HP Kayak: A PC Workstation with Advanced Graphics Performance

Ross A. Cunniff

World-leading 3D graphics performance, normally only found in a UNIX® workstation, is provided in a PC workstation platform running the Windows NT® operating system. This system was put together with a time to market of less than one year from project initiation to shipment.

Computer graphics workstations are powerful desktop computers used by a variety of technical professionals to perform their day-to-day work. Traditionally, such computers have run with a version of the UNIX operating system. In the past year, however, workstations featuring Intel processors such as the Pentium™ Pro and Pentium II and running the Microsoft® Windows NT operating system have begun to gain ground in both capability and market share. Hewlett-Packard has historically been a leader in the UNIX workstation business. In February, 1997, Hewlett-Packard began a project to put its high-performance workstation graphics into a PC workstation platform.

Technical Challenges

Fitting HP workstation graphics into a Windows NT platform was not an easy task. The task was made more exciting with the addition of schedule pressure. The schedule gave us only four months to reach functional completion and only two months after that to finish the quality assurance process. This schedule was made even more challenging because the hardware was not yet complete. It was difficult at times to distinguish software defects from hardware defects. This article describes how we overcame some of the challenges we encountered while implementing this project.

The Hardware

The hardware for the HP Kayak workstation (**Figure 1**) is based on the VISUALIZE fx4 graphics subsystem for real-time 3D modeling (see the article on page 28). However, a couple of changes were necessary. First, to achieve

Ross A. Cunniff
A senior software engineer at the HP Performance Desktop Computing Operation, Ross Cunniff has been with HP since 1985. He was the lead software engineer for the 3D device driver used in the HP Kayak workstation. He continues to be the lead 3D device driver engineer for high-end graphics products. He received a BS degree in mathematics and a BS degree in computer science in 1985 from the University of New Mexico. His professional interests include computer graphics, particularly 3D hardware acceleration.

Figure 1

An HP Kayak XW workstation.

the performance available in the graphics hardware, the bus interface had to be changed from the standard Peripheral Component Interconnect (PCI) to the accelerated graphics port (AGP),† since no commodity PC chipset supported PCI 2X. With normal industry-standard PCI, we would have been limited to 132 Mbytes/s for I/O, which would have hurt our performance on several important benchmarks. With the accelerated graphics port, the available I/O bandwidth increased to 262 Mbytes/s.

The second change necessary to the hardware was the addition of industry-standard VGA graphics. During the boot process of Windows NT, and at occasional intervals after that, the computer will access VGA graphics registers directly. To achieve this, a VGA daughtercard was created that displays its graphics through the video feature connector created for the UNIX video solution. The main graphics board was modified slightly, making it possible to dynamically switch between VGA graphics and VISUALIZE fx^4 graphics. **Figure 2** shows a hardware block diagram for an HP Kayak workstation.

† AGP is a bus that transfers data to and from a graphics accelerator.

