In spite of the continuing presence of hype and misrepresentation in the Windows graphics controller industry, discovering the true performance of a graphics subsystem is not very difficult. With a grasp of controller architecture, a sense of history, and a small suite of tests, you can determine the facts quite readily.
Chapter 1. Introduction

Graphics performance can make or break a system running Microsoft Windows. Everyone, including Microsoft, recognizes the need for high-performance graphics in a properly configured Windows system. *

This need, once recognized, created a new market for high-performance graphics controllers. Vendors immediately put forth products that had high Windows performance—or that they claimed had high Windows performance. Facing year-long chip development cycles, some vendors allowed the claims for their current products to outstrip their actual performance, unless viewed through a pair of vendor-supplied rose-colored glasses.

This in turn sparked a series of heated editorials and feature articles in the trade press, pointing out the prevalence of “benchmark cheating,” and making claims, of varying levels of accuracy, as to the level of honesty in the industry, or the degree to which it was possible to acquire honest performance measurements in an industry with a high level of flim-flam.

Our motivation for presenting this white paper is three-fold:

1. to remind people that some products have much higher performance than others,
2. to point out that not all vendors make exaggerated claims, and
3. to show you how to use benchmarking to separate the performers from the pretenders.

* For example, Microsoft invited eight graphics controller vendors to its recent Windows Hardware Engineering Conference, asking them to describe recent developments in the state of the art in graphics components.

This white paper describes how Windows graphics works, how hardware can make Windows graphics run faster, and how to measure the performance of such hardware with a high degree of confidence. All of the examples are based on recent experience. Where possible, examples are drawn from real, commercially available products, although we do not identify such products by name.

A NOTE ABOUT BENCHMARK CHEATING

Unfortunately, it has become difficult to talk about Windows graphics performance without spending a lot of time talking about benchmark cheating (or “benchmark-aware drivers,” as some people prefer to call them).

We define Windows benchmark cheating as follows: intentionally adding features to device drivers that improve benchmark scores without improving the performance of real applications.

This definition only singles out blatant benchmark cheating. Some people include additional behavior as cheating (such as “features that improve benchmark performance more than they improve application performance”), but we have found it simpler to neglect such grey areas and point out only cases which are completely unambiguous.

The harm of benchmark cheating is obvious: its sole purpose is to fool people into making decisions based on false data.

This paper gives specific advice on how to avoid the snares set by benchmark cheaters.
Chapter 2. Windows Graphics

2.1. Graphics Primitives

Windows draws everything with a handful of graphics primitives, which are illustrated in figure 1. These graphics primitives are the only things that Windows itself knows how to draw.

Application programs may allow you to draw, say, Bezier curves, but Windows 3.1 does not actually know how to draw such curves. The application breaks the user-requested form into a form Windows can understand (probably a polyline with a large number of very short line segments), and Windows in turn hands this series of drawing commands to the display driver.

Thus, the set of “interesting” graphical hardware enhancements is limited by the set of Windows graphics primitives. A chip that draws Bezier curves is useless in a Windows 3.1 environment, since Windows 3.1 cannot use this functionality.

2.2. Hardware vs. Display Drivers

The display driver is a piece of custom software that translates Windows GDI (graphical device interface) calls into commands that draw images on the display. Windows never examines the frame buffer directly, and never manipulates the frame buffer controller directly. Instead, it makes calls to the display driver, which deals with the actual hardware.

Since hardware capabilities vary enormously, Windows is very flexible about working with different controllers. The first line of defense is the GDIINFO structure, which is a list of the driver’s capabilities. If a driver does not support certain functions, Windows synthesizes these functions with low-level calls.

For example, if a driver does not support circle-drawing, Windows draws circles by other means, typically by drawing a great many short horizontal line segments (the scanline function). This allows Windows to take advantage of accelerated features while still supporting primitive display architectures.

While there are many interesting special cases, the vast bulk of the performance that can be extracted from Windows lies in only two areas:

1. raw frame buffer bandwidth, and
2. direct acceleration of Windows primitives.

These topics are covered in the next chapter.
2.2. Hardware vs. Display Drivers, continued

1. **BitBlt.** Bit maps are sent to the display with a host-to-screen BitBlt (bit block-transfer) call to put them there originally, and moved from one place to another with a screen-to-screen BitBlt call. Full-color host-to-screen bit maps are not accelerable, and depend on the bandwidth of the host and frame buffer controllers for speed. Screen-to-screen BitBlt is done by a BitBlt engine, which is present on every accelerated display controller. Windows supports some caching of bit maps, patterns, the area overwritten by menus, etc.; these bit maps are stored in their machine representation in off-screen memory, preventing multiple format conversions and host-to-screen transfers.

