
PowerStorm 4DT: A High-performance Graphics Software Architecture

The PowerStorm 4DT series of graphics devices established DIGITAL as the OpenGL performance leader for mid-range workstations, both on the DIGITAL UNIX and the Windows NT operating systems. Achieving this level of success required combining the speed of the Alpha microprocessor with the development of an advanced graphics subsystem architecture focused on exceptional software performance. The PowerStorm 4DT series of graphics adapters uses a modified direct-rendering technology and the Alpha CPU to perform geometry and lighting calculations.

Benjamin N. Lipchak
Thomas Frisinger
Karen L. Bircsak
Keith L. Comeford
Michael I. Rosenblum

The PowerStorm 4D40T, 4D50T, and 4D60T mid-range graphics adapters from DIGITAL have exceeded the performance of all OpenGL graphics devices costing as much as \$25,000. In addition, these products achieved twice the price/performance ratio of competing systems at the time they were announced.

The PowerStorm 4DT series of mid-range graphics devices was developed in 1996 to replace the company's ZLX series. In its search for a vendor to replace the graphics hardware, DIGITAL found Intergraph Systems Corporation. This company had been designing three-dimensional (3-D) graphics boards for a few years and was then on its second-generation chip design. The schedule, cost, and performance of Intergraph's new design matched our project's target goals. Intergraph was building software for the Windows NT operating system on its Intel processor-based workstations, but was not doing any work for the UNIX operating system or the Alpha platform.

The goals of the PowerStorm 4DT project were to develop a mid-range graphics product powered by the Alpha microprocessor that would lead the industry in performance and price/performance.

This paper describes the competitive environment in the graphics industry at the conception of the PowerStorm 4DT project. It discusses our design decisions concerning the graphics subsystem architecture and performance strategy. The paper concludes with a performance summary and comparison in the industry.

Competitive Analysis

Overall performance of today's mid-range workstations is markedly better than that of just two years ago. This improvement is largely due to the dramatic increases in CPU speeds, both in the number of instructions executed per clock cycle and the number of clock cycles per second. Without trivializing the efforts of the CPU architects, such year-over-year increases in CPU performance have become the trend of the last decade, especially with the Alpha microprocessor.

More astounding is the central role that the graphics component of the workstation is playing in defining the overall performance of the workstation. We are in the age of visual computing. Whether or not an application requires 3-D graphics, even the most primitive applications often rely on a graphical user interface (GUI). As such, the graphical components of today's system-level benchmarks now carry significant weight.

More importantly, a prospective buyer often looks at results from standard graphics benchmarks as a general indication of a machine's overall performance. In the computer-aided design/computer-aided manufacturing (CAD/CAM) market, a customer typically buys a workstation to run a set of applications that has a large 3-D component. Performance is measured by how fast a workstation can create and manipulate 3-D objects. For the most part, this performance is determined wholly by the graphics subsystem. The hardware components of the graphics subsystem, however, vary from vendor and may or may not include the CPU.

Performance Metrics

Simply stated, the primary goal of the PowerStorm 4DT graphics device series was to provide the fastest mid-range OpenGL graphics performance while offering the best price/performance ratio. OpenGL is the industry-standard 3-D graphics application programming interface (API) and associated library that provides a platform-independent interface for rendering 3-D graphics.¹

Quantifying performance can be an elusive goal. Product managers in our Workstation Graphics Group chose two metrics to compare the performance of the PowerStorm 4DT adapter to our competitors' products. The first metric was performance on the industry-standard OpenGL Viewperf benchmark, Conceptual Design and Rendering Software (CDRS).² This benchmark was chosen for its universal acceptance in the CAD/CAM and process control markets. When buyers compare graphics performance of two systems running OpenGL, the Viewperf scores are among the first measurements they seek. The second measurement was performance on the Pro/ENGINEER application from Parametric Technology Corporation (PTC).

The CDRS benchmark, as shown in Figure 1, was established by the OpenGL Performance Characterization (OPC) organization as one of several Viewperf viewsets. It emulates the variety of operations a user typically executes when running a CAD/CAM application. Specifically, this benchmark uses a series of tests that rotate a 3-D model on the screen in a variety of modes, including wireframe vectors, smooth-shaded facets, texturing, and transparency. Performance is measured by how many frames per second can be generated. Higher frame rates equate to faster and smoother rotations of the model. Each test carries a

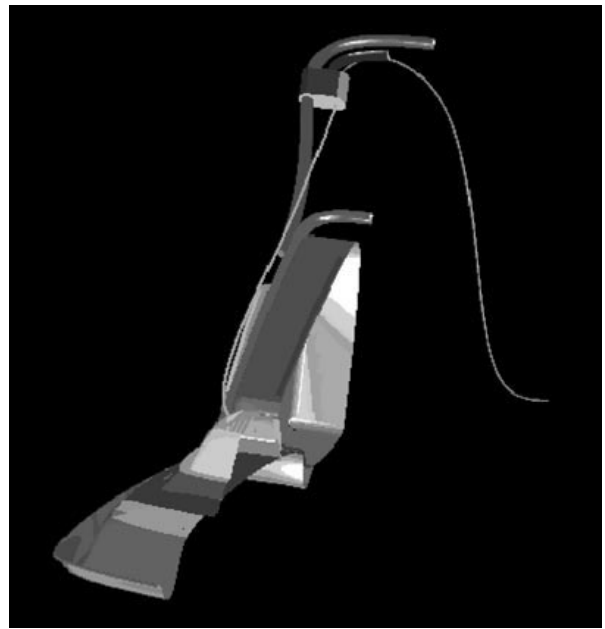


Figure 1
CDRS Viewperf Benchmark of OpenGL Performance

weight determined to roughly correspond to how important that operation is in a real-world CAD/CAM package. The test results are geometrically averaged to produce a composite score. This single number is a representation of the graphics performance of any given system.