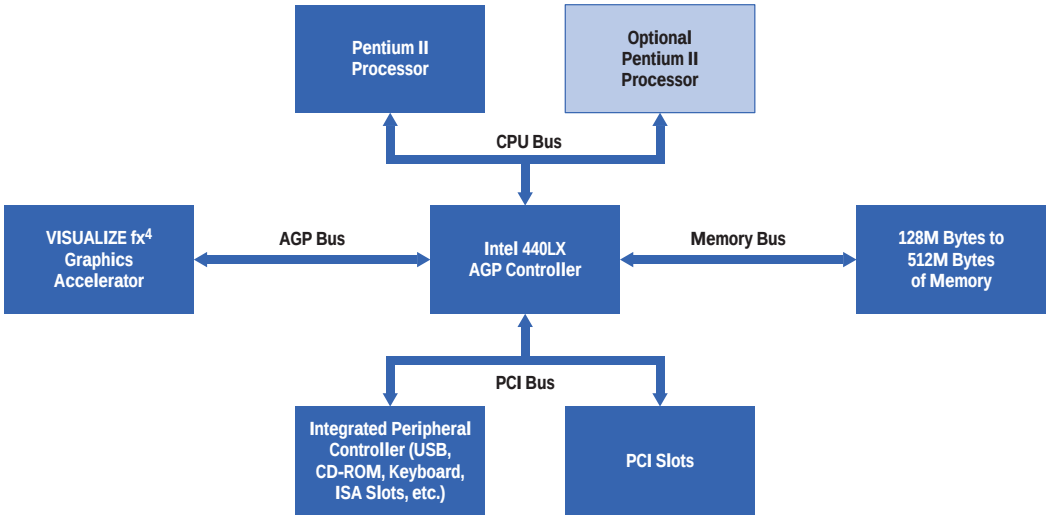
Windows NT Driver Architecture

The fact that the hardware for the HP Kayak workstation is similar to the VISUALIZE fx^4 hardware, which runs the UNIX operating system, made the software effort much easier. However, many significant hurdles had to be overcome to get the software running under Windows NT.

The first challenge was the Windows NT device driver architecture (**Figure 3**). On HP-UX*, graphics device drivers have a large amount of kernel support, allowing them to access the graphics hardware directly from user-level code without having to execute any special locking routines. This direct hardware access (DHA) method is not present on Windows NT. Instead, all accesses to the hardware must be performed from the kernel (ring 0 in **Figure 3**).

Figure 2

A hardware block diagram for an HP Kayak workstation.

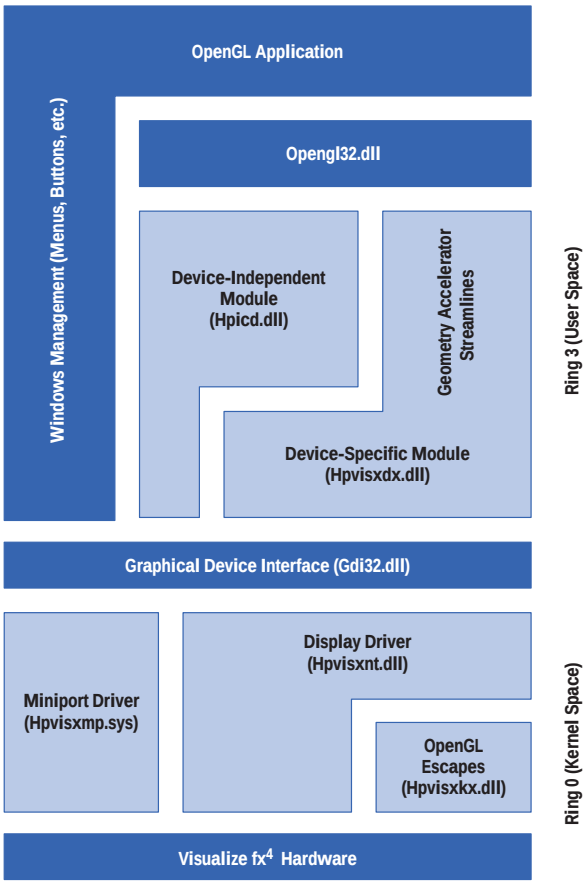


Fortunately, the VISUALIZE fx^4 architecture specifies a buffered form of communication in which graphical commands are placed into command data packets in a large buffer in the hardware. It was a simple task to modify the HP-UX drivers to access a software allocated command data packet buffer instead. When one of these software buffers gets full, it is passed to the ring 0 driver that forwards the buffer to the hardware.

The lighter-shaded modules in **Figure 3** represent the libraries that were delivered by HP to support the VISUALIZE fx^4 hardware. The libraries in ring 3 (Hpicd.dll and Hpvisxdx.dll) were fairly straightforward ports of the corresponding UNIX libraries libGL.sl and libddvisxgl.sl. The libraries in ring 0 (Hpvisxmp.sys, Hpvisxnt.dll, and Hpvisxkx.dll) had to be created from scratch to support the Windows NT driver model. These modules make up about 30 percent of the size of the ring 3 modules.

