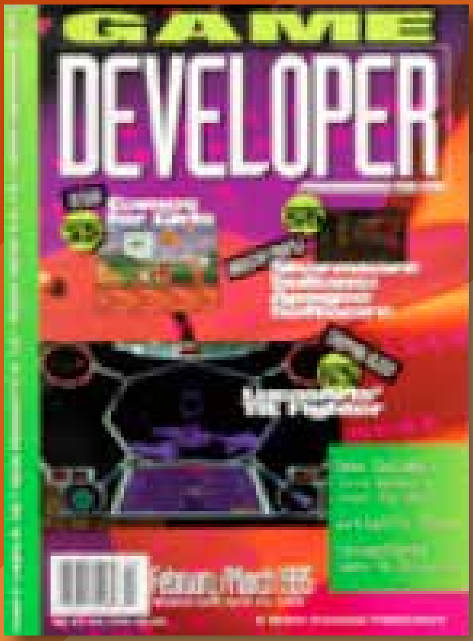
gd

GAME DEVELOPER MAGAZINE

FEBRUARY/MARCH 1995



UBM Tech

Back

The No Go Logo

There may never be a game with a "Windows '95 Compatible" logo, not even from Microsoft. Microsoft, by arrogant fiat, has decided that the seemingly literal phrase, with it's seemingly straightforward purpose, should be held hostage to the whims of some Redmondian marketing genius. Windows '95, the new operating system from Microsoft, will roll out later this year and, largely due to the bundling agreements Microsoft has with clone makers, will quickly gain its greatest marketshare in the home computer market.

To run under Windows '95, your program will have to do one of two things: run as a Windows program, with the input, output, and multitasking methods dictated by the Win32 API, or run in a virtualized DOS box.

It's likely that much of the hard-won knowledge of how to get the most performance from machines running DOS may no longer work on machines running Windows '95. We all know how few games run in the DOS boxes of Windows 3.1, the Windows '95 DOS box will be like that (except different in unknown ways). The only answer seems to be the off-putting boot disk.

Your alternative is to create a true Windows application. There are some advantages, the greatest of which is device independence. Lifting the burden of programming for every video and sound chipset in the known world should free up time for...well...learning the confines of the Windows API.

But let's say that you've PeekMessaged and PostMessaged your way around the event queue; your program is WinGed and WinTooned, and you're the happiest little WinCamper in the whole wide WinWorld. Can you put that logo on your game? Not even close.

First, can you send your saved game over your home's Ethernet backbone (that is, is it mail-enabled)? Second, can you embed an Excel spreadsheet of your inventory in the middle of your character sheet (that is, does it support OLE 2.0)? Do you have a tabbed dialog that walks you through the game (that is, do you have Wizards)? Finally, does it work on a different operating system, with a different base architecture including a different tasking model (that is, Windows NT)?

In other words, to be "compatible" with Windows '95, your game has to be a mail-enabled, en-Wizarded OLE Server application that runs under NT. All criticism to date about this policy has come from shrinkwrap application vendors, who have pointed out that this is burdensome even for office products. Honestly, though, I can understand the argument that "compatible" when applied to an office application may mean a certain set of services above and beyond display services. With entertainment products, the very products most needy of some kind of validation, this argument is without merit.

I suggest two courses of action. First, complain to Bill Gates himself, asking for a reconsideration of the policy or suggesting an alternative "Ready to Run Under Microsoft Windows '95" validation appropriate for digital entertainment products. Mail directed to billg@microsoft.com will not get through without being screened, but it will be read by someone and, perhaps, even by Gates himself. Second, create a utility—a character editor or some such—that has all the necessary components. The functionality or appropriateness isn't important, this is just a silly way to get around the silly restrictions. With such a utility, your game isn't overly burdened, your box gets the logo, and your users, if you have a dynamite game, are oblivious to this tempest in a teakettle. ■

Larry O'Brien, Editor

GAME DEVELOPER

Editor Larry O'Brien
gdmag@mfi.com

Senior Editor Nicole Freeman
76702.706@compuserve.com

Production Editors Barbara Hanscome
73611.633@compuserve.com
Nicole Claro
76702.1141@compuserve.com

Editorial Assistant Diane Anderson
diane_anderson@mfi.com

Contributing Editors Alex Dunne
75010.2665@compuserve.com
Chris Hecker
checker@microsoft.com
David Sieks
dsieks@arnarb.harvard.edu
Wayne Sikes
70733.1562@compuserve.com

Editor-at-Large Alexander Antoniades
sander@mfi.com

Cover Photography Charles Ingram Photography

Publisher Veronica Costanza

Group Director Regina Starr Ridley

Advertising Sales Staff

West/Southwest
Yvonne Labat (415) 905-2353
ylabat@mfi.com

New England/Midwest
Kristin Morgan (212) 626-2498
kmorgan@mfi.com

Marketing Manager Susan McDonald
Art Director/Marketing Christopher H. Clarke
Advertising Production Coordinator Denise Temple
Director of Production Andrew A. Mickus
Vice President/Circulation Jerry M. Okabe
Group Circulation Director Gina Oh
Circulation Manager Kathy Henry
Circulation Assistant Phil Payton
Newsstand Manager Pam Santoro
Reprints Stella Valdez (415) 655-4269

Miller Freeman
A MEMBER OF THE UNITED NEWSPAPERS GROUP

Chairman of the Board Graham J.S. Wilson
President/CEO Marshall W. Freeman
Executive Vice President/COO Thomas L. Kemp
Senior Vice Presidents H. Vern Packer, Donald A. Pazour, Wini D. Ragus
Vice President/CFO Warren (Andy) Ambrose
Vice President/Administration Charles H. Benz
Vice President/Production Andrew A. Mickus
Vice President/Circulation Jerry Okabe
Vice President/Software Development Division Regina Starr Ridley

Back

The Golden Era of Siliwood

Alex Dunne

As the artistic lines separating Hollywood and Silicon Valley increasingly blur, two major entertainment industries—film and computer and video games—find they might not make such strange bedfellows.