Although standard benchmarks are good performance indicators, they cannot substitute for actual performance on an application. To ensure that the PowerStorm 4DT adapter realized exceptional real-world performance, the second metric chosen was the CAD/CAM industry's market share leader, the Pro/ENGINEER application. PTC provides the industry with a set of playback files called trail files. As shown in Figure 2, each file contains a recording of a session in which a user has created and rotated a 3-D part. The recordings typically have large wireframe and smooth-shading components and little or no texture mapping. Performance is measured by how quickly a system can play back a trail file. The CDRS benchmark stresses only the graphics subsystem, but the Pro/ENGINEER trail file stresses the CPU and the memory subsystem as well.

Graphics Hardware Standards

In 1996, Silicon Graphics Inc. (SGI) captured the mid-range graphics workstation market with its Indigo2 Maximum IMPACT graphics subsystem powered by the MIPS R10000 microprocessor. DIGITAL, Sun Microsystems, and International Business Machines (IBM) Corporation had yet to produce a product with the performance SGI offered; instead, they competed in the low to lower mid-range graphics arena.

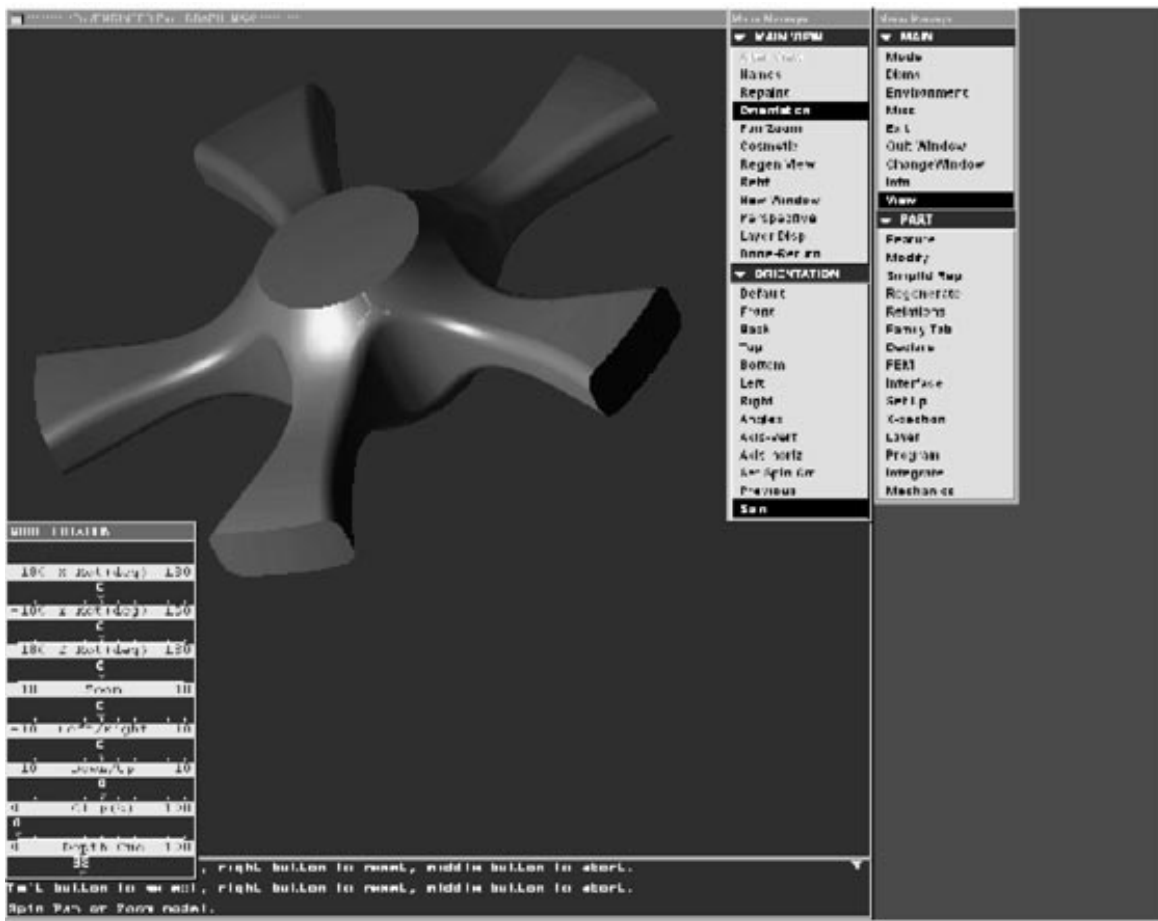


Figure 2
Screen Capture from the Pro/ENGINEER Trail File Used to Stress the PowerStorm 4DT Series

Hewlett-Packard was notably absent from either bracket due to its lack of a mid-range workstation with OpenGL graphics capability. Mid-range workstations can be loosely classified as costing from \$15,000 to \$40,000. Graphics performance in this price range differs, sometimes dramatically, from vendor to vendor.

