Benchmark Battle

Click & Retrieve Source CODE!

BY WARD HITT

VB3 beats VB4 at graphics processing, but in other tests VB4 pulls up even or surges ahead in 16- or 32-bit mode.

The transition from VB3 and Windows 3.x to VB4 and Windows 95 or NT makes for a challenging time. In fact, this transition can be downright stressful. Not only do we need to master new technology, but we also need to straddle two sets of tools and platforms. Many users are still running 486-based computers configured with 8 MB of RAM, while others have upgraded to loaded Pentiums—this mix occurs often within the same organization. Some corporations made the leap to Win95, while others are waiting for a more mature release.

Users are working on a range of machines and operating systems. How do you address this melange? Do you develop for all platforms? Do you stick to just 16-bit applications running on both operating systems, or rewrite for 32-bit? Complicating matters is the fact that you probably want to take advantage of new features in VB4, such as classes, improvements to the editor, enhancements to the DAO model, and many others.

But the key question is this: if you move to Windows 95 or Windows NT running VB4 32-bit, are your apps *faster*?

As I discovered while conducting benchmark tests for this article, the answer depends on the type of application you create. For graphics processing using VB methods, VB4 32-bit is the slowest.

Then again, for other types of operations it's the fastest. And sometimes VB4 is about the same as VB3. VB4/16 also wins its share of tests.

To quantify your choices, I've conducted a series of benchmarks of VB3 and VB4/16 and VB4/32, on Windows 3.1, Windows 3.11, Windows 95, and Windows NT. The results may surprise you. Let me offer a caveat: this suite of simple tests provides a set of data points, not universal guidelines for all developers under all situations. These results are no substitute for solid coding techniques and optimization tailored for your users' specific systems. But they do shed insight on performance issues.

Before I provide an analysis of my results, let me explain how this project evolved. Last fall I completed more than 800 benchmark tests of VB4 in 16- and 32-bit modes on Windows 95.

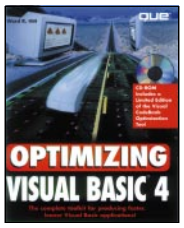
I reported some results in a sidebar to an article on optimization (see "Weighing 32-Bit Performance Trade-Offs," *VBPJ*, January 1996). One result showed that VB4/16 is faster than VB4/32 for graphics processing. Intrigued by the results, *VBPJ* editors decided to expand the tests to measure performance against VB3, VB4, and multiple operating systems. To give the project breadth and depth, a team was assembled to brainstorm about benchmarks in general, design the test suites, conduct the tests, consult, and double-check results.

Based on discussions with *VBPJ* editors, along with results I observed from the original 800 VB4 tests, we agreed that representative tests would use:

- A 486/66 computer with 8 MB of RAM, and a Pentium/100 computer with 16 MB of RAM.
- VB3, VB4/16, and VB4/32.
- Windows 3.1, Windows 3.11, Windows 95, and Windows NT.
- Functionality test of graphics, form loading, text display, and number crunching.

Because the majority of *VBPJ* readers create custom databases, author and CompuServe section leader Steve Jackson agreed to perform a comprehensive series of data-access techniques, including Jet DAO, the ODBC API, and RDO, using all versions of VB and multiple operating sys-

Ward R. Hitt is a consultant and developer located in Washington D.C. He is the author of the book Optimizing Visual Basic *(Que), a VB programming tool called CodeBank published by Visual Components, and other apps. He's writing a book about VB and the Windows API. Reach Ward on CompuServe at 73361,106.*



Action/Method	Windows 3.1 VB3	Windows 3.1 VB4/16
VB Graphics	19.214	28.53
API Graphics	1.232	1.009
Form Load	1.15	1.023
Text Display	0.494	0.494
Floating-Point Division	1.013	0.936
Integer Division	1.667	1.435
Dynamic-String Concatenation	4.284	3.256
Fixed-String Concatenation	8.624	11.754