2. **Rectangle.** Windows has a special call to create non-rotated rectangles. Accelerators with a polygon engine and a pattern RAM can draw solid or patterned rectangles (and other polygons) at full frame-buffer bandwidth. Most accelerators, though, use a special case of BitBlt to draw rectangles at no more than one-third this speed.

3. **Polyline.** Connected line segments make up a polyline, the simplest case of which is the single line segment. Accelerators with full polyline support can draw a new line segment after the host transfers just one x,y pair. Windows uses the concept of “pens,” which specify line color, line width, end cap style, and dash styles. Few, if any, accelerators support anything but one-pixel-wide patterned lines in hardware, and many do not support even that. If the accelerator cannot draw a particular line, Windows creates it with more primitive functions (typically a series of horizontal scan-line calls), at substantial speed penalty.

4. **Polygon.** Windows polygons can be filled with solid colors or “brushed” with patterns. Their perimeters have the same pen properties as polylines. Accelerators with polygon engines can draw and fill patterned polygons at full frame-buffer bandwidth. If an accelerator lacks a polygon engine, Windows substitutes a series of horizontal scan-line calls, which will incur a tremendous speed penalty.

5. **Curves.** Windows supports circles, circular arcs, chords, vertically and horizontally aligned ellipses, elliptical arcs, and pie wedges (but not Bézier curves or parametric quadratic polynomials, the workhorses of advanced graphics). No hardened graphics processor supports any of these functions, but circles, at least, are typically supported by microcoded controllers. If an accelerator does not support these functions, Windows substitutes horizontal scan line calls or polylines. (As these curves are not a suitable set for general-purpose graphics, the lack of a curve-drawing engine has less impact on overall speed than one might suppose.)

6. **Text.** Windows supports internal device fonts, but the ever-increasing number of fonts in use makes putting fonts in display hardware inadvisable. Instead, Windows provides bit maps of the individual characters to the driver. Windows also supports font-caching by the device driver, which allows characters to be saved in their machine representation in driver or off-screen memory. This can double text speed. Two-color (foreground/background) host-to-screen text transfer can be accelerated via color expansion, which converts a monochrome text bitmap into two specified colors in hardware.

7. **Cursor.** The Windows cursor can be displayed either through BitBlt commands, or with a hardware cursor.

Figure 1. Windows does its drawings with a small set of graphics primitives, which are described here.
Chapter 3. Hardware Acceleration

Graphics performance is the result of the architectural decisions made when designing the controller, and the quality of the implementation. Quality is not something that can be judged from anything but careful testing, but you can tell whether a controller architecture is capable of high performance by noting a few key features, all of which are invariably noted in data sheets. To a large extent, you can tell fast controllers from slow ones by these few crucial features. Understanding these features also provides a “sanity check” to help separate true performance from industry hype.

This chapter describes the key features of:
1. frame-buffer architecture
2. graphics acceleration
3. microcoded vs. hard-wired controllers
4. workstation-derived vs. PC-derived designs

3.1. Raw Frame-Buffer Bandwidth

The single most important factor in graphics controller performance is frame-buffer bandwidth. This is a measure of how fast data can be moved into or out of the graphics controller’s frame buffer (that is, the memory it uses to store the displayed image). The speed of the frame buffer is the limiting factor for:

1. high drawing speed
2. high display resolution
3. high color depth
4. high monitor refresh rate

A fast frame buffer can support all of these at the same time, while a slower frame buffer can only support trade-offs between them. It is not unusual to have a frame-buffer controller that forces the following trade-offs:

- high drawing speeds, but only at low resolutions
- true-color display, but only at low drawing speeds
- high resolution, but only at low drawing speeds and low monitor refresh rates

Such trade-offs are of questionable value in a demanding environment such as Windows.
3.1. Raw Frame-Buffer Bandwidth, continued

3.1.1. TYPE OF FRAME BUFFER

Frame-buffer bandwidth is traditionally measured in megabytes per second (MB/s). Four factors determine the bandwidth of a frame buffer:

1. Memory technology. The frame buffer can be made from standard DRAMs or the higher-performance VRAMs (video RAMs). VRAM devices have a data-out port in addition to the usual read/write port, giving them twice the total bandwidth of DRAMs.