Considering only raw graphics hardware performance, a vendor had to offer a certain level of performance to be competitive with SGI. By 1996 standards, a competitive device needed to be capable of achieving the following:

- 1 million Gouraud-shaded, 25-pixel, Z-buffered triangles per second
- 2 million flat-shaded, antialiased, 10-pixel vectors per second
- Trilinear, mipmapped, texture fill rates of 30 megapixels per second
- 24-bit deep color buffer
- 4-bit overlay buffer
- 4-MB dedicated or unified texture memory

- Dedicated hardware support for double buffering and Z-buffering
- Screen resolution of 1,280 by 1,024 pixels at 72 hertz

In 1996, the PowerStorm 4D60T, the most advanced graphics adapter in the new series, was capable of the following:

- 1.1 million Gouraud-shaded, 25- to 50-pixel, Z-buffered triangles per second
- 2.5 million flat-shaded, antialiased, 10-pixel vectors per second
- Trilinear, mipmapped, texture fill rates of greater than 30 megapixels per second
- 32-bit deep color buffer
- 8-bit overlay buffer
- 0- to 64-MB dedicated texture memory
- Dedicated hardware support for double buffering (including overlay planes) and Z-buffering
- Screen resolution up to 1,600 by 1,200 pixels at 76 hertz

It is important to understand that these are hardware maximums. The interesting work is not in achieving these rates under the best of conditions, but in achieving these rates under most conditions. To reiterate, building hardware that can theoretically perform well and building a system that performs well in benchmark applications are two distinctly different goals. The latter requires the former, but the former in no way guarantees the latter.

Different viewpoints on the best way to provide the highest level of performance have divided the industry into several camps. Workstation vendors must decide which approach best exploits the competitive advantages of their systems. In the mid-range workstation market, our graphics philosophy is decidedly different from that of our competitors. For the most part, DIGITAL is alone in its choice of a CPU-based, direct-rendering graphics architecture.

In the next section, we discuss the various graphics design architectures in the industry, focusing on the design of the PowerStorm series and comparing it with SGI's approach.

Graphics Subsystem Architectures

The two essential choices for graphics subsystem design are deciding between indirect and direct rendering and choosing whether the CPU or an application-specific integrated circuit (ASIC) performs the geometry and lighting calculations. In this section, we discuss the advantages and disadvantages of both rendering schemes and calculation devices and explore designers' decisions for graphics subsystem architectures.

By order of occurrence, 3-D graphics can be divided into three stages: (1) transfer of OpenGL API calls to the rendering library, (2) geometry and lighting, and (3) rasterization. In the next section, we compare direct and indirect image rendering.

Direct Versus Indirect Rendering

Before the popularization of the Windows NT operating system and the personal computer, almost all graphics workstations used the X Window System or a closely related derivative. The typical X Window System implementation is a standard client-server model.³ An application that draws to the screen requests the X server to manage the graphics hardware on its behalf.

The graphics API, either Xlib for two-dimensional (2-D) applications or OpenGL for 3-D, was the functional breaking point. Traditionally, client applications would make graphics API calls to do drawing or another graphics-related operation. These calls would be encoded and buffered on the client side. At some point, either explicitly by the client or implicitly by the API library, the encoded and buffered requests would be flushed to the X server. These commands would

then be sent to the X server over a transport such as the Transmission Control Protocol/Internet Protocol (TCP/IP), a local UNIX domain socket, or local shared memory.

When the requests arrived at the X server, it would decode and execute them in order. Many requests would then require the generation of commands to be sent to the hardware. This client-server model was named indirect rendering because of the indirect way in which clients interacted with the graphics hardware.

Direct rendering is a newer method often employed in the design of high-end graphics systems.^{4,5} In this scheme, the client OpenGL library is responsible for all or most 3-D rendering. Instead of sending commands to the X server, the client itself processes the commands. The client also generates hardware command buffers and often communicates directly with the graphics hardware. In this rendering scheme, the X server's role is greatly diminished for 3-D OpenGL requests but remains the same for 2-D Xlib requests.

The designers chose to support direct rendering for the PowerStorm 4DT adapter. Direct rendering offers considerably better performance than indirect rendering. Note, however, direct rendering does not preclude indirect rendering. All devices that support direct rendering under the X Window System also support indirect rendering.

In the following subsections, we discuss the advantages and disadvantages of direct and indirect rendering. We also explain the impetus for making the PowerStorm 4DT adapter the first graphics device from DIGITAL capable of direct rendering.

Indirect Rendering One advantage of indirect rendering that should never be underestimated is its proven track record. This technology is widely accepted and understood. It offers network transparency, which means a client and server need not reside on the same machine. A client can redirect its graphics to any machine running an X server as long as the two machines are connected on a TCP/IP network. This model worked well until faster CPUs and graphics devices were developed. The protocol encode, copy, and decode overhead associated with sending requests to the server became a bottleneck.

The increased use of display lists provided an intermediate solution to this problem. Display lists are a group of OpenGL commands that can be sent to the X server once and executed multiple times by referencing the display list ID instead of sending all the data each time. Display lists dramatically reduced communication overhead and returned graphics to the point at which communication to the X server was no longer the bottleneck.

Unfortunately, display lists had significant disadvantages. Once defined, they could not be modified. To achieve performance using indirect rendering, almost

all OpenGL commands had to be collected into display lists. This caused resource problems because display lists could be quite large and had to be stored in the X server until explicitly deleted by the client. Probably the greatest disadvantage was that display lists were generally awkward for application programs to use. Application programmers prefer the more straightforward method of immediate-mode programming by which commands are called individually. For these reasons, indirect rendering proved to be insufficient, even with the advent of display lists.

Direct Rendering The PowerStorm 4DT project team was committed to designing a product with leadership performance for both the display-list-mode and immediate-mode rendering. The designers realized early that they would have to adopt direct rendering to address the performance problems with immediate-mode indirect rendering.