TABLE 1 ***Battle of the 16-Bitters.*** *VB3 starts out with a bang by beating VB4/16 in graphics-methods benchmarks. However, switching to API graphics calls puts VB4/16 in the lead, and it never relinquishes it. Although text-display speeds are identical, VB4/16 pulls away for form loading and number crunching.*

Action/Method	Windows 95 VB3	Windows 95 VB4/16	Windows 95 VB4/32
VB Graphics	23.58	25.806	65.355
API Graphics	1.304	0.989	1.509
Form Load	1.580	2.691	3.185
Text Display	0.619	0.659	1.044
Floating-Point Division	0.934	0.790	0.476
Integer Division	1.535	1.017	0.715
Dynamic-String Concatenation	3.819	2.606	1.374
Fixed-String Concatenation	8.913	10.910	16.440

TABLE 2 ***Battling With Win95 On a 486.*** *Results aren't as clear-cut running Windows 95 on the 486 with 8 MB of RAM. VB3 performs well for graphics work again, and leads the pack for form loading and text display. Number crunching favors VB4/32 all the way, but VB3 wins again for fixed-string concatenation.*

tems (see the accompanying article, "Clocking Data Access," in this issue). Steve also duplicated my tests to double-check my results and note hardware or software problems that might occur when testing on other machines (see the accompanying sidebar, "Double-Checking the Benchmarks").

WHY THESE ALGORITHMS?

I have a simple reason for choosing form loading, text display, and number crunching as functions to test. Most people use computers to display graphics and text, to store and retrieve data, and for number crunching. My consulting editors agreed that graphics applications are widespread, and a dramatic indicator of design tradeoffs in Windows 95 and VB4.

As for form loading, almost all VB apps must load forms, and forms load slowly in VB3. I wanted to see if speed improved in VB4. We agreed to test displaying text in standard VB controls because, again, almost all VB apps display text.

Lastly, we decided to test number crunching to see if VB4/32 indeed performs calculations faster than the 16-bit versions, as expected.

I used the same suite of eight separate routines for all tests. All tables show results for all routines, except where VB4/32 can't be run, such as with Windows 3.1.

First, I tested graphics speed of VB Point and PSet graphics methods using a routine I call Shade3D, which takes a sample text string and applies 3-D shading to it based on pixel-by-pixel analysis of the edge points of the text graphic. I used this routine for the January 1996 cover story I wrote on optimizing VB, "Tune For Blazing Speed."

The Shade3D font size was increased to 24 points to strain the system a little. The second test uses the GetPixel and SetPixel Graphical Device Interface (GDI) API calls to replace VB's graphics methods (and employs other optimizations).

The third test measures the time required to load and display a form with 62 controls, using the simplest technique: the Show method. Fourth, I measured text-display speed by assigning values to 60 labels and text boxes on the form.

The last four tests measure internal CPU intensive operations: floating-point division, integer division, dynamic-string concatenation, and fixed-string concatenation. These last four tests were run 100,000 times in order to obtain more significant timings.

USE API CALLS FOR LIGHTNING-FAST VB4 GRAPHICS PROCESSING.

WHO WINS?

Before analyzing tests according to operating system and hardware (beginning with Windows 3.1 on the 486 and ending with the loaded Pentium running NT), here's an overview of results.

I found two weaknesses in the design of VB4: graphics processing using VB methods, and fixed-string concatenation. Although VB3, VB4/16, and VB4/32 swap the first-place position in form loading and text display, VB3 wins all the graphics-processing tests for VB methods, but not API graphics. VB3 wins all the fixed-string concatenation tests. Meanwhile, VB4/32 wins all the number crunching.

I'll elaborate on these findings, but for now just keep in mind that VB4 was designed from the ground up to use VBA and OLE, so it's significantly different from VB3, and design changes can affect results.