Figure 3

The Windows NT device driver architecture.



Integration with 2D Windows NT Graphics

The second challenge was to integrate the 3D OpenGL graphics support with the standard Windows NT graphical device interface. Microsoft specifies two methods that can be used to do this. The first, called a *miniclient driver*, is a rasterization-level OpenGL driver that uses the Microsoft OpenGL software pipeline for lighting and transformation. This driver would have been easy to create, but it would not have allowed us to take advantage of the hardware transformation and lighting provided by VISUALIZE fx^4.

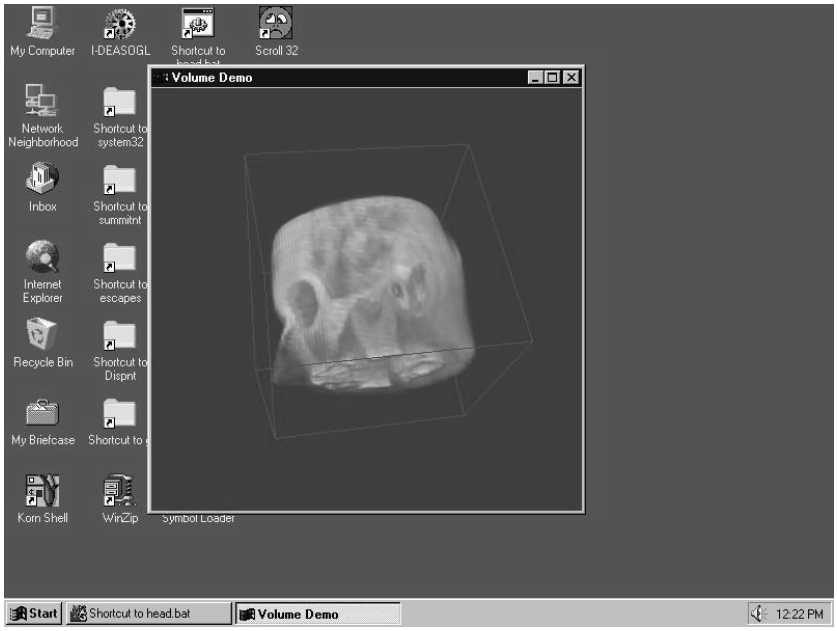
The second method, called an *installable client driver*, is a geometry-level OpenGL driver that leaves implementation of the lighting and transformation pipeline up to the driver writer. The driver allows us full access to all OpenGL API routines. This is the route we chose because we already had a full implementation of OpenGL, which we had created to run on the HP-UX operating system. This implementation was ported to the installable client driver model over a span of several weeks, while we added support for Windows NT multithreading. The bulk of the VISUALIZE fx^4 graphical device interface driver was written by a separate team of experts without much consideration for 3D graphics acceleration. This enabled them to get the Windows NT display driver running in a short amount of time and allowed them to continue enhancing 2D performance without severely impacting the 3D device driver team. Some of the results of these efforts are shown in **Figure 4**.