2. Bus width. Frame buffers have data bus widths of 8, 16, 24, 32, or 64 bits. Wider buses allow higher bandwidth. All high-performance controllers have frame buffer bus widths of 32 or 64 bits. In addition, the buses between the host and the controller, and between the controller and the RAMDAC, all have to be at least 32 bits wide, or there will be a serious loss of speed, resolution, or both.

3. Interleaving. Interleaving is a technique that increases memory bandwidth by using multiple banks of memory, without increasing bus width. A properly designed 32-bit interleaved frame buffer can be as fast as a 64-bit non-interleaved frame buffer.

4. Controller speed. It is difficult to design a memory controller that can keep up with high-speed memory subsystems. Without a low-latency, high-efficiency controller chip, much of the underlying frame buffer bandwidth will be wasted.

High-speed graphics controllers are very difficult to implement properly, and can thus spell the difference between success and failure, even when the underlying architecture is sound. Unfortunately, controller quality can be determined only by testing, where architectural quality can often be determined by inspection.

Figure 2 gives a rough idea of the relative performance of the first three of these elements.

![Figure 2. Simplified depiction of the relative performance of VRAM and DRAM in 32-bit and 64-bit bus widths](image)

3.1.2. SPEED PREDICTIONS

A truly high-speed controller will have all of the following:

1. a VRAM frame buffer,
2. a 32-bit or 64-bit data path on all of these:
   a. frame-buffer interface
   b. host-bus interface
   c. RAMDAC interface
3. interleaved memory
4. a high-speed internal architecture

Performance should correlate very strongly to the degree to which these conditions are met.
3.2. Acceleration of Windows Primitives

Drawing bit-mapped graphics is a very specialized task, one which requires specialized hardware if it is to be done efficiently. An accelerated graphics controller contains at least one drawing engine that performs graphics operations more efficiently than a general-purpose CPU could. A full-fledged accelerator will have several tightly coupled drawing engines: a BitBit engine, a polygon engine, a clipping engine, and so on.

Some of these functions are relatively easy. Bit-block transfer (BitBit), for instance, is the first graphics engine anyone implements, because it’s simple and because it gives the most acceleration of any single drawing engine. Thus, all accelerated controllers have a BitBit engine.

Additional functionality becomes more and more difficult. Many drawing tasks are algorithmically complex, making them difficult to express in circuitry (wide lines, for example). Others are simple, but require sizable areas of silicon to implement (an on-chip pattern RAM falls into this category). A few primitives are used very rarely, so it seems unlikely that anyone would ever accelerate them fully (pie wedges are a good example).

3.2.1. SPEED PREDICTIONS

All other things being equal, the more things that are accelerated directly by dedicated hardware, the better — but the efficiency of different vendors’ drawing engines varies enormously, so predictions can be tricky.

3.3. Microcoded vs. Hard-Wired Controllers

There are two basic approaches to designing accelerated graphics controllers: microcoded controllers and hard-wired controllers. The relative merits of the two approaches are similar to the “CISC vs. RISC” processor technologies.

Hard-wired controllers have no internal microprocessor. Instead, they devote all their silicon to low-level drawing engines. This gives hard-wired controllers a small set of simple graphics commands that they perform very quickly. Such commands include, “draw a line from this point to that point,” or “set the clipping window to these coordinates.” They have no ability to chain operations into larger ones; for example, they can’t be told to draw a blue rectangle with a green border. Instead, the host CPU issues two commands: one for the rectangle, and one for the border.

The fastest controllers on the market today are hard-wired controllers.

Microcoded controllers have, in addition to the drawing engines, an on-board processor (the microcontroller) that can create complex graphics by running its own internal programs (microcode). By having a dedicated processor that is tightly coupled to the drawing engines, microcoded controllers can draw certain difficult shapes, such as circles and splines, very quickly — even though the graphics controller has no special curve-drawing hardware per se. The main CPU can use the same algorithm to draw the same curve, but the latencies in CPU-to-controller communication give the advantage to the microcontroller.

The downside of microcontrollers is that, to be any help, they have to be almost as fast as the host CPU — and host CPUs have gotten very fast indeed. Secondly, the space taken up by the microcontroller is space that can’t be used for hardware drawing engines, so microcoded controllers tend to be slower at basic drawing operations than hard-wired controllers, and faster only in difficult operations such as curves. Finally, microcontrollers are successful only when their strengths are used fully by the operating environment — but Windows leans heavily on elementary graphics functions, not complex ones, as shown in figure 3.