Ninety years after it first burst onto the scene, the cinema is undergoing a renaissance. More than just a rebirth, actually, it's really a fusion with computer and video games that's resulting in some cool entertainment: a new breed of interactive, "live-action" games featuring Hollywood movie stars. Such games, like Hell, Under a Killing Moon, and Wing Commander III, are coming out with more frequency, and they're boosting the acting careers of some people in Tinseltown.

Before we look at live-action games, let's first take a quick look into the past. The evolution of the American film industry might be a good model to explore in attempting to extrapolate the future of these games and their impact.

First invented by Thomas Edison in the late 19th century, motion pictures were a new form of entertainment that appealed to the masses, relied on state-of-the-art technology, and made stars out of performers like Charlie Chaplin, Rudolph Valentino, and Mary Pickford. As film technology evolved, so grew the impact of the industry on the nation. The first movies shown in nickelodeons were 10 minutes long, black-and-white, and silent. However, as the medium evolved, sound and color were added, and movies got beefed out to two hours in length. (I won't even talk about today's cutting-edge advancements like THX surround-sound and IMAX screens.)

In terms of their economic impact, look where American movies are today. Hollywood is the undisputed entertainment mecca of the world. Take the fact that the French refused to drop economic barriers to the American film industry during the Uruguay round of the General Agreement on Tariffs and Trade (GATT) talks a couple of years ago. Why? Because they feared that a flood of American movies into France would strangle the relatively small French film industry.

America is good at delivering entertainment to the world, hence its value to our country as a viable commodity. As we enter the 21st century and face increasing technological competition from abroad, American entertainment is going to be a lucrative export for the country. I predict that a significant component of that entertainment export will consist of live-action games that star American actors and actresses. Like the early cinema, however, live-action games have some technical hurdles to clear before they attain widespread popularity: better player navigation and better player interaction.

Enter the Dragon

Live-action games have roots that can be traced back to that (in)famous arcade game Dragon's Lair. I remember it well—as a vidiot in the early 1980s, I dropped way too many quarters into that game at the local pizza parlor (it was one of the first games that demanded 50 cents, which really chapped my hide). Looking back, the game wasn't as exciting as other arcade games of its day, yet one element made it unique: rather than being composed of sprites and tiles, it was an animated cartoon you played off a laser disc.

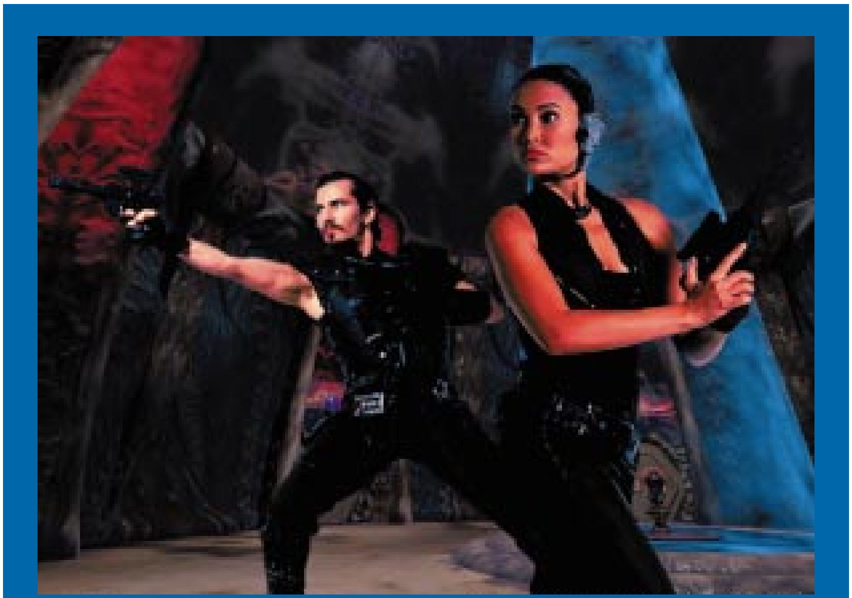
The game was based on a series of short cartoon scenes spliced together in real time and controlled by the player's actions at key junctures in the game. Using liberal amounts of fast-twitch muscle tissue and a good memory, the plot of Dragon's Lair came together, and the player eventually rescued the princess from the dragon. Now that the industry is seeing the convergence of fast video transfer rates, CD-ROM storage, and the ubiquity of personal computers in homes, the consumer market is ripe for live-action games that follow the Dragon's Lair paradigm and feature familiar actors and actresses. Siliwood is starting to capitalize on this market.

What's Siliwood?

Siliwood is the combination of Silicon Valley technical sophistication and the glitzy star power of Hollywood, evidenced in the newest crop of multimedia games. Siliwood is just a buzzword, but the promise that these two industries hold for creating a new entertainment industry is real. At a time when America's competitive edge in software development is being threatened by increasingly educated and efficient third-world countries, the marriage of entertainment and technology will definitely give the resulting products the 21st-century spin they'll need.

I recently had a brush with Siliwood. This past summer I was fortunate to be invited to the making of a new live-action game, The Daedalus Encounter, developed by Mechadeus. Starring Tia Carrere (of *Wayne's World* fame), the project's live footage was shot in a warehouse soundstage near San Francisco's multimedia gulch area before it was brought back to the Mechadeus developers for integration with the interface and the game's logic. It was an interesting film set that relied on the same film techniques as *Star Wars* and *Superman*. The two leads acted out sequences against invisible enemies and traded lines against a blue screen backdrop. Behind the bright lights, cameras, and film crew was a monitor that showed the performers superimposed into rendered scenery. It was quite different than other game development shops I've had a chance to see, needless to say.