As mentioned earlier, the philosophy behind classical direct rendering is that each client handles all OpenGL processing, creates a buffer of hardware commands for the device, and then sends the commands to the device without any X server interaction. This model has several drawbacks. First, access to the graphics hardware is difficult to synchronize between clients and the X server. Second, windows and their properties such as position and size have to be maintained by the clients, which also requires a complex synchronization design. SGI used this model for its IMPACT series of graphics devices.

The PowerStorm 4DT designers took a more conservative approach, based largely on the same model. One fundamental difference is that each client generates hardware command buffers in shared memory. The client then sends requests to the X server telling it where to locate the hardware commands. The X server sets up the hardware to deal with window position and size and then initiates a direct memory access (DMA) of the hardware command buffer to the graphics device. Essentially, the X server becomes an arbitrator of hardware buffers. This approach worked quite well, because the X server was the logical place for synchronization to occur and it already maintained window properties. We were able to have all the performance benefits of classical direct rendering without the pitfalls.

One implication of direct rendering is that the client and the server have to be on the same physical machine. When first evaluating direct rendering, designers were curious to determine how often our customers used this configuration; that is, did most users perform their work and display their graphics on the same computer? Our surveys showed that more than 95 percent of users did display their graphics locally. The remaining 5 percent rarely cared about performance. Today, this may seem obvious; two years ago, it could not be assumed.

Direct rendering offered a huge performance improvement to nearly all our customers. The performance gains were two to four times the performance of indirect rendering.

Direct-rendering 2-D Most graphics device implementations use direct rendering only for OpenGL, because indirect rendering of immediate-mode OpenGL is protocol rich. As mentioned previously, the transfer of this protocol to the X server can be quite expensive. One interesting aspect of our design is its support for direct rendering of 2-D Xlib calls.

Other graphics vendors consider 2-D performance important only for 2-D benchmarks. These benchmarks, which largely stress the graphics hardware's ability to draw 2-D primitives quickly, can generate a lot of work for the hardware with relatively few requests. Unlike 3-D, these requests do not need much geometry processing before they can be sent to the hardware. This means that very little protocol is needed to saturate the hardware. As long as the protocol generation does not produce a bottleneck, indirect rendering performs as well as direct rendering. In addition, given that OpenGL benchmarks like CDRS have almost no 2-D component, it seems reasonable to conclude that indirect-rendered 2-D should suffice.

Benchmarks often are not sufficiently representative of real applications, especially when they isolate 2-D and 3-D operations. CAD/CAM applications typically have a substantial 2-D GUI, which interacts closely with the 3-D components of the application. A benchmark that exercises both 2-D and 3-D by emulating a user session on an application will provide results that more accurately reflect the performance witnessed by an end user. These benchmarks simply measure how long it takes to complete a session, so both 3-D and 2-D performance impact the overall score.

Our research showed that with a highly optimized OpenGL implementation, in many cases it was no longer the 3-D components that slowed down a benchmark, but the 2-D components. Further examination revealed that it was the same protocol bottleneck evident with indirect-rendered OpenGL. Applications were generating relatively small drawing operations with many drawing attribute changes intermixed, such as draw line, change color, draw line, change color, and so forth. This type of request stream tends to generate tremendous amounts of protocol, unlike 2-D benchmarks that rarely change drawing attributes.

Accordingly, 2-D direct rendering presented itself as the logical solution. With the direct-rendering infrastructure and design already in place, developers simply needed to extend it for 2-D/Xlib. This required the development of two additional libraries: the Vectored X library and the Direct X library (unrelated to Microsoft's DirectX API).

The Vectored X library replaced the preexisting Xlib. It allows devices that support direct rendering to vector, or redirect, Xlib function calls to direct-rendering routines instead of generating the X protocol and sending it to the X server. If a graphics device does not support direct rendering, it defaults to the generic protocol-generating routines. It is important to understand that this is a device-independent library responsible only for vectoring Xlib calls to the appropriate library.

The Direct X library, on the other hand, is a device-dependent library. It contains all the vectored functions that the Vectored X library calls when the device supports direct rendering. This library operates in much the same way as the direct-rendering OpenGL library. It processes the requests and places graphics hardware commands in a shared memory buffer. The X server later sends the buffer to the graphics device by DMA.

The entire functionality of the X library is not implemented through direct rendering for several reasons. In many cases, a shared resource resides in the server (e.g., the X server performs all pixmap rendering). In other cases, the hardware is not directly addressable by the client (e.g., the X server handles all frame buffer reads). Often the client does not have access to all window information that the server maintains (e.g., the X server handles all window-to-window copies). Fortunately, these operations are either not frequently used, not expected to be fast, or easily saturate the hardware.

Further details of the Vectored X library and Direct X library are beyond the scope of this paper. The concept of direct-rendered 2-D, however, is sound. It has helped DIGITAL outperform other vendors on many application benchmarks that were largely focused on OpenGL but had significant 2-D components. Our 2-D direct-rendering technology has also enhanced 2-D performance and response time for the many thousands of exclusively 2-D applications for the X Window System.

Geometry and Lighting

The geometry and lighting phase can be performed by the host CPU or by a specialized, high-speed ASIC, which is typically located on the graphics device. Regardless of where these calculations take place, the general idea is that the user's vertices are transformed and lit, then fed to the rasterizer. Since the rasterizer is on the graphics device, choosing the host to do the geometry and lighting implies that the transformed and lit vertices are then sent across the bus to the rasterizer. The use of a specialized ASIC implies that the user's vertices are sent across the bus, transformed and lit by the custom ASIC, and then fed directly to the rasterizer. The information transferred across the bus is obviously different, but in terms of amount of data per vertex, it is approximately the same. Therefore, bus bandwidth does not become a deciding factor for either design.