3.3.1. SPEED PREDICTIONS

Hard-wired controllers will win because Windows is a hostile environment for microcoded controllers, which get no opportunity to display their strengths.
3.4. PC-Derived vs. Workstation-Derived Controllers

The issues and requirements of Windows graphics today are very similar to those of the workstation graphics of two to three years ago. The workstation market embraced graphical user interfaces before the PC market did, and the need for high performance was recognized (and addressed) sooner.

The advent of Windows, local-bus PCs, and the 486 have finally made real the old promise that PCs will eventually be indistinguishable from workstations. Workstation technology can solve the same problems for PCs that it did for workstations. If chosen carefully, workstation technology can do so at a price that is acceptable in the PC market.

PC-based graphics controllers evolved along a different path from that followed by workstations, based around incremental evolution from previous standards, with low cost and total hardware compatibility being crucial goals. Performance was secondary, and GUIs were not an issue. Companies that made such products have had to change their strategies very rapidly to meet the new requirements of Windows in local-bus PCs.

The workstation-derived controllers have a strong performance edge, as shown in figure 5. There is a wide performance gap between the two groups: the fastest PC-derived controller is slower then even the slowest workstation-derived controller.

Both groups are working hard at converging: vendors with PC-derived parts are increasing the speed and the feature set of their parts, while the workstation-derived controller vendors are decreasing costs and adding more PC- and Windows-specific features. Soon, it will no longer be possible to guess strengths and weaknesses given the origin of a product line. But at the moment these clues are still valuable. See figure 4 for a comparison of the two technologies.

<table>
<thead>
<tr>
<th>Workstation-Derived Controller</th>
<th>PC-Derived Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher performance</td>
<td>Lower performance</td>
</tr>
<tr>
<td>Higher cost</td>
<td>Lower cost</td>
</tr>
<tr>
<td>More external “glue” required</td>
<td>Less external glue required</td>
</tr>
<tr>
<td>Higher frame-buffer performance</td>
<td>Lower frame-buffer performance</td>
</tr>
<tr>
<td>More high-resolution modes</td>
<td>Fewer high-resolution modes</td>
</tr>
<tr>
<td>More drawing functions</td>
<td>Fewer drawing functions</td>
</tr>
<tr>
<td>Uses 32-bit RAMDAC</td>
<td>Uses 8-bit RAMDAC</td>
</tr>
<tr>
<td>Uses memory interleaving</td>
<td>Does not interleave memory</td>
</tr>
</tbody>
</table>

Figure 4. Comparison of workstation-derived and PC-derived controllers

3.4.1. SPEED PREDICTIONS

Workstation controllers will retain their speed edge for some time.

Figure 5. Controller performance vs. bus and controller type
4.1. Types of Benchmarks

We’ve identified four types of benchmarks, each with its own strengths and weaknesses:
1. applications tests
2. pseudo-applications, or “GUI movies”
3. GDI function-call tests
4. low-level hardware tests

4.1.1. APPLICATIONS TESTS
Applications tests are done by taking a suite of relevant applications and running them in such a way as to measure how long it takes them to do something. In Windows, this is often done via the Macro Recorder application.

ADVANTAGES
This kind of test can be tailored to your specific requirements very easily. It is nearly immune to benchmark cheating, especially if you don’t give out your exact test suite to vendors.

DISADVANTAGES
Custom benchmarks are time-consuming to set up and run. Application test results measure the performance of the entire system (CPU, graphics system, memory speed, disk speed, level 2 cache size, local bus controller efficiency, etc.), without doing anything to indicate which subsystems might need improvement.

4.1.2. GUI MOVIES
A “GUI movie” is a special Windows application program that looks like it is executing real applications, but isn’t. Instead, it is running a “canned” sequence of graphics calls as fast as the graphics driver and controller can put them on the screen.

WINTACH, put out by Texas Instruments, is an example of a GUI movie.

ADVANTAGES
GUI movies focus on graphics: since there is no underlying application, there is little CPU overhead. Effective GUI movies are interesting to watch and step through a variety of workaday Windows tasks. They run to completion in only a few minutes, which is helpful. They generally (but not always) do enough different things that it is difficult for a driver writer to cheat on them — that’s up to the GUI movie’s author.

DISADVANTAGES
Most GUI movies are written by chip or board vendors, and they have to be viewed with a certain amount of suspicion. Also, there’s no guarantee that the application-like output of a GUI movie is actually drawn the same way applications draw things. Since GUI movies look so good, they can be doubly misleading.