Integrating the Windows NT Driver with Ring 0

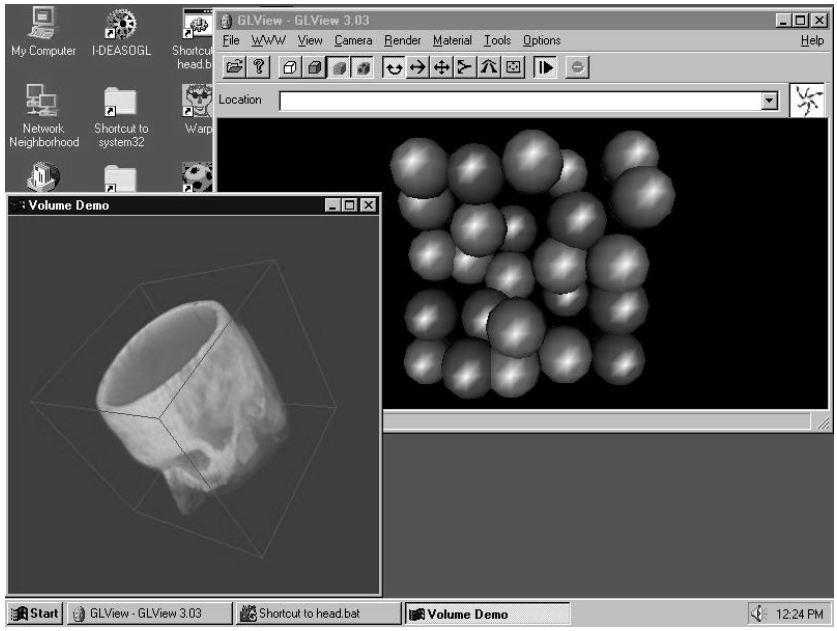
A third challenge was to integrate the Windows NT driver with the ring 0 portion of the OpenGL driver while maintaining separate code bases for the different teams. We decided to make our ring 0 driver a separately loadable library. This decision kept the source code separate. It enabled much faster edit-compile-debug cycles, since it allowed us to replace a portion of the ring 0 driver without having to reboot the computer. However, the separation added extra complexity because we had two very different drivers accessing the same piece of hardware. To solve this problem, we created a variable called a *hardware access token*. Each driver has a special token

Figure 4

(a) A 3D image in a 2D environment. (b) Several 3D programs in a 2D environment.



(a)



(b)

that it places in the hardware access token to indicate that it was the last driver to access the hardware. When a driver detects that the token is not its own, it executes procedures known as *context save* and *context restore*. The context save reads all applicable hardware state information from the device into software buffers. The context restore places the previously saved state back into the hardware. This same mechanism is used to mediate hardware accesses between different processes running OpenGL.

Integration of VISUALIZE fx^4 Architecture

A fourth challenge for the team was the integration of the VISUALIZE fx^4 stacked planes architecture (**Figure 5a**) into the Windows NT environment. Workstations traditionally have very deep pixels, each pixel having up to 90 bits of information. This information includes support for such things as transparent overlays, double buffering, hidden surface removal, and clipping. Windows NT expects a slightly different model, in which the extra per pixel information is allocated in offscreen storage when a 3D rendering context is created (**Figure 5b**). What this means is that when the window state is changed (for example, when a window is moved on the desktop), Windows NT does not make any special calls to the device driver. This presented a problem, since our stacked planes architecture needs to keep all of the extra information directly associated with the correct visible screen pixels.

To fix this problem, we used a Windows mechanism called a *window object* (**Figure 6**). The window object tracks a window state and executes callbacks into our driver when a window state is modified. This added an unfortunate amount of complexity into our driver, since the window state is asynchronous to all other hardware accesses and not all of the window state information we need was directly available to us. In addition, applications expect to be able to mix Windows NT graphical device interface rendering and 3D OpenGL rendering in the same window. These two problems required us to add a double

Figure 5

(a) VISUALIZE fx^4 stacked frame buffer model. (b) Windows NT offscreen frame buffer model.