Unfortunately, as with Dragon's Lair a decade ago, many of these live-action adventure and role-playing games suffer from a stifling story line that forces players to pursue a limited course of action through the game world. If you screw up at a particular juncture (zigged when you should have zagged, and subsequently got shot by a bullet, for instance), you're forced to back up to the beginning of the scene where you died, listen to the same dialogue again, and correct your mistake ("I *have* to zag this time—I can't bear to hear that scene's dialogue one more time!").



The Daedalus Encounter stars Tia Carrere and Christian Bocher. "Using name talent will enrich the game and its consumer appeal," says John Evershed, Mechadeus's executive producer.

Improving Playability

Just as leaps like sound and color helped movies catch on with the public, technological advancements in live-action games will help them mature and gain popularity. For instance, the repetitive nature of these games will be overcome by sophisticated, highly developed plots containing dozens, hundreds, or thousands of unique solutions.

I know that using a flexible story line was a priority for Mechadeus because the game's story board was posted on a wall and looked like the flow chart from hell: lines from a starting point that connected to a number of different hubs, which in turn connected to more, like a geometric function. The paths between hubs crossed everywhere, and eventually funneled back down to a series of end points—the finishing scenes. This large road map of the game's plot, we were told, allowed players to accomplish tasks in no set order and allowed more flexibility in game play. In addition, a player's "attitude" affected how the story unfolded, so that a Doom-style, shoot-everything strategy revealed a different side to the game than an "I'm O.K., you're O.K., let's be friends" style of play.

Another (probably years away) step forward for live-action games will be the inclusion of sophisticated artificial intelligence, allowing interaction with nonplayer characters (NPCs) that is less scripted and more spontaneous. It would require some fairly intense graphics and sound manipulation to make NPCs say something intelligent

John Rhys-Davies and Mark Hamill, two veterans of George Lucas's films, star in Origins's Wing Commander III.

in the right voice and with the right facial expressions! Perhaps someday in the future, game AI will get good enough to produce game solutions that even the developers hadn't anticipated.

Finally, it's interesting to consider that these games might be the launching point for future acting careers. We might see the rise of some cyber Chaplins and Pickfords. Maybe some lucky ones will begin their rise to fame in the game industry and then cross over to television or movie spots. In the meantime, the faces that have already appeared in games lend credence to these games as an acting vehicle: Jonathan Frakes, Morgan Fairchild, Joe Piscopo, Mark Hamill, Malcolm McDowell, Dennis Hopper, Grace Jones, and Margot Kidder. (And that's just what I could dig up in two minutes from scanning some ads.)

As Siliwood comes into its own, the line between games and movies will rapidly fade. Ads in game magazines already look like blockbuster movie ads, and we've begun to see stars' mug shots alongside blurbs detailing the minimum system requirements. Interactive game drama is here, so forget the theater and renting movies—fire up the Intel nickelodeon. ■

Alex Dunne is contributing editor for Game Developer *magazine.*

Autoplay On!

Nicole Claro

New technology in the game industry evolves at an increasingly rapid pace. Animation tools, accelerator cards, and new online systems are just some of the things shaping the industry today.

Installing and running CD-ROMs on Windows can be an onerous task. You insert the disk, go to the File Manager, click on the corresponding drive, find and execute the installation file—all time that could be spent more productively. Good news for Windows developers and users alike. Microsoft has conceived a new technology that makes CD-ROMs install and run automatically on the Windows '95 operating system. Developers will be able to add the support, called AutoPlay, to any applications they create for use with Windows.

AutoPlay is currently being used in development of titles by Humongous Entertainment and Hummer Winblad Venture Partners. Developers say making this coding investment during the design phase will save time and money in support calls after the product's release. As long as a title has been AutoPlay-enabled, you simply insert the disc in the CD-ROM drive and, after checking for a file named AUTORUN.INF in the root directory, Windows '95 will immediately run the title.

For More Information Contact:
Microsoft Corp.
1 Microsoft Wy.
Redmond, Wash. 98502-6399
Tel: (206) 882-8080
Fax: (206) 936-7329

More Power to the Power Mac

Electric Image Inc. has released ElectricImage Animation System Power Macintosh 2.1, a three-dimensional graphics system designed specifically for computer graphics and animation created on the Power Macintosh. The newest version still incorporates all the features of its predecessor, v. 2.0, but can work much faster. Electric Image says it's seen rendering improvements of between three and eight times faster than the first system. Certain effects can render over eight times faster. The company projects an average of about four to five times faster than the Macintosh version.

The increase in speed will be especially applicable to motion-picture work. In fact, ElectricImage has been used for special effects in *Star Trek: Generations*, *The Mask*, and *Jurassic Park*, as well other films and interactive CD-ROMs. A "snappier" interface also allows the choreography to occur at a faster pace. ElectricImage Power Macintosh 2.1 imports, renders, and animates objects from mulitplatform modeling programs. It includes sync sound animation and blur techniques including Motion Vector, adaptive anti-aliasing, and many plug-ins. It also lets you create lens flares at assigned light sources with elements in the flare controlled from the project window. ElectricImage Power Macintosh 2.1 is priced at $7,495. Registered owners of ElectricImage 2.0 can upgrade for $495.

For More Information Contact:
Electric Image Inc.
117 E. Colorado Blvd., Ste. 300
Pasadena, Calif. 91105
Tel: (818) 577-1627
Fax: (818) 577-2426

Online Onslaught

When I was a teenager, video game play was an isolated, solitary pursuit.