VB3 wins all the graphics tests when using VB methods. In the tests of the graphics methods under Windows 3.1, VB3 is about 33 percent faster than VB4/16. Under Windows 95, VB3 is about 56 percent faster than VB4/16, but almost three times faster than VB4/32. VB3 wins convincingly for Windows 3.11 and Windows NT as well. VB4 performs poorly for graphics processing because it must thunk to a 16-bit GDI. For graphics-intensive applications using VB methods, VB3 is a superior performer.

However, I recommend avoiding VB graphics methods and using API graphics calls instead. The API calls dramatically slash graphics processing time on the 486, and make significant improvements to processing time on the Pentium.

VB4/16 wins all the tests using graphics API calls except on NT, which VB4/32 wins. So moving to API calls put VB3 out of the running for graphics performance.

In form loading, VB4/16 is about 33 percent faster than VB3 under Windows 3.11, 15 percent faster under NT, and slower on both the 486 and the Pentium under Windows 95.

VB4/32 was slower than VB4/16 to load the test form under Windows 95, but much quicker than either VB3 or VB4/16 under Windows NT. VB3 was the fastest to load under Windows 95.

I have one subjective note regarding perceived speed: the Form2 load timing routine is terminated in the form's Activate event, but there may be a meaningful difference in the way different versions of VB handle form drawing.

Under VB3 and VB4/16, the form appears to fully paint before the MsgBox appears. However, with VB4/32, the text boxes and labels do not draw completely before the beep and MsgBox. This may be an optimization, or it may be due to 32-bit multitasking. But whatever the cause, your users may perceive VB4/32 from loading as not quite as fast as the numbers indicate.

Most VB apps need to repeatedly display text in controls, so I tested using both labels and text boxes. VB4/16 text display was about twice as fast than VB3 under Windows 3.11, about equal under Windows 95, but slower under NT (displaying 16-bit apps in NT is slowed because there is no 16-bit code in NT). VB4/32 was about 25 percent slower than the other versions under Windows 95, but was much faster under NT—again, because VB4/32 shines in the 32-bit environment of Windows NT.

Number-crunching speed increased moving from VB3 to VB4/16 to VB4/32. VB4/16 is faster than VB3 under 3.1 and NT, and equivalent under 95. VB4/32 is significantly faster than VB3 and VB4/16 in all tests except for fixed-string concatenation, which is surprisingly slow in VB4/32.

A Microsoft engineer told me that fixed-string concatenation has more "baggage" to go through than dynamic-string concatenation. He couldn't elaborate, but added that fixed-string concatenation is a target for improvement in the next release of VB.

The VB4 routines required only minor, nonsubstantive editing to enable them to run under VB3. I used code that is generally simple and to the point for these tests, in order to avoid debates about what I'm really testing. The purpose of these benchmarks is to test common, widely used functionality, not to spark a debate about esoteric coding techniques.

The number crunching is a good example of my test routines (see Listing 1). You can download all test code as VBBENCH.ZIP (see "How To Reach Us" in the Letters department for downloading instructions). If you download and test the code, send me an e-mail with your results and the type of system you used (including RAM, CPU, clock speed, and Windows version). I may establish a knowledge base of benchmark tests.

TESTING ON A 486

I'll analyze results starting with the oldest technology in both hardware and software, then move to the loaded Pentium running Windows 95 and NT. I conducted a suite of benchmarks on my 486 with 8 MB of RAM running Windows 3.1 (see Table 1). VB3 wins only the graphics methods contest. VB4/16 quickly catches up, then takes and never relinquishes the lead through text display, form loading, and especially number crunching. If you're going to stick with 16-bit development, VB4/16 could be best if you can live with APIs for graphics work.

Next, I tested Windows 95 on the 486 system (see Table 2). VB4/16 is more than twice as fast as VB4/32 using graphics methods, and about 50 percent faster when using the API. VB3 posts the fastest graphics-methods times: it's about 9 percent faster than VB4/16, but slower than VB4/16 when using API calls.