Host CPU Geometry and Lighting Traditionally, DIGITAL has chosen the host CPU to perform the geometry and lighting calculations. The PowerStorm project designers chose this approach because of the Alpha microprocessor's exceptional floating-point speed, and because almost all 3-D calculations involve floating-point values. At the time this project was conceived, the only general-purpose, widely available processor capable of feeding more than 1 million transformed and lit vertices per second to the hardware was the Alpha CPU. An additional benefit of having the Alpha CPU do the work was an overall cost reduction of the graphics device. Custom ASICs are expensive to develop and manufacture.

Another important and related advantage is that our software becomes proportionally faster as clock speeds rise on available Alpha microprocessors. This results in a near linear performance increase without any additional engineering cost. For example, using the same software, a 500-megahertz (MHz) Alpha microprocessor is able to produce 25 percent more vertices per second than a 400-MHz Alpha microprocessor. Because of this, developers can write optimized Alpha code once and reuse it for successive generations of Alpha CPUs, reaping performance improvements with virtually no further invested effort.

It is obvious that rendering can proceed no faster than vertices can be generated. If the OpenGL library can transform and light only 750,000 vertices per second, and the graphics device can rasterize 1 million, the effective rendering rate will be 750,000. In this example, the OpenGL geometry and lighting software stages are the bottleneck. However, if the numbers were reversed, and the hardware could only rasterize 750,000 vertices while the OpenGL software provided 1 million, the rasterization hardware would become the bottleneck.

Thus far, we have discussed two potential bottlenecks: the OpenGL implementation itself and the rasterization hardware. The third and potentially most damaging bottleneck may be the client's ability to feed vertices to the OpenGL library. It should be clear that this is the top level of vertex processing. The OpenGL library can render no faster than the rate at which the client application feeds it vertices. Consequently, the rasterizer can render primitives no faster than the OpenGL library can produce them. Thus, a bottleneck in generating vertices for the OpenGL library will slow the entire pipeline. Ideally, we would like each level to be able to produce at least as many vertices as the lower levels can consume.

Clearly, the performance of the application, in terms of handing vertices to the OpenGL library, is a function of CPU speed. This is only an issue for applications that have large computation overhead before rendering. Currently, almost all graphics benchmarks have little or no computation overhead in getting ver-

tices to the OpenGL library. Most attributes are pre-computed, since they are trying to measure only the graphics performance and throughput. For the most part, this holds true for the traditional CAD/CAM packages. However, some emerging scientific visualization applications as well as some high-end CAD applications require significant compute cycles to generate the vertices sent to the OpenGL library. For these applications, only the DIGITAL Alpha CPU-based workstations can produce the vertices fast enough for interactive rates.

There are some potential disadvantages to this design. Namely, the CPU is responsible for both the application's and the graphics library's computations. If the application and the OpenGL implementation must contend for compute cycles, overall performance will suffer. Analysis of applications revealed that typical 3-D and 2-D graphics applications do internal calculations followed by rendering. Only under rare circumstances do the two processes mix with a substantial ratio. If the applications should start mixing their own processing needs with those of the OpenGL library, the notion of host-based geometry would need to be revisited.

Another potential disadvantage is the rate at which Alpha CPU performance increases versus the rate at which the rasterizer chip's performance increases. The emerging generation of graphics devices is capable of rasterizing more than 4 million triangles per second. It is unknown whether future generations of the Alpha CPU will be able to feed the faster graphics hardware.

ASIC-based Geometry and Lighting Performing geometry and lighting calculations with a custom ASIC on the graphics device is often referred to as OpenGL in hardware because most of the OpenGL pipeline resides in the ASIC. The OpenGL library is limited to handing the API calls to the hardware. SGI has adopted the ASIC-based approach for many generations of workstations and graphics devices. In this section, we discuss why this method works for them and its potential shortcomings.

SGI workstations use either the R4400 or the R10000 CPU developed by MIPS Technologies. Although these CPUs have good integer performance, their floating-point performance cannot generate the number of vertices that the Alpha CPU can. As a consequence, SGI has to use the custom-graphics ASIC approach. One advantage to the custom ASIC is the decoupling of graphics from the CPU. Since each can operate asynchronously, the application has full use of the CPU.

Typically, custom geometry ASICs, also known as geometry engines, perform better than a general-purpose CPU for several reasons. First, the custom ASIC must perform only a well-understood and limited set of calculations. This allows the ASIC designers to optimize their chip for these specific calculations, releasing them from the burden and complexity of general-purpose CPU design.

Second, the graphics engine and the rasterizer can be tightly coupled; in fact, they can be located on the same chip. This allows for better pipelining and reduced communication latencies between the two components. Even if the geometry engine and rasterizer are located on different chips, which is not at all uncommon, a much stronger coupling exists between the geometry engine and the rasterizer than does between the host CPU and rasterizer.

Third, geometry engines can yield high performance when executing certain display lists. The use of a display list allows an object to be quickly re-rendered from a different view by changing the orientation of the viewer and reexecuting the stored geometry. If the display list can fit within the geometry engine's cache, it can be executed locally without having to resend the display list across the bus for each execution. This helps alleviate the transportation overhead in getting the display list data over the bus to the graphics device. It is unclear how often this really happens since rasterization is typically the bottleneck. If the display list is filled with many small area primitives, however, its use can result in noticeable performance gains. Geometry engines often have a limited amount of cache. If an application's display list exceeds the amount of cache memory, performance degrades significantly, often to below the performance attainable without a geometry accelerator. Our research shows that display list sizes used by applications increase every year; therefore, cache size must increase at the same rate to maintain display list performance advantages.