I'd put on headphones (you know, those big, weird, early-80s ones that pumped music into one entire side of your head, rather than just your little ears), lock out the rest of the world, and play Venture for upwards of three hours. My, how things change. Now here's a system that can run games and hook up one player in California and one in Texas. No, it's an e-mail network. No, it's a goldmine of professional sports tie-ins. No, it's a dessert topping and a floor wax. Actually, it's all these things—o.k., maybe not the last two.

Catapult Entertainment recently went online with its XBAND video Game Network, a venture designed to coincide with the release of THQ Inc.'s XBAND Video Game Modem for Sega Genesis. Currently, the XBAND Video Game Network and Modem is available in New York, Los Angeles, San Francisco, Dallas, and Atlanta. The XBAND modem supports Sega Genesis only, and the XBAND Network supports Mortal Kombat, Mortal Kombat II, NBA Jam, Madden NFL '95, NHL '94, and NHL '95 (Will there even be an NHL '95?).

The XBAND Network will go nationwide, and Super Nintendo-compatability will be available by the first quarter of this year. The network features a mail system, an online newspaper, and entertainment, sport, and video game news updates. You (that is, parents) will be able to set controls on number of hours, times of day, and long-distance restrictions. (Three hours a day, no more—after the afternoon T.V. you're not supposed to be watching.)

For More Information Contact:
T-HQ Inc.
5016 North Pkwy. Calabasas
Ste. 100
Calabasas, Calif. 91302
Tel: (818) 591-1310
Fax: (818) 591-1615

Copies in No Time

When I master programming, I'm going to do a series of cool interactive CD-ROMs—my first one is going to incorporate a voice recognition system to teach users different dialects of New York accents (each borough is distinct, you know). And how will I copy my master disk?

MicroTech Conversion Systems has released ImageMaker, a recordable CD duplication system that produces 48 disks per hour. The product uses 12 Yamaha CDR 100 4X drives and is twice as fast as any previous system from MicroTech. With record drives working from a master CD at up to 36MB per minute—writing both 63-minute (550MB) and 74-minute (650 MB) media—it can write a 650MB disk in about 18 minutes.

Each ImageMaker starts out as an 80486 DX-66 computer, SVGA montor, SCSI 1.2 GB drive, and 14.44 baud modem with full remote diagnostics. Individual customer specifications then determine the final version of each ImageMaker, the price of which varies depending on configuration. However, average price is $1,875 per "X" (X being the data transfer rate of CD drives as a multiple of the standard audio CD rate or 150-KB per second).

For More Information Contact:
MicroTech Conversion Systems
940 Industrial Ave.
Palo Alto, Calif. 94303
Tel: (415) 424-1174
Fax: (415) 424-1176

Chips and Cards

Criterion's RenderWare-based applications will now use ATI's 64-bit graphics accelerator cards and chips. RenderWare, used in Criterion's three-dimensional games and virtual reality applications, is the first interactive three-dimensional graphics API for Windows and DOS. RenderWare is designed to provide real-time graphics without the need for a separate three-dimensional accelerator. Used with an accelerator, though, performance is greatly increased.

ATI's mach64 family of chips and cards was designed to take advantage of 486 and Pentium-based systems. RenderWare's mach64 device driver will give RenderWare applications a 30% to 100% boost in performance. The result is a an efficient, low-cost three-dimensional development platform. ATI's accelerator cards range in price from $179 to $699.

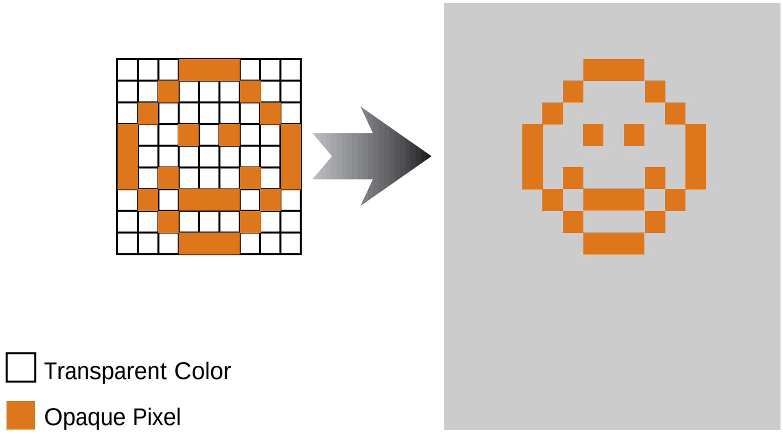
For More Information Contact:
Criterion Software Ltd.
20 Alan Turing Rd.
Guildford GU2 5YF, U.K.
Tel: 44 483 448800
Fax: 44 483 448811

Nicole Claro is production editor for Game Developer *magazine.*

Back

Changing The Rules for Transparent BLTs

Figure 1. A Transparent BLT



When I sit down to write an article, the first question I always ask myself is, “Who is going to read this?” No, I don’t mean, “Who in their right mind would read this?” I mean who is the audience for this article, and how technical are they?

For this column, I'd like the answer to be “experienced programmers,” and I intend to aim the content at just such a readership. My goal is to provide detailed coverage of specific game programming techniques and to present production-quality code, sometimes at the expense of less experienced developers who might want to read the code a few times and step through it in a debugger to see how it works. This is not to say I'll be cryptic, but I'm going to try to move fast enough to keep the advanced people interested, while giving the beginners something they'll need to think about for a bit before grasping all the issues, both explicit and implied. Let me know what you think via the contact information at the end of the article!

Transparency

Transparent block transfers (BLTs—pixel copies) are one of the more useful techniques for game programmers. A transparent BLT can be roughly defined as a block transfer where some pixels are not copied from the source to the destination, leaving destination pixels showing through. The list of effects you can generate with a simple transparent BLT is endless: sprites, floating text or game scores, cursors, shadows, floating maps, and the like. How are transparent BLTs implemented? We'll answer that question with working code, optimize the code, and write a transparent BLT that will handle both WinG DIB orientations as a bonus.