VB4/32 is slowest for all graphics techniques on the 486 system. VB4/32 is always fastest for number crunching, yet it falls behind consistently for fixed-string concatenation. Form display is somewhat faster under VB3, while VB4 has faster calculation performance compared with VB3.

As I discovered during testing, VB4's graphics performance lags because it thunks down to the 16-bit graphics engine for image processing. Although VB4 on Windows 95 has superior performance in many cases, graphics processing is its Achilles' heel. If you're developing graphics-intensive apps,

CONTINUED ON PAGE 42.

CONTINUED FROM PAGE 38.

you may be better off with VB3 or redoing your graphics logic using API calls rather than VB methods.

Although running VB4-generated apps on 8 MB of RAM is definitely not a good idea, many users still work with 486 systems. Windows 95 requires approximately 5 MB of RAM (bare-bones networking setup running the Explorer) and the VB/VBA run time takes 1 MB after initializing the OLE libraries, which can easily eat 2 MB on a 32-bit system. We're looking at a *very* memory-constrained setup here. But again, this suite of tests establishes the boundary conditions for low-end users.

My tests indicate that VB4/32 mode is more sensitive than VB4/16 to RAM shortages. Although Windows 95 specifications state that only 4 MB of RAM are required (that's the spec on the box: internally, it sets a low-memory flag when running on machines with less than 5 MB), the amount needed for best performance is, of course, much higher.

I used an 8 MB 486 machine because I believe the configuration represents the type of machine the average user will have for at least the next year, and optimization should be performed on typical systems, not state-of-the-art boxes. Of course, I expect a gradual migration to Pentiums. A typical corporate upgrade will be a Pentium 100 with 16 MB of RAM running on a network.

VB4

```
Private Sub Command2_Click()
BeginTime "Load 60 control form", TIME_MSGBOX
Form2.Show
End Sub
Private Sub Command3_Click()
'runs suite of mathematical & string manipulation tests
Dim foo As Long
Dim dA As Double, dB As Double, dC As Double
Dim iA As Integer, iB As Integer, iC As Integer
Dim sA As String, sB As String, sC As String
Dim sfA As String * 10, sfB As String * 10, sfC As _
  String * 20
iA = 12
iB = 72
dA = 12
dB = 72
sA = "ABCDEFGHIJ"
sB = "KLMNOPQRST"
sfA = "ABCDEFGHIJ"
sfB = "KLMNOPQRST"

BeginTime "Floating Point Division", TIME_MSGBOX
For foo = 1 To 100000
  dC = dB / dA
Next foo
EndTime

BeginTime "Integer Division", TIME_MSGBOX
For foo = 1 To 100000
  iC = iB / iA
Next foo
EndTime

BeginTime "Concatenation", TIME_MSGBOX
For foo = 1 To 100000
  sC = sB & sA
Next foo
EndTime

BeginTime "Fixed String Concatenation", TIME_MSGBOX
For foo = 1 To 100000
  sfC = sfB & sfA
Next foo
EndTime
End Sub
```

LISTING 1 ***Keep Tests Simple.*** *The purpose of my tests was to compare commonly used functionality, rather than to spark debate over exotic programming techniques. For example, this code for my number-crunching tests is simple and straightforward.*

PACING PENTIUM PERFORMANCE

Next, I measured VB3, VB4/16, and VB4/32 performance on identical systems running 100 MHz Pentium CPUs, with 16 MB of RAM, local-bus video, and three operating systems: Windows 3.11 for Workgroups, Windows 95, and Windows NT 3.51.

The first thing I noticed on the Pentium is an order-of-magnitude speed improvement for graphics processing for all versions of VB, versus performance on the 486 (see Table 3). Although all code runs faster on the Pentium than the 486, graphics speed improves dramatically.