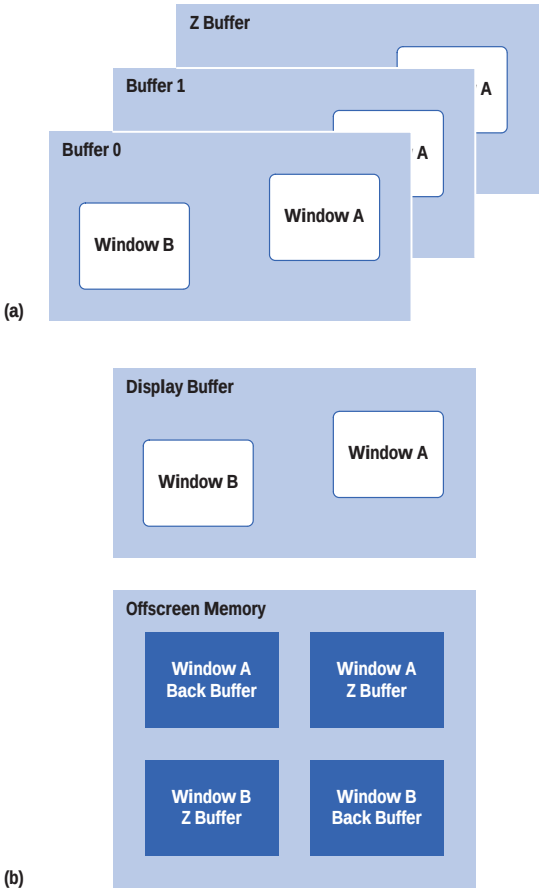
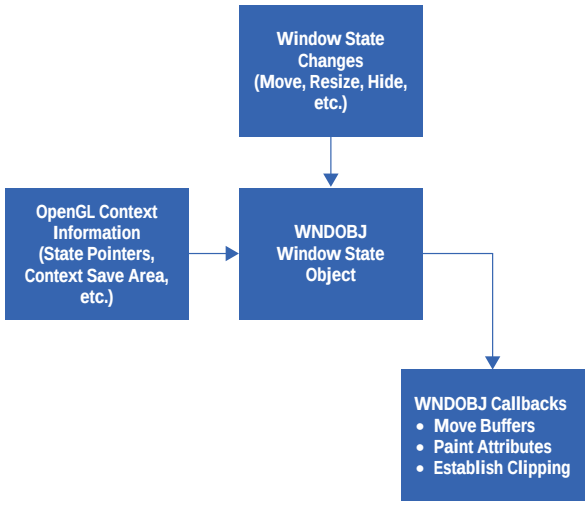


Figure 6

The components of a window object.



buffering mechanism that actually copies the physical back buffer bits into the displayed front buffer. This is significantly slower than the native per pixel double buffering of VISUALIZE fx[4]. However, it fits better into the Windows NT model and enables all applications to run. We still enable the native method for applications and benchmarks that work correctly with it, since it is significantly faster.

Performance

A fifth challenge for the team was performance. In the graphics workstation market, performance is usually the main differentiator. The most popular single measure of performance in the PC graphics market is the OPC Viewperf benchmark known as CDRS-03.[1] By July, 1997, we had achieved a CDRS-03 rating of 74—a performance level that exceeded all known competitors. This met our goals set at the beginning of the project. However, we were aware that the hardware was capable of supporting much higher performance. With a goal in mind of a SIGGRAPH 97 announcement in August, we redesigned the device driver. The redesign optimized certain paths through the driver, enabling much higher performance for this benchmark and for important applications such as Unigraphics and Structural Dynamics Research Corporation (SDRC). As a result, we were able to announce a CDRS-03 rating of over 100 at SIGGRAPH 97.

In addition to benchmark performance, the team focused on application performance because it is typically this measure that determines whether a customer will buy the product. We obtained a variety of in-house applications and built up expertise in running the applications. We also obtained data sets that represented typical customer workloads and adjusted various performance parameters (such as display list size) to maximize performance for the benchmark. Using this technique, the performance with some data sets was up to 100 times faster.

Conclusion

With VISUALIZE fx[4], Hewlett-Packard has the fastest Windows NT graphics on the market.[1,2,3] Integrated into the HP Kayak XW platform, the graphics device and its successors will help Hewlett-Packard maintain its market leadership.

References

1. *CDRS (CDRS-03) Results*, OpenGL Performance Characterization Project:
http://www.specbench.org/gpc/opc/opc.cdrs.html

2. *Advanced Visualizer (AWadvs-01) Results*, OpenGL Performance Characterization Project:
http://www.specbench.org/gpc/opc/opc.AWadvs.html

3. *Data Explorer (DX-03) Results*, OpenGL Performance Characterization Project:
http://www.specbench.org/gpc/opc/opc.dx.html

▶ Go to Next Article

▶ Go to Journal Home Page