4.1.3. GDI FUNCTION-CALL TESTS
GDI function call tests are relatively simple benchmarks that make a series of calls to specific GDI functions, and report the resulting speed, often on a function-by-function basis. Like GUI movies, they test the speed of the graphics device driver and the actual device at the same time. Thus, a slow driver may make a fast chip look slower than it really is, or a clever driver may make a slow chip look faster than it really is. Usually, this doesn’t matter to the end user, who is in no position to fix either the hardware or the software.

The PC Labs Windows Benchmark (WINBENCH) from Ziff-Davis is an example of a GDI function-call test.

ADVANTAGES
If the test gives an item-by-item breakdown of Windows graphics speed, it’s easy to do detailed comparisons of different graphics controllers. Overall figures of merit, such as WINBENCH’s WINMARK number, need to be treated with some caution, however, especially if the weighting factors are not given. These tests typically run to completion in just a few minutes.

DISADVANTAGES
The simplicity of these tests is also their downfall. It’s tests like these that attract most of the benchmark cheaters. Since these tests do only a few well-defined things, it is possible to write a driver that is “benchmark-aware” — that is, one that does things solely to make the benchmarks run faster, even if such optimizations have no benefit (or even reduce performance) in real applications.
4.1 Types of Benchmarks, continued

4.1.4. LOW-LEVEL HARDWARE TESTS

Low-level hardware tests don’t run under Windows at all. Instead, they are programs that manipulate the graphics hardware directly. Such programs are hardware-specific, and thus there are no standard programs that run on all graphics controllers.

ADVANTAGES
Such programs test the actual raw hardware speed of the device, rather than the combination of the device and Windows.

DISADVANTAGES
As they are controller-specific, low-level hardware tests are too specialized for general use.

4.2. Standard Benchmarks

Standard benchmarks come from four sources:

1. You
2. Other users
3. User advocates, such as testing labs and members of the press
4. Vendors

Most of us would agree that the best benchmarks are the ones we develop ourselves, with our own needs in mind. If we can’t do this, or don’t have time, we can obtain benchmarks from other users or from user advocates such as testing labs, which presumably have a strong incentive to maintain their reputation for impartiality.

The perils of trusting a vendor’s own benchmark are obvious: the deck is almost certainly stacked in his favor. The perils of other people’s benchmarks are less obvious, but just as real. Is the benchmark competently designed? Does it test the same things you want to test? Do you understand what it’s doing? Did the author have his own axe to grind?

Benchmarks from vendors are the least desirable, since they always seem to be skewed towards the vendor’s product, sometimes in subtle ways that are difficult to compensate for.

In all cases, you should follow the same rule: Study any benchmark before basing a decision on its results.

Figure 6 lists some standard Windows benchmarks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Type</th>
<th>Description</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINBENCH</td>
<td>PC Labs (User advocate)</td>
<td>GDI Function Calls</td>
<td>Simple test that measures display performance by testing individual GDI function calls. Gives performance by call, and also gives an overall “figure of merit,” the “WINMARK” score.</td>
<td>Very widely used. Scores for individual functions make it possible for you to use with your own weighting functions. Updated frequently to discourage cheating.</td>
<td>Simple design encourages cheats. The longer it’s been since a new WINBENCH release, the more cheats are put into drivers.</td>
</tr>
<tr>
<td>WINTACH</td>
<td>Texas Instruments (Vendor)</td>
<td>GUI Movie</td>
<td>A “GUI movie” showing displays from what appear to be real applications, but aren’t.</td>
<td>Shows a wider variety of text and graphics tasks than WINBENCH. Fun to watch.</td>
<td>It looks compelling, but what’s really going on inside?</td>
</tr>
<tr>
<td>SPEEDY</td>
<td>Hercules/IIT (Vendor)</td>
<td>GUI Movie</td>
<td>Another GUI movie, this time from Hercules and IIT.</td>
<td>As WINTACH, but less interesting.</td>
<td>As WINTACH, but more blatantly biased. Replicates items from WINBENCH 3.1 so the same cheats can work with both.</td>
</tr>
<tr>
<td>Torque</td>
<td>Gibson Research (User advocate)</td>
<td>GDI Function Calls</td>
<td>Bus bandwidth test</td>
<td>Simplicity.</td>
<td>Not comprehensive.</td>
</tr>
</tbody>
</table>

Figure 6. Some standard benchmarks
4.3. Testing Methods

We recommend the following four-step approach to speed analysis:

1. Predict speed based on graphics system architecture.
2. Predict speed based on standard benchmarks.
3. Predict speed based on application testing.
4. Reconcile the results of the three approaches.