Again, VB3 logged exceptional performance for VB graphics methods when running Windows 3.11 on the Pentium (see Table 3). Under Windows for Workgroups, VB3 is almost twice as fast as VB4/16 using VB graphics methods (of course, 32-bit VB can't run on Windows 3.11). VB4/16 pulls ahead for graphics API performance, and continues to lead in form loading, text display, and number crunching, except for fixed-string concatenation.

When the operating system is upgraded to Windows 95, VB3 slows down slightly while VB4/16 speeds up slightly, indicating that VB4/16 is probably optimized for Windows 95 (see Table 4). VB4/32 is slower than VB4/16, but only by about 20 percent.

But this improvement is much better than the 100-percent speed differential seen between VB4/32 and VB4/16 on the 486. Again, a trend is apparent: VB3 is fastest for graphics methods, VB4/16 is fastest for API graphics, form loading and text display, and VB4/32 is fastest for number crunching on the Pentium running Windows 95 (confirming that the 32-bit capability of VB4 does indeed speed number-crunching performance).

Under Windows NT, VB3 is still the best performer using VB Point and Pset methods. However, VB4/32 is about 20 percent faster than VB4/16 (see Table 5). Windows NT has a true 32-bit GDI, so there's no thunking for VB4/32.

When executing the optimized API version of the graphics test, we see significantly different results. VB4 is much faster than VB3 when making GetPixel and SetPixel calls. Under Windows 3.11 and Windows NT, VB4/16 is 33 percent faster, while it is almost twice as fast under Win95. VB4/32 is faster than VB4/16 on NT, but slower under Win95. Again, no thunking for VB4/32 on NT.

Running the test suite on Windows NT revealed some interesting design decisions by NT engineers. First, graphics processing is slower for all versions of VB running on NT compared with Windows 95. VB4/32 is more than twice as slow for VB graphics methods on NT than it is on Windows 95. VB4/16 is almost three times slower under NT than under Windows 95, while VB3 is fully five times slower under NT than on Windows 3.11.

	Windows 3.11	
Action/Method	VB3	VB4/16
VB Graphics	2.197	4.170
API Graphics	0.380	0.297
Form Load	0.165	0.110
Text Display	0.110	0.055
Floating-Point Division	0.274	0.164
Integer Division	0.439	0.220
Dynamic-String Concatenation	0.933	0.549
Fixed-String Concatenation	1.922	2.637

TABLE 3 ***Gauging Pentium Performance With Windows 3.11.*** *The pattern holds true for Windows 3.11 on a Pentium: VB3 excels in graphics methods, but VB4/16 takes the lead for API graphics and stays there until it hits fixed-string concatenation.*

	Windows 95		
Action/Method	VB3	VB4/16	VB4/32
VB Graphics	2.540	3.960	4.957
API Graphics	0.412	0.216	0.247
Form Load	0.147	0.153	0.244
Text Display	0.074	0.075	0.104
Floating-Point Division	0.261	0.191	0.134
Integer Division	0.412	0.243	0.205
Dynamic-String Concatenation	1.012	0.710	0.342
Fixed-String Concatenation	2.310	3.070	3.266

TABLE 4 ***The Pentium With Windows 95.*** *VB3 graphics are slow under Windows 95 Windows 3.11, while VB4/16 graphics are fast. 32-bit graphics lag due to thunking. VB3 wins at form loading and text display, but VB4/32 wins at number crunching.*

	Windows NT		
Action/Method	VB3	VB4/16	VB4/32
VB Graphics	7.936	12.160	10.375
API Graphics	2.112	1.664	1.282
Form Load	0.448	0.384	0.231
Text Display	0.192	0.256	0.130
Floating-Point Division	0.192	0.192	0.141
Integer Division	0.256	0.255	0.201
Dynamic-String Concatenation	1.150	0.576	0.381
Fixed-String Concatenation	2.110	2.880	2.844

TABLE 5 ***What About Windows NT?*** *Each suite runs slower on Windows NT than on Windows 95. VB4/32 gives a good showing compared to other flavors of VB. After stalling in graphics, VB4/32 pulls ahead except in fixed-string concatenation.*

Also, VB4/32 beats VB4/16 in graphics processing for the first time, although good old VB3 wins the race. For the first time, VB4/32 leads the pack in every other test except for that pesky fixed-string concatenation. If you're developing non-graphics applications for NT, VB4/32 is clearly faster than VB3 or VB4/16, based on these benchmarks.