There are a number of ways you can implement transparent BLTs. The most common specifies a single pixel value in the source bitmap (the sprite, if you will) that will not be copied from the source to the destination. The BLT routine examines each pixel and decides whether it is the “transparent color” or whether it's an actual data value that needs to be copied to the destination bitmap, as shown in Figure 1. Other techniques include using a mask to specify which pixels are copied, using raster operations under Windows, and using special bitmap formats (like run length encoding) for the source sprite. When we start optimizing, we'll look into some of these other techniques and how they compare with the base technique.

First, we'll create a relatively naive

transparent BLT. I'm going to write the BLT for use under Windows (my preferred development environment), but it isn't Windows specific and should port to DOS or other platforms without problems. We'll use device independent bitmaps (DIBs), which are just in-memory bitmaps with a header describing their pixel format.

Our naive implementation will read every pixel in the source DIB, check for the transparent color, and optionally write the pixel to the destination DIB. Since we want this code to work well on Windows with WinG, we'll need to deal with the two possible destination "DIB orientations."

Orient Yourself

There are two DIB orientations, top-down and bottom-up. Top-down DIBs are arranged in memory much like the DOS Mode 13h frame buffer or your average DOS bitmap. The pointer to the DIB bits points to the topmost scanline on the DIB, and as the value of the pointer increases, it moves down the DIB surface. On the other hand, bottom-up DIBs are "upside-down," with the pointer referencing the bottom-most scanline, its value increasing as it moves up the DIB surface. Movement across scanlines from left to right is always accompanied by an increase in memory address; only the vertical movement is affected by the orientation. WinG chooses the fastest DIB orientation based on the run-time configuration, so code that expects the best performance must be prepared to deal with either type. This is actually quite easy in practice, and the technique I describe here draws to both orientations without any performance penalty.

Listing 1 shows our first transparent BLT. This code only handles 8 bits-per-pixel DIBs, but could you can easily extend it to other formats. Our initial inner loop looks like this:

```
for(Y = 0;Y < Height;Y++) {
for(X = 0;X < Width;X++) {
  if(*pSourceBits != TransparentColor) {
// not transparent?
         *pDestBits = *pSourceBits;
// copy the pixel
  }
  pDestBits++;
// advance to next pixels
  pSourceBits++;
  }
  pDestBits += DestDeltaScan;
// advance to next dest
  pSourceBits += SourceDeltaScan;
// and source pixels
}
```

We introduce the `DeltaScan` variables (`DestDeltaScan` and `SourceDeltaScan`) to enable top-down and bottom-up drawing. We always start the BLT from the top, and the `DeltaScans` move their respective pointers down the DIB surface from one scanline to the next. We set up the `DeltaScans` to move from the end of one processed span to the beginning of the next span, so we step directly to the next span of pixels to BLT without calculating a new `X` or `Y` offset from the start of the DIB, avoiding multiplies in the loop and other overhead. On top-down DIBs, the `DeltaScan` is positive (the "down" of "top-down" indicates the direction in which a positive pointer increment

Chris Hecker

What's a good concept to follow when you're working with transparent BLTs? If the rules forbid you from getting your images onscreen quickly enough, change the rules!

Listing 1. Simple Transparent BLT

```
#include<windows.h>
#include<assert.h>


void TransparentBlt( BITMAPINFOHEADER *pDestHeader, BYTE *pDestBits,
         int XDest, int YDest, BITMAPINFOHEADER *pSourceHeader,
         BYTE *pSourceBits, BYTE TransparentColor ){
   int DestDeltaScan, DestWidthBytes, DestRealHeight;
   int SourceDeltaScan, XSource = 0, YSource = 0;
   int Width, Height;

   DestWidthBytes = (pDestHeader->biWidth + 3) & ~3;   // dword align

   assert(pDestHeader->biSizeImage);      // insure biSizeImage is set

   if(pDestHeader->biHeight < 0){
      // dest is top-down
      DestRealHeight = -pDestHeader->biHeight;  // get positive height
      DestDeltaScan = DestWidthBytes;        // travel down dest
   }else{
      // dest is bottom-up
      DestRealHeight = pDestHeader->biHeight;
      DestDeltaScan = -DestWidthBytes;       // travel down dest
      // point to top scanline
      pDestBits += pDestHeader->biSizeImage - DestWidthBytes;
   }

   // pDestBits -> top scanline of dest
   // DestDeltaScan -> distance from scan to scan in dest

   // clip source to dest

   assert(pSourceHeader->biHeight < 0);   // assume top-down source DIB
   Width = pSourceHeader->biWidth;
   Height = -pSourceHeader->biHeight;

   if(XDest < 0){
      // left clipped
      Width += XDest;
      XSource = -XDest;
      XDest = 0;
   }

   if((XDest + Width) > pDestHeader->biWidth){
      //right clipped
      Width = pDestHeader->biWidth - XDest;
   }

   if(YDest < 0){
      // top clipped
      Height += YDest;
      YSource = -YDest;
      YDest = 0;
   }
```

Listing 1.

```
if((YDest + Height)>DestRealHeight)
{          // bottom clipped
   Height = DestRealHeight - /
YDest;
}

SourceDeltaScan = /
   (pSourceHeader->biWidth + 3)/
 & ~3;        // dword align

// step to starting source pixel
pSourceBits += /
(YSource * SourceDeltaScan) + /
XSource;

// step to starting dest pixel
pDestBits += /
(YDest * DestDeltaScan) + XDest;

// account for processed span in
// delta scans
SourceDeltaScan -= Width;
DestDeltaScan -= Width;

if((Height > 0) && (Width > 0))
{
   // we have something to BLT
   int X, Y;

   for(Y = 0;Y < Height;Y++) {
      for(X = 0;X < Width;X++) {
         if(*pSourceBits != /
TransparentColor) {
              // not transparent?
             *pDestBits = /
*pSourceBits; // copy the pixel
         }
         pDestBits++;
// advance to next pixels
         pSourceBits++;
      }
      pDestBits += DestDeltaScan;
// advance to next dest
      pSourceBits += /
SourceDeltaScan;
// and source pixels
   }
 }
}
```

travels), and the pointer increases through memory as we process the BLT. On bottom-up DIBs, the pointer needs to decrease to move down the surface, so the `DeltaScan` is negative.