The three different types of speed analysis will make it unlikely that large errors or misrepresentations will go undetected, and they give you a good body of data to base decisions on.

4.3.1. ARCHITECTURE AND SPEED

Architecture is the source of performance; it determines the arena in which a graphics controller can play. As we discussed in chapter 3, there are specific features that separate high-performance controllers from the rest. You can make good ballpark performance predictions just by examining the features set of a controller. It ought to have performance similar to other controllers with similar architectures.

4.3.2. STANDARD BENCHMARKS AND SPEED

Standard benchmarks such as WINBENCH are also good indicators of overall speed, so long as you protect yourself against benchmark cheating (benchmark cheating is covered in detail in section 4.4). Since most benchmark cheating is directed specifically at WINBENCH, this benchmark measures speed when used on honest vendors’ products, and it measures honesty on other people’s products. By adding other benchmarks and applications tests to your test suite, you can readily identify honest and dishonest vendors.

These benchmarks are relatively quick-running, and give detailed results as to how fast individual Windows graphics functions run.

Be careful to run all of your tests on identically configured machines. Since it’s difficult to dedicate a machine solely to benchmarking, we recommend that you re-test older controllers in the same machine when comparing them to a new controller. Running tests in two different machines will give identical results only if the CPU type, CPU speed, level 2 cache size, motherboard controller chip set, and DRAM size, speed, and organization are all the same.

THE WINMARK SCORE

One number that WINBENCH gives you is the WINMARK score, which is a round-up figure using a complicated non-linear weighting function. PC Labs uses this number as a figure of merit for graphics controllers.

Whether you should use this number as a figure of merit is something you should investigate. PC Labs has the advantage of understanding the values of the weighting factors and the reasons behind them. You should have a similar understanding before you accept any single figure-of-merit number as your most important test result.

4.3.3. APPLICATIONS TESTS AND SPEED

Probably the most demanding part of the testing is to perform your own formal, in-house applications testing. To be meaningful, this testing has to involve important applications used in realistic ways. Ideally, the testing will be relatively insensitive to non-graphics issues; it should do very little disk I/O, and shouldn’t involve non-repeatable delays such as waiting for the user to press a key.

These tests are often done with the Microsoft Windows Macro Recorder, which plays back mouse movements and key strokes approximately as fast as the system will accept them.

In addition to applications tests, GUI movie tests such as WINTACH are useful to double-check the results. While not as unbiased as WINBENCH or as tailored to your own needs as applications testing, WINTACH is easy to run and exercises a wide variety of graphics commands.

4.3.4. RECONCILING THE RESULTS

Ideally, the results from these three steps should agree. If not, additional tests should be run, the test setup inspected, and vendors queried until the phenomenon is understood. Some common test anomalies with WINBENCH are listed in figure 7.
### 4.3. Testing Methods, continued

<table>
<thead>
<tr>
<th>Anomaly</th>
<th>Possible Explanations</th>
<th>Additional Tests</th>
</tr>
</thead>
</table>
| WINBENCH results seem high compared to applications tests. They are high for the controller’s architecture, too. | 1. Benchmark cheating.  
2. Applications tests include "problem applications." See section 4.3.6.                                                                  | Test an older driver, one that predates the latest WINBENCH. See section 4.4.  
Contact vendor and ask about future driver plans. |
| WINBENCH results substantially higher than in previous driver.          | 1. Benchmark cheating.  
2. Algorithmic improvement in driver.                                                                                                               | (A 10% increase in WINBENCH performance with a new driver represents a significant achievement. Be suspicious of major performance boosts in a single driver revision.)  
Run application tests and more general tests like WINTACH to see if the improvement is general. Also, make sure nothing has changed in the test system. |
| WINBENCH and applications test results both lower than expected.        | Something slow in test system.                                                                                                                           | Very fast controllers spend significant amounts of time waiting for the CPU, and are thus sensitive to anything that degrades system performance. In particular, make sure the secondary cache is turned on. |
| WINBENCH results seem low compared to WINTACH or SPEEDY results.        | Microcoded controllers perform better in biased WINTACH and SPEEDY tests than in more general tests.                                                      | Recognize that WINTACH and SPEEDY are dangerous tests when comparing microcoded controllers to non-microcoded controllers (but are useful with two controllers of the same type.) SPEEDY exploits at least one common WINBENCH benchmark cheat. |
| WINBENCH results decline with increased screen resolution (WINBENCH results are supposed to be resolution-independent). | DRAM-based controller. Such controllers need most of their bandwidth for screen refresh at higher resolutions (VRAM bandwidth is constant). | Determine memory type.                                                                          |
| WINBENCH results vary with the version of WINBENCH (WINBENCH numbers are supposed to be consistent from release to release). | 1. Benchmark cheating.  
2. Increasing CPU overhead of WINBENCH with each release.  
3. "Noise."                                                                                         | See section 4.4. Note that WINBENCH, for all that it gives results to seven or eight figures, only seems accurate to within about 10% from release to release (and the weighting factors sometimes change from release to release). Results from run to run of the same version of WINBENCH on the same system also vary by about 2%. |