The primary disadvantage of using custom ASICs to perform the geometry and lighting calculations is the expense associated with their design and manufacture. In addition, a certain risk is involved with their development: hardware bugs can seriously impact a product's viability. Fixing the bugs causes the schedule to slip and the cost to rise. Hardware bugs discovered by customers can be devastating. With host-based geometry, a software fix in the OpenGL library can easily be incorporated and distributed to customers.

A sometimes unrecognized disadvantage of dedicated geometry engines is that they are bound to fixed clock rates, with little room for scalability. Although this is true of most CPU designs, CPU vendors can justify the engineering effort required to move to a faster technology, because of competitive pressures and the larger volume of host CPU chips.

Rasterization

During the rasterization phase, primitives are shaded, blended, textured, and Z-buffered. In the early years of raster-based computer graphics, rasterization was done using software. As computer graphics became more prevalent, graphics performance became an issue. Because rasterization is highly computational and requires many accesses to frame buffer memory,

it quickly became the performance bottleneck. Specialized hardware was needed to accelerate the rasterization part of graphics. Fortunately, hardware acceleration of rasterization is well understood and is now the de facto standard. Today, nearly every graphics device has rasterization hardware. Even low-priced commodity products have advanced raster capabilities such as texture mapping and antialiasing.

In the next section, we relate our strategy for obtaining optimal graphics software performance from an Alpha processor-based system.

Performance Strategy

The goals of the PowerStorm 4DT program were largely oriented toward performance. Our strategy consisted of having a generic code path and then tuning performance where necessary using Alpha assembly and integrated C code.

Performance Architecture

The designers optimized the software performance of the PowerStorm 4DT series within the framework of a flexible performance architecture. This architecture provided complete functionality throughout the performance-tuning process, as well as the flexibility to enhance the performance of selected, performance-sensitive code paths.

In this context, code paths refer to the vertex-handling routines that conduct each vertex through the geometry, lighting, and output stages. Whereas most OpenGL API calls simply modify state conditions, these vertex routines perform the majority of computation. This makes them the most likely choices for optimization.

The Generic Path A solid, all-purpose code base written in C and named the generic path offers full coverage of all OpenGL code paths. The generic path incurs a significant performance penalty because its universal capabilities require that it test for and handle every possible combination of state conditions. In fact, under certain conditions, the generic path is incapable of driving the hardware at greater than 33 percent of its maximum rendering rate. The generic path assumes responsibility for the rare circumstances that are not deemed performance-sensitive and thus not worthy of optimization. It also acts as a safety net when high-performance paths realize mid-stride that they are not equipped to handle new, unanticipated conditions.

Multicompiled Speed of Light (SOL) Paths High-performance SOL paths provide greatly increased performance where such performance is necessary. Under prescribed conditions, SOL paths replace the generic path, yielding equivalent functionality with performance many times that of the generic path. SOL paths

were written for the combinations of state conditions exercised most frequently by the target applications and benchmarks.

The developers responsible for performance tuning designed two classes of SOL paths. First, they generated a large number of SOL paths by compiling a C code template multiple times. Whereas the generic path is composed of several routines, each corresponding to a single stage of the pipeline, a multicompiled SOL path integrates these stages into a monolithic routine. Each compilation turns on and off a different subset of state conditions, resulting in integrated paths for every combination of the available conditions. This multicompilation of integrated SOL paths yields the following benefits:

- The C compiler is allowed a broader overview of the code and can more wisely schedule instructions. In contrast, the generic path is composed of several individual stages. These relatively short routines do not provide the C compiler with enough space or enough scope to make informed and effective, instruction-ordering decisions. Multicompiling the various stages into a series of monolithic, integrated routines relieves each of these problems.
- The multicompilation assumes a fixed set of conditions for each generated path. This eliminates the need for run-time testing of these conditions during each execution of the path. Instead, such testing is necessary only when state conditions change. Validation, as this testing is called, determines which new path to employ, based on the new state conditions. With the great number and complexity of state conditions influencing this decision, validation can be an expensive process. Performing validation only in response to state changes, rather than for every vertex, results in significant performance gains.
- The SOL path coverage at least doubles every time that support for a new state condition is added to the template. Each new condition increases the number of combinations of conditions being multicompiled into SOL paths by a factor of two or more. An adverse side effect of this strategy is that the compile time and resulting library size will increase at the same rate as the SOL path coverage.

Assembly Language SOL Paths Hand-coded Alpha assembly language paths constitute the other class of high-performance SOL paths. These paths, designed specifically for extremely performance-sensitive conditions, require much more time and attention to produce. Taking advantage of the many features of the Alpha microprocessor transforms assembly language coding from a science into an art form.⁶ The Alpha assembly coders kept the following issues foremost in their minds:

- The 21164 and subsequent Alpha microprocessors are capable of quad-issuing instructions, which means that as many as four instructions can be initiated during each cycle. The combination of instructions that may be issued, however, depends on the computational pipelines and other resources employed by each instruction. Coders must carefully order instructions to gain the maximum benefit from the multiple-issue capability.
- As a consequence of the above restrictions, integer and floating-point operations must be scheduled in parallel. With few exceptions, only two floating-point and two integer instructions can be issued per cycle. Efficiency in this case requires not only local instruction-order tweaking but also global changes at the algorithmic level. Integer and floating-point operations must be balanced throughout each assembly routine. If a particular computation can be easily performed using either integer math or floating-point math, the choice is made according to which pipeline has more free cycles to use.
- Register supply is another factor that affects the design of an assembly language routine. Although the Alpha microprocessor has a generous number of registers (32 integer and 32 floating-point), they are still considered a scarce resource. The coder must organize the routine such that some calculations complete early, freeing registers for reuse by subsequent calculations.
- The crucial performance aspect of assembly coding is transporting the data where and when it is needed. The latency of loading data from main memory or even from cache into a register can easily become any routine's bottleneck. To minimize such latencies, load instructions must be issued well in advance of a register's use; otherwise, the pipeline will stall until the data is available. In an ideal architecture with an infinite quantity of registers, all loads could be performed well in advance. Unfortunately, due to the scarce amount of free registers, the number of cycles available between loading a register and its use is frequently limited.