Because we always want the BLT to start at the top of the DIBs, we need the pointers to start there, too. For top-down DIBs, this is no problem; the bits pointer already points to the top scanline. For bottom-up DIBs, we need to move the pointer from the bottom scanline to the top using the following expression:

```
pDestBits += pDestHeader->
 biSizeImage - DestWidthBytes;
```

This adds the size of the DIB in bytes to the pointer—bringing it past the top scanline—and subtracts the width of a single scan to bring the pointer back onto the DIB, leaving it pointing at the beginning of the top scanline.

The last bit of code in Listing 1 (before the actual BLT) clips the source to the destination. We step through the extents, adjusting the source and destination offsets and the width and height when necessary. Finally, if we have pixels to draw after the clip, we go into our loop.

Change the Rules

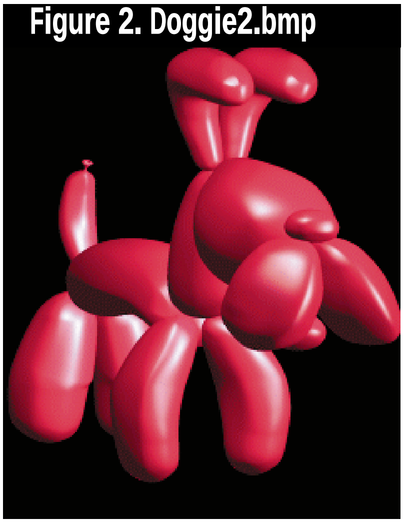
Now that we've got the setup code out of the way, we can try to optimize the inner loop. The first question we must ask is always, “Do I need to optimize the inner loop?” If this code is just supposed to draw a score on top of a bitmap the answer might be no. But if that were the case, this would be a short column, so let's assume this code is our program's bottleneck.

Many people, including myself, make the same mistake over and over again when they start to optimize a piece of code. They usually look at the C version they have working and start rewriting it in assembly language, without taking a step back to ask themselves, “What's this algorithm really doing?”

The key to writing code that runs very fast is *not* to optimize code that obeys the current set of rules and structure you've imposed on it, the key is to *change* the rules. My favorite scene from *Star Trek 2: The Wrath of Khan* is the one where Bones introduces Kirk to a young Starfleet Academy graduate as the only person who has ever aced the final exam, the Kobiashi Maru. When the graduate asks Kirk how it is possible he beat a test that's specifically programmed to be unbeatable, Kirk replies that he sneaked into the testing room the night before his exam and reprogrammed the computer. Kirk would make a great optimizer.

Let's step back and see if we can change the rules. The answer to “What's this algorithm really doing?” is not, “Checking every byte for the transparent color and copying it if necessary.” That just happens to be the way the current implementation works. The real answer for most sprite-type source bitmaps is, “Skipping a bunch of transparent bytes, copying some data bytes, skipping some more, and then doing it all over again.” If we understand this latter answer, a whole range of optimization opportunities open up to us.

Figure 2. Doggie2.bmp



We can take advantage of these opportunities by examining the way our current implementation deals with common input data and looking for ways to change it for the better. Most sprites are irregular shapes with transparent areas on the sides of the bitmap and pixel data in the center. Let's take an example scanline from such a sprite. These values are in hex:

```
FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF 01 01 02 02 03 03 03 04 04 04 03
03 03 02 01 FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF
```

If we assume `FF` is our transparent color, the current code will loop through these 46 bytes and skip 31 of them because they're transparent. In other words, it's spending 67% of its time on this scan deciding to do nothing. The other 33% of the time, it's checking for the transparent color when all it needs to do is copy the data. The amount of transparent color per scanline is obviously dependent on your sprite artwork; I'm using the `doggie2.bmp` bitmap (as shown in Figure 2) supplied with the WinG SDK, which is a fairly typical sprite image.

The “rule change” we need so we can take advantage of the source redundancy is a change to the source bitmap format. Instead of storing each pixel separately (and processing each pixel separately in the BLT), let's use a compression technique to encode pixel spans compactly. This technique is called run length encoding (RLE).

There are many forms of RLE, but most use a few different “token” types to compress bitmaps. Common tokens include `Run` records, which have a value and a number of pixels to copy that value in the destination; `Copy` records, which tell the decompressor to copy a series of pixels from the source like a normal BLT; and `Skip` (or `Jump`) records, which give a number of pixels to skip in the destination. (You can find documentation for one type of RLE format in the Windows SDK documentation under BITMAPINFO. The PCX file format is another RLE format commonly used on PCs.)

I've defined a simple RLE format for compressing our source bitmap, with the tokens shown in Table 1. Each record is a `DWORD` in the source bitmap, with the high word specifying the type of the token, and the low word specifying the run length for each token.

Table 1. RLE Tokens

NEWLINE	0000NNNN	NNNN=number of bytes to next NEWLINE record
SKIPRUN	0001NNNN	NNNN=number of pixels to skip in destination
COPYRUN	0002NNNN	NNNN=number of pixels to copy from source to destination

Here is the same scanline encoded with this RLE format:

```
0000001F 0001000F 0002000F 01 01 02 02
03 03 03 04 04 04 03 03 03 02 01
00010010
```

Now, instead of checking every byte as it transfers, the code can look at each record. If it's a SKIPRUN, the decompressor just increments the destination pointer, skipping over the pixels that wouldn't be drawn anyway (they're transparent in the source), and if it's a COPYRUN, the decompressor copies the pixels without checking for the transparent color. Plus, although we're not concerned with size compression right now, this encoding is only 31 bytes long, compared to 46 bytes for the raw scanline.