Figure 7. WINBENCH troubleshooting guide
4.3. Testing Methods, continued

4.3.5. HEADROOM

System speed may have a strong effect on graphics speed, especially with high-performance controllers. If you put a controller with high headroom into a faster system, the graphics performance will increase. If you put a controller with low headroom into a faster system, the graphics performance will remain the same — robbing you of much of the benefit of a faster CPU.

This happens because the fastest graphics controllers are so fast that they spend a lot of their time idle, waiting for the CPU to issue another graphics command — even in extremely graphics-intensive tasks. Such controllers really shine in a DX/2 or Pentium system.

With slower controllers, the roles are reversed, and the CPU spends a great deal of time waiting for the controller to finish. A fast CPU can do nothing to overcome a slow controller.

Because CPU power can have a strong effect on results, it’s a good idea to test a controller on both a “typical” machine and a “fully loaded” machine.

4.3.6. PROBLEM APPLICATIONS

One area that can cause anomalously low performance is what we call “problem applications.” These are graphics-intensive applications that make heavy use of features that the Windows 3.1 GDI does not support very well.

When Windows supports a drawing function directly, that function is always handled in the same way. When Windows does not support a function, applications writers write their own individual versions of that function. This results in controller vendors having to write special optimization code for each popular package that synthesizes non-supported graphics capabilities.

The two most common problem areas are complex curves and smooth shading.

Curves. The problem with curves comes from Windows 3.1’s weak curve set (fixed in Windows NT by the addition of Bezier curves). Writers of applications (mostly drawing programs) that make extensive use of complex curves are left to their own devices as to how to draw them. The problem can be attacked in a variety of ways, some of which are easier to accelerate than others.

Shading. The shading problem comes up when gradient fills or complex patterns are used. Again, this is outside Windows’s normal bounds, and the applications writers approach the problem in different ways.

The upshot is that it is fairly common to run into a controller that runs more quickly on Micrografx Designer than on Corel Draw, or vice versa. This is more a driver tuning issue than a hardware performance issue; alert vendors will work out these kinks over time.

Figure 8. Example of headroom: increase in performance with an increase in CPU power for three controllers.
4.4. Benchmark Cheating

Benchmark cheating is the process of adapting drivers so a product scores well on benchmarks, without giving any benefit to real applications. Benchmark cheating is common in the Windows graphics controller industry. Its main focus is on PC Labs’ WINBENCH test suite.

Benchmark cheating is harmful because it is used to fool people into buying performance that does not exist. On the other hand, cheaters, once exposed, often reveal more about themselves and their level of technical expertise (or lack thereof) than they would like.

4.4.1. THE “SAWTOOTH” GRAPH

The problem with benchmark cheating (from the cheater’s point of view) is that it is easily detected. One method of detection is to simply plot benchmark performance over time, and to note what happens when a new, cheat-retardant benchmark is released.

Figure 9 shows how this process unfolds with WINBENCH. Three vendors are attempting to sell controllers in the marketplace. Vendor A has a high-performance controller. Vendors B and C have similar medium-performance controllers. Vendor B has embarked on an all-out benchmark cheating effort so he can claim that his part is as fast as vendor A’s.

Each new driver from every vendor shows higher performance. All vendors claim that these gains stem from algorithmic improvements that will benefit applications as well as benchmarks.

Then PC Labs releases a WINBENCH 3.1, which has been recoded to render all the common benchmark cheats useless. What happens?