Each of these assembly language programming considerations requires intense attention but yields unmatched performance.

Performance Tuning

After reviewing benchmark comparisons and recommendations from independent software vendors, we determined which areas required performance improvement. We approached performance tuning from two directions: either by increasing SOL path coverage or improving the existing SOL code.

Increasing SOL path coverage was the more straightforward but the more time-consuming approach. If an SOL path did not exist for a specific condition, a new one would have to be written. Adding a new option to the multicompile template required a significant effort in some cases. Implementing a new assembly language SOL path always required significant effort.

Improving the performance of an existing SOL path required an iterative process of profiling and recoding. We employed the DIGITAL Continuous Profiling Infrastructure (DCPI) tools to analyze and profile the performance of our code.⁷ DCPI indicated where bottlenecks occurred and whether they were due to data cache misses, instruction slotting, or branch misprediction. This information provided the basis for obtaining the maximum performance from every line of code.

Development of 3-D Graphics on Windows NT

At the start of the PowerStorm 4DT project, the Windows NT operating system was an emerging technology. The DIGITAL UNIX platform held the larger workstation market share, while Windows NT accounted for only a small percentage of customers. For that reason, designers targeted performance for applications running on DIGITAL UNIX and developed 3-D code entirely under that operating system.

Nevertheless, we recognized the potential gains of developing 3-D graphics for the Windows NT system. One of the company's goals was to be among the first vendors to provide accelerated OpenGL hardware and software for Windows NT.

With a concerted effort and a few compromises, the team developed the PowerStorm 4DT into the fastest OpenGL device for Windows NT, a title that was held for more than 18 months. To achieve this capability, the designers made the following key decisions:

- To write code that was portable between the DIGITAL UNIX and Windows NT systems.
- To dedicate two people to the integration of the DIGITAL UNIX-based code into the Windows NT environment. Most OpenGL code was operating-system independent, but supporting infrastructure needed to be developed for Windows NT.
- To use Intergraph's preexisting 2-D code and to avoid writing our own. Intergraph provided us with a stable 2-D code base for Windows NT. This code base had room for optimization, but further optimization of the 3-D code took precedence.
- To ship the graphics drivers for DIGITAL UNIX first, and the graphics drivers for Windows NT three months later. In this way, we allowed the DIGITAL UNIX development phase to advance unimpeded by the efforts to port Windows NT.

Results and Conclusion

In August of 1996, the PowerStorm 4D60T graphics adapter was best in its price category with a CDRS performance number of 49.01 using a 500-MHz Alpha processor. It yielded a new price/performance record of \$321 per frame per second. At the same time, SGI attained a CDRS number of only 48.63 on a system costing nearly three times as much.

Figure 3 shows the relative performance of the PowerStorm 4D60T for four of the major Viewperf benchmarks. The viewsets are based on the following applications: CDRS, a computer-aided industrial design package from PTC; Data Explorer (DX), a scientific visualization package from IBM; DesignReview (DRV), a model review package from Intergraph; Advanced Visualizer, a 3-D animation system from Alias/Wavefront (AWadvs).

The PowerStorm 4D60T mid-range graphics adapter easily outperformed the Indigo2 High IMPACT system from SGI by a wide margin and even surpassed SGI's more expensive graphics card, the Indigo2 Maximum IMPACT, by a factor of more than 2:1 in price/performance on these benchmarks. Figure 4 shows that the PowerStorm 4D60T was the performance leader in three of the four benchmarks. SGI has yet to produce a graphics product in this price range that outperforms the PowerStorm 4D60T.

Acknowledgments

The authors would like to acknowledge the many other engineers who made the PowerStorm 4DT project a successful one, including Monty Brandenburg, Shih-Tang Cheng, Bill Clifford, John Ford, Chris Kniker, Jim Rees, Shuhua Shen, Shree Sridharan,