As for comparing operating systems, keep in mind it's an apples-to-oranges comparison. I had a long talk with a Microsoft engineer about operating systems. He explained that the products I tested are totally different operating systems with many design differences, especially in their memory-allocation schemes, disk I/O design, network access, and other important areas.

It's difficult, perhaps impossible, to pinpoint why performance varies. I can examine the results in these simple, controlled benchmarks and use them as one set of data points.

The much-discussed plan of many corporations to skip Windows 95 and go straight to Windows NT involves performance trade-offs. Microsoft designed NT to be more reliable and robust than Windows 95, yet it seems that speed is sacrificed in the display area, although number crunching is faster on NT than on Windows 95. Windows 95 has lower memory and processor requirements than NT, and Windows 95 is not slowed by layers of security between applications. Because GDI access is nonreentrant in Windows 95, it must be serialized, and one application crashing the GDI will bring down the entire system.

It's clear that if your users are running non-graphics intensive applications on fast Pentiums with plenty of RAM, speed differences between all three versions of VB are measured in fractions of a second. One could build a case for summarily using VB4 for Pentium applications.

Personally, I strive to make applications as snappy as possible. I can say that nanoseconds usually don't matter (such as the one-nanosecond difference in text display between VB3 and VB4/16 under Windows 95), unless you're in a loop and doing a lot of them. Generally speaking, VB4 versions are about equal to or better than VB3 for Windows 95 or NT.

On the other hand, if your users are operating under Windows 3.1 with 8 MB of RAM, VB3 could still be the tool to use. The best and only way to interpret this data, as always, is in the context of your own situation.

Double-Checking the Benchmarks

I duplicated Ward Hitt's benchmark tests running the same operating systems on comparable hardware. My results closely match Ward's across machines and operating systems, with the exception of graphics processing on the Pentium.

My graphics-method tests for all VB versions on Windows 95 were 60 percent slower. VB3 graphics-method test results on Windows 3.11 were 24 percent slower, and VB4/16 graphics tests were 60 percent slower. However, the ranking of the graphics tests remains about the same: VB3 is approximately twice as fast as VB4/16. My raw results are slower.

Ward's machine has local-bus video. Mine doesn't, although its CPU and memory match Ward's configuration. My computer has a Super VGA card. I believe my video card and driver are slower than Ward's, which accounts for the speed difference on my machine.

I had one problem that demonstrates how tricky it can be to benchmark code across different machines. When I ran the two string tests on my Pentium/100 with 16 MB of RAM running Windows 3.11, I recorded much slower VB4 speeds than Ward recorded.

Ward's dynamic- and fixed-string VB4 Windows 3.11 test results were 0.55 and 2.64 seconds, respectively, but my initial results were 2.4 and 9.0 seconds.

These results were a glaring anomaly among otherwise close comparisons. How could my string results be three to four times slower? Upon investigation, I found that some of my Windows 3.11 DLLs were older than those in my Windows 95 directory: they were probably from a beta version of VB4 I tested months ago.

Beta software is often compiled with extra debugging options that contribute to overhead. Running the WPS program that accompanies VB3 revealed that the VB4 test program was using STORAGE.DLL and a number of OLE DLLs in addition to the VB4 run time.

I copied the newer versions of these DLLs from the VB4 installation CD, reran the tests, and got results of 0.55 and 2.6 seconds for the string tests—an exact match of Ward's results.

Clearly, software as well as hardware components affects test results. If you download Ward's code and duplicate the tests, I'd like to hear from you if your results vary because of differences in your hardware or software.—*Steve Jackson*