1. The honest vendors suffer a small performance drop (less than 10%), due to some combination of increased CPU-intensiveness of the new version, changes in the weighting factors, and changes that coincidentally invoke the use of less well-optimized sections of the driver than previously.

2. The cheaters are revealed as frauds.

3. Die-hard cheaters immediately start coming up with cheats that work with the new WINBENCH.

4. PC Labs begins work on a new WINBENCH that will nullify the new cheats.

The resulting plot, shown in figure 9, is what we call the “sawtooth graph.” It shows the ebb and flow of the “benchmark wars” being waged between the benchmark creators and the benchmark cheaters.

The fact that such a simple record of performance over time can reveal benchmark cheating, without additional tests of any kind, makes the enormous amount of effort put into benchmark cheating look pretty pathetic.

(Note that vendor A is the only one of the three to show a steady increase in actual performance over time. The others are working on cheating as their sole source of “improved” performance.)

Figure 9. WINBENCH performance vs. time. Vendors B and C really have identical performance, but vendor B is doing a great deal of cheating — only to be caught each time a new version of WINBENCH comes out.
4.4. Benchmark Cheating, continued

4.4.2. DETECTING BENCHMARK CHEATING

The simplest way of detecting benchmark cheating is to run WINBENCH and applications tests, then compare the results. If vendor A is faster on applications tests, and vendor B is faster on WINBENCH, you should be suspicious.

The next step should be to run additional benchmarks (preferably non-vendor benchmarks). A vendor who only shines on WINBENCH is probably up to no good.

Finally, try to get the driver the suspect vendor was using just before the latest edition of WINBENCH. Each version of WINBENCH is more cheat-retardant than the last. An older driver will have cheats that worked on the older WINBENCH — but they won’t work on the newer WINBENCH. If the performance is much higher on the new driver than the old, you’ve probably caught a cheater.

<table>
<thead>
<tr>
<th>WINBENCH</th>
<th>Driver</th>
<th>Result if Cheating</th>
<th>Result if Not Cheating</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>OLD</td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>OLD</td>
<td>NEW</td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>NEW</td>
<td>OLD</td>
<td>LOW (&lt;10% drop)</td>
<td>HIGH</td>
</tr>
<tr>
<td>NEW</td>
<td>NEW</td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Figure 10. Cheaters are revealed when their old drivers are run on a new WINBENCH.

4.4.3. FAMOUS BENCHMARK CHEATS

THE “BART SIMPSON” CHEAT

Before WINBENCH 3.11, text speed was measured by writing the same line of text over and over. The more technically adept benchmark cheaters recognized that they could create a single-entry text string cache. That is, when they received a string to convert into a device bitmap and display on the screen, they would first check to see if this string happened to be identical to the last one they displayed. If so, they would retrieve the already-rendered bitmap of the text, and copy it into place with a single BitBlt operation. This improves the display speed by about a factor of two.

Unfortunately, it only doubles the display speed if you’re writing the same line of text over and over. We couldn’t think of any use for this ourselves, but Bart Simpson is often put in the position of staying after school until he has written the same line a hundred times, so maybe he could benefit from it.

One vendor, apparently daunted by the task of creating the simple cache the Bart Simpson cheat requires, compared every incoming string to the hard-wired string, “The quick brown fox jumped over the lazy dog, then sat on a tack.”

Not even Bart Simpson can benefit from that.

THE BITBLT CHEAT

While the Bart Simpson cheat slows down real applications (checking every string for something that doesn’t happen in real life slows the system down), at least it doesn’t put the wrong pixels on the screen. The BitBlt cheat does.

One of WINBENCH’s BitBlt tests builds large blue and red rectangles from tiny (one-pixel) rectangles. Since it is much faster to draw a single big rectangle than large numbers of tiny ones, it’s possible for a driver to cheat as follows:

1. Monitor graphics function calls as they go through the driver.
2. Notice that a sequence of calls matches one of the WINBENCH tests.
3. Assume that WINBENCH is being run, and perform a short cut.

This method has two basic drawbacks:

1. Software overhead to benefit only WINBENCH penalizes everything else.
2. You can guess wrong, and draw garbage to the screen if you aren’t really in WINBENCH. (The more dishonest drivers acquire reputations for unreliability due to bugs like this.)

4.5. Conclusions

In spite of the continuing presence of hype and misrepresentation in the Windows graphics controller industry, discovering the true performance of a graphics subsystem is not very difficult. With a grasp of controller architecture, a sense of history, and a small suite of tests, you can determine the facts quite readily.