Instead of using SKIPRUNs to compress transparent runs, an alternative encoding would use another record type, the COLORRUN. This record encodes a strip of pixels with the same value. If we used COLORRUNs, we'd be able to change the transparent color on-the-fly to make new parts of the source bitmap invisible, but our decompressor would need to treat COLORRUNs differently depending on whether they encoded the transparent color or not.

In an RLE bitmap, each scanline is a different length in memory, so it's sometimes hard to find a certain line. The NEWLINE record makes clipping and subrectangle BLTing much easier. If we want to skip to a certain line, we start at the first scanline and move from NEWLINE to NEWLINE until we get to the one we want.

Listing 2 shows the new transparent BLT, TransparentBltRLE. The setup code for the destination and the clipping calculations stay the same, and both were copied from Listing 1. The actual inner loop looks a lot different from Listing 1 because we need to parse the source RLE. Clipping an RLE bitmap in the X-axis gets interesting; we need to loop over the records until we find one that intersects our BLT rectangle, process the "active" portion (the portion that actually intersects), then start the BLT loop on the next record.

Listing 3 is the RLE compressor, CompressSprite. It's a fairly simple state machine that writes out records on state transitions from SKIPRUNs to COPYRUNs or vice versa. This code could use a bit of work. It doesn't shrink the allocated memory after compressing the sprite, for example. We'll discuss other optimizations to the format below.

Numbers

Listing 2 is significantly faster than Listing 1 when BLTing the doggie. Table 2 contains some performance numbers (for 1,000 iterations). Listing 2 is two times faster than Listing 1. More interesting still, Listing 2 is almost twice as fast as fast32.asm, the assembly language transparent BLT we shipped with the WinG SDK! Fast32.asm is basically an optimized 386 assembly language version of Listing 1, and it uses some special techniques to increase speed, but it's clear that changing the rules gives a much bigger payoff than just brute force optimization or assembly language.

Give Me More

If we want to max out Listing 2, there are a number of other techniques to consider. You'll notice that if a source scanline looks like this:

```
FF 01 FF 01 FF 01 FF 01 FF
```

CompressSprite will generate this:

```
0000002C 00010001 00020001 01 00010001
00020001 01 00010001 00020001 01
00010001 00020001 01 00010001
```

This is definitely a waste of space and almost certainly a speed loss, too. To fix this case, we could extend our RLE format to contain a transparent color, and instead of simply copying the COPYRUN data bytes with memcpy, as we did in Listing 2, we could run the equivalent of Listing 1's inner loop on them. This gives us the benefits of both techniques. We could go even farther and make a new record type, TRANSCOPYRUN, for runs that contain pixels with interspersed transparent colors and keep COPYRUN for plain copies so we don't slow down the normal nontransparent runs. Our compressor would have to be smarter, too. It would look at the data and make a decision about whether it is better to compress a run of transparency with a SKIPRUN or to simply embed the transparent pixels in a TRANSCOPYRUN.

Obviously, well-written assembly language code would make things faster as well, but we could probably optimize the C code without resorting to assembly language and still get some more performance. For example, we could DWORD align our copies, we could unroll once or twice (although on a Pentium especially, this probably wouldn't be a big win and it would make our code bigger and less cacheable), and we could redesign the RLE format so we read less (using DWORD tokens is wasteful in most circumstances). Another option to consider is compiling code to do the transparent BLT, so instead of our sources being bitmaps, they'd be blocks of code that draw the sprite directly. Fast32.asm uses hysteresis to speed things up, and we could put that in our RLE decompressor as well.

Table 2. Timing Numbers

Listing 1	7,100 ms
Listing 2	2,414 ms
fast32.asm	6,950 ms

[Fast32.asm is from the WinG SDK doggie sample application.]

Hysteresis is basically "stickiness," or a tendency to stay the same. For example, when I'm awake, I tend to stay awake for a long time, and when I'm asleep in bed, I stay there, too. You can use hysteresis in transparent BLTing by recognizing that when you're in a transparent run you'll probably be there for a while, and similarly, when you're copying pixels, you'll do that for a bit rather than switching between the two. Of course, our RLE format takes advantage of a lot of this redundancy, so hysteresis might not make much sense for our decompressor.

The only way to know is to understand your data and truthfully answer the question, "What's this algorithm really doing?"

Actually, you need to answer this question in two parts. The first, as we discussed, is understanding what the algorithm is supposed to do, not what the current implementation does. The second part comes in when you've decided on an optimization strategy, and is best summarized by Michael Abrash's quote from *Zen of Code Optimization* (Coriolis, 1994), "Assume nothing!" Time your algorithms, don't assume certain performance. I use the `timeGetTime` API on Windows, which returns millisecond-accurate timings, and Michael uses the Zen Timer, but whatever you do, time your results.

