. The real limitation of the Personal Visualizer is that it is primarily an output tool. It shows the results of rendering a scene that represents some real-world or artificial environment or phenomenon. It has limited capabilities to manipulate, interact with, and explore that environment or phenomenon. We have a long way to go before we fully achieve the kind of pictorial conversation discussed by Rob Myers in 1986 where a true graphic dialog is carried out between humans and computers [5].

I hope that personal visualization tools of the future are truly transparent to the user. Graphics will be ubiquitous. Just as the text-based computer now permeates our lives so, too, will graphics-based computers. They will deal inherently with three dimensions since the world that they interact with and represent is three-dimensional. They will also be multimedia and will carry on dialog with humans through multiple communication channels. We will need to build visualization tools that can harness all of this communications and capability.

Although the graphical user interface of the Personal Visualizer is a big step toward making photorealistic rendering accessible to the casual user, it is my hope that we will look back on such tools as complex and cumbersome in the future. We need to strive for personal visualization tools that enable us to manipulate objects and environments in the computer as easily as we manipulate them in the real world.

Acknowledgements

The Wavefront Personal Visualizer was developed by Chris Kitrick, Dave Immel, Julie Daily, Cliff Brett, Lindy Lindstrom, Jay Tomlinson, and Josh Aller. Roy Hall, Kim Shelley, Don Brittain, and Jon Barber developed software components that were used in the Personal Visualizer. Derry Frost was the product designer and was responsible for the conceptual integrity of the product. Diane Ramey, Catherine Babine, and Brad Weed wrote the documentation and the on-line help. Steve Miley was responsible for configuration management and release engineering. Jim Hill and Patsy Crawford tested the product and helped customers with beta testing. Karen Gadway provided administrative support. Each of these individuals as well as many others contributed to the success of the project and grappled with the issues described in this paper.

The Wavefront Personal Visualizer was developed in conjunction with Silicon Graphics, Inc. Silicon Graphics provided the original impetus for the project and was the originator of the "render button" concept. Tim Heidmann, Ed Bank, and Trish Jones from Silicon Graphics worked closely with us in our development effort.

Julie Daily, Derry Frost, Diane Ramey, Chris Kitrick, Dave Immel, Karen Gadway, and Care Heller-Pittman made helpful suggestions and comments in reviewing this paper for publication. Derry Frost and Diane Ramey assisted in preparing the figures.

References

[1] Frederick P. Brooks, *The Mythical Man-Month, Essays on Software Engineering.* Addison-Wesley. 1975. Page43.

[2] Dr. Daniel E. Lipkie, Steven R. Evans, John K. Newlin, Robert L. Weissman. Star Graphics: An Object-Oriented Implementation. *Computer Graphics.* Vol 16., Number 3. ACM Siggraph. July 1982. Page 115-124.

[3] Jeff Johnson, Theresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, Kevin Mackey. The Xerox Star: A Retrospective. *Computer.* Vol 22. Number 9. IEEE Computer Society. September 1989. Page 11-29.

[4] *Working in a New Space.* Silicon Graphics, Inc. 1989.

[5] Rob Myers, Pictorial Conversation: Design Considerations for Interactive Graphical Media. *Proceedings of the 1986 Monterey Computer Graphics Workshop.* USENIX Association. November 1986. Page 17-35.