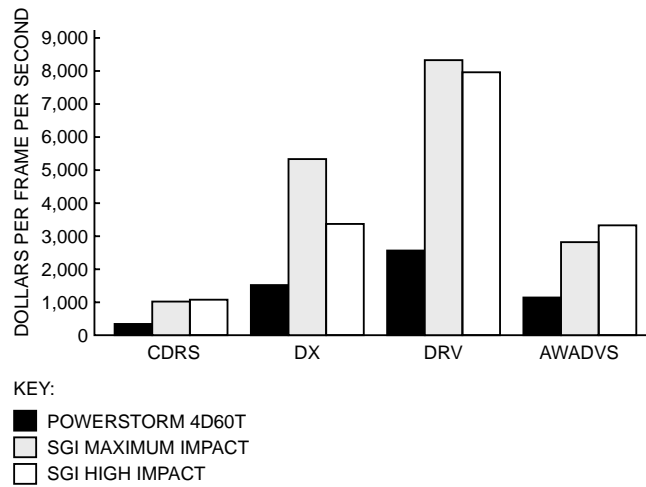


Figure 3

Price/Performance Comparison of Graphics Adapters on Viewperf Benchmarks

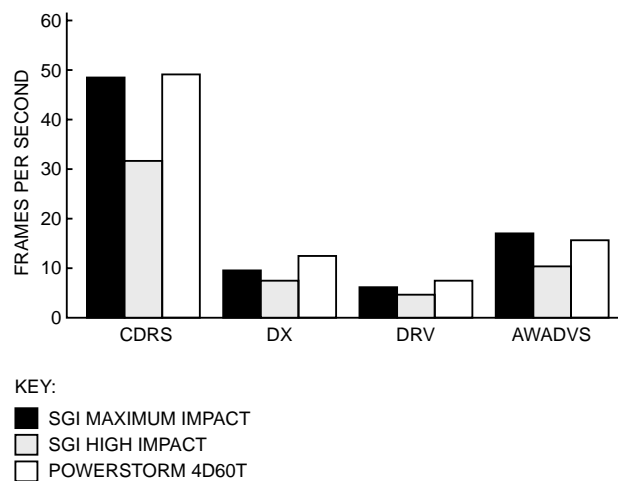


Figure 4

Performance of Graphics Adapters on Viewperf Benchmarks

Bruce Stockwell, and Mark Yeager. We would also like to thank the Graphics Quality Assurance Group and the Workstation Application Benchmarking Group for their unending patience and cooperation.

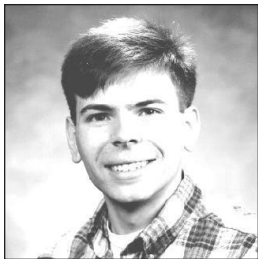
References

1. M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification* (Mountain View, Calif.: Silicon Graphics, Inc., 1995).
2. The OpenGL Performance Characterization Project, <http://www.specbench.org/gpc/opc.static>.
3. R. Scheifler and J. Gettys, *X Window System* (Boston: Digital Press, 1992).
4. H. Gajewska, M. Manasse, and J. McCormack, "Why X Is Not Our Ideal Window System," *Software Practice and Experience* (October 1990).
5. M. Kilgard, "D11: A High-Performance, Protocol-Optional, Transport-Optional Window System with X11 Compatibility and Semantics," *The Ninth Annual X Technical Conference*, Boston, Mass. (1995).
6. R. Sites and R. Witek, *Alpha AXP Architecture Reference Manual* (Boston: Digital Press, 1995).
7. J. Anderson et al., "Continuous Profiling: Where Have All the Cycles Gone?" *The 16th ACM Symposium on Operating Systems Principles*, St. Malo, France (1997): 1-14.

General References

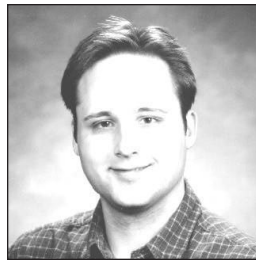
- J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics Principles and Practice* (Reading, Mass.: Addison-Wesley, 1993).
- M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide* (Reading, Mass.: Addison-Wesley, 1997).

Biographies



Benjamin N. Lipchak

Benjamin Lipchak joined DIGITAL in 1995 to develop software for the PowerStorm 4DT graphics adapter and later developed 3-D software for the PowerStorm 4D30T project. A senior software engineer in the Workstation Graphics Group, he is currently leading the software effort of a new graphics project. Benjamin received B.S. (highest honors) and M.S. degrees in computer science from Worcester Polytechnic Institute. He is the recipient of the Salisbury Award in Computer Science.



Thomas Frisinger

Tom Frisinger was a senior software engineer in the Workstation Graphics Group at DIGITAL for three years. During that time, he contributed to nearly all aspects of the PowerStorm 4DT project. As a member of the core software engineering team, he helped develop software for the 4D40T, 4D50T, and 4D60T graphics adapters as well as the 4D30T and 4D51T models. He was also part of the core software design team for the 4D31T graphics accelerator. Tom is currently doing research and development in PC graphics for ATI Research, Inc.



Karen L. Bircsak

As one of the developers of the PowerStorm 4DT graphics adapter, Karen Bircsak designed and implemented enhancements to the X library and contributed to other software development areas. A principal software engineer in the Workstations Graphics Group, Karen is currently working on supporting new graphics hardware. Prior to joining DIGITAL in 1995, she held software engineering positions at Concurrent Computer Corporation and Data General Corporation. She earned a B.S. in computer science and engineering from the University of Pennsylvania in 1984 and an M.S. in computer science from Boston University in 1990.



Keith L. Comeford

Keith Comeford is a principal software engineer in the Workstation Graphics Development Group. He is currently working on the next generation of graphics cards and accelerators for DIGITAL. Keith was the project leader for the Windows NT drivers for the PowerStorm 4D40T/50T/60T graphics cards. In previous project work, Keith contributed significantly to the GKS and PHIGS implementations in a variety of capacities from developer to project leader for more than 10 years. Keith joined DIGITAL in 1983 after receiving a B.S. in computer science from Worcester Polytechnic Institute.



Michael I. Rosenblum

Mike Rosenblum is a consulting software engineer at DIGITAL and the technical director for the Workstations Business Segment Graphics Group. He was the project leader and architect of the PowerStorm 4DT series and implemented some of its 2-D DDX code. Currently, he is managing two graphics projects and consulting to the company on graphics-related issues. Mike joined DIGITAL in 1981, to work on the terminal driver in the VMS Engineering Group. Later he helped design the company's first workstations. He has a B.S. in computer science from Worcester Polytechnic Institute and is a member of the ACM.