One Last Word

In the future, I plan to cover (in a technical way, naturally) digital wave audio mixing, perspective texture mapping, animated cursors, and maybe some wacky 32-bit programming hacks under 16-bit Windows. Write and let me know what you think or, better yet, post to rec.games.programmer or the CompuServe GAMDEV forum so everyone can join in. I also hang out on BIX in Michael Abrash's ibm.pc/fast.code conference, simply the best place to discuss optimization I've ever seen. ■

Chris Hecker works for a large software company in the Pacific Northwest. He can't mention the name because then he'll need all sorts of disclaimers. It's just a coincidence that he can be reached at checker@microsoft.com or through Game Developer *magazine.*

Listing 2. RLE Transparent BLT (Continued on p. 20)

```
#include<windows.h>
#include<windowsx.h>
#include<string.h>
#include<assert.h>



#define ISSKIPRUN( Record ) (int)((((DWORD)(Record)) & 0xFFFF0000) ==
0x00010000)
#define ISCOPYRUN( Record ) (int)((((DWORD)(Record)) & 0xFFFF0000) ==
0x00020000)

#define RUNLENGTH( Record ) (int)(((DWORD)(Record)) & 0xFFFF)


void TransparentBltRLE( BITMAPINFOHEADER *pDestHeader, BYTE *pDestBits,
         int XDest, int YDest, BITMAPINFOHEADER *pSourceHeader,
         BYTE *pSourceBits, BYTE TransparentColor ){
   int DestDeltaScan, DestWidthBytes, DestRealHeight;
   int XSource = 0, YSource = 0;
   int Width, Height;

   DestWidthBytes = (pDestHeader->biWidth + 3) & ~3;   // dword align

   assert(pDestHeader->biSizeImage);      // insure biSizeImage is set

   if(pDestHeader->biHeight < 0){
      // dest is top-down
      DestRealHeight = -pDestHeader->biHeight;  // get positive height
      DestDeltaScan = DestWidthBytes;        // travel down dest
   }else{
      // dest is bottom-up
      DestRealHeight = pDestHeader->biHeight;
      DestDeltaScan = -DestWidthBytes;       // travel down dest
      // point to top scanline
      pDestBits += pDestHeader->biSizeImage - DestWidthBytes;
   }

   // pDestBits -> top scanline of dest
   // DestDeltaScan -> distance from scan to scan in dest

   // clip source to dest

   assert(pSourceHeader->biHeight < 0);      // assume top-down source DIB
   Width = pSourceHeader->biWidth;
   Height = -pSourceHeader->biHeight;

   if(XDest < 0){
      // left clipped
      Width += XDest;
      XSource = -XDest;
      XDest = 0;
   }

   if((XDest + Width) > pDestHeader->biWidth){
      //right clipped
      Width = pDestHeader->biWidth - XDest;
```

Listing 2. (Continued on p. 21)

```
}

if(YDest < 0){
   // top clipped
   Height += YDest;
   YSource = -YDest;
   YDest = 0;
}

if((YDest + Height) > DestRealHeight){
   // bottom clipped
   Height = DestRealHeight - YDest;
}

// step to starting dest pixel
pDestBits += (YDest * DestDeltaScan) + XDest;

// account for span in delta scans
DestDeltaScan -= Width;

if((Height > 0) && (Width > 0)){
   // we have something to BLT
   int X, Y;
   DWORD *pCurrentSourceScan = (DWORD *)pSourceBits;

   // prestep to starting source Y

   for(Y = 0;Y < YSource;Y++){
      pCurrentSourceScan = (DWORD *)((BYTE *)pCurrentSourceScan +
                          RUNLENGTH(*pCurrentSourceScan));
   }

   for(Y = 0;Y < Height;Y++){
      DWORD *pCurrentSourceRecord = pCurrentSourceScan + 1;

      // prestep to starting source X

      X = 0;

      while(X < XSource){
         X += RUNLENGTH(*pCurrentSourceRecord);

         if(X > XSource){
            // we need to partially process the current record

            int Overlap = X - XSource;
            int ActiveOverlap = (Overlap > Width) ? Width : Overlap;

            if(ISCOPYRUN(*pCurrentSourceRecord)){
               // copy overlap pixels to destination

               // get pointer to data
               BYTE *pCopyRun = (BYTE *)pCurrentSourceRecord + 4;

               // prestep to desired pixels
               pCopyRun += RUNLENGTH(*pCurrentSourceRecord) - Overlap;

               memcpy(pDestBits,pCopyRun,ActiveOverlap);
```

Listing 2. (Continued from p. 20)

```
               }

               // skip to next dest pixel
               pDestBits += ActiveOverlap;
            }

            // skip to next record

            if(ISCOPYRUN(*pCurrentSourceRecord)){
               // skip any data bytes
               pCurrentSourceRecord =
                  (DWORD *)((BYTE *)pCurrentSourceRecord +
                     RUNLENGTH(*pCurrentSourceRecord));
            }

            pCurrentSourceRecord++;              // skip record itself
         }

         X = X - XSource;

         while(X < Width){
            int RunLength = RUNLENGTH(*pCurrentSourceRecord);
            int RemainingWidth = Width - X;
            int ActivePixels = (RunLength > RemainingWidth) ?
                                RemainingWidth : RunLength;

            if(ISCOPYRUN(*pCurrentSourceRecord)){
               // copy pixels to destination

               // get pointer to data
               BYTE *pCopyRun = (BYTE *)pCurrentSourceRecord + 4;

               memcpy(pDestBits,pCopyRun,ActivePixels);
            }

            // skip to next dest pixel
            pDestBits += ActivePixels;

            // skip to next record

            if(ISCOPYRUN(*pCurrentSourceRecord)){
               // skip any data bytes
               pCurrentSourceRecord =
                  (DWORD *)((BYTE *)pCurrentSourceRecord + RunLength);
            }

            pCurrentSourceRecord++;              // skip record itself

            X += RunLength;
         }

         pDestBits += DestDeltaScan;

         pCurrentSourceScan = (DWORD *)((BYTE *)pCurrentSourceScan +
                              RUNLENGTH(*pCurrentSourceScan));
      }
   }
}
```

Listing3. RLE Compressor

```
#include<windows.h>
#include<windowsx.h>
#include<string.h>
#include<assert.h>

#define NEWLINE( Length ) /
((DWORD)(0x00000000 | /
(short unsigned)(Length)))

#define SKIPRUN( Length ) /
((DWORD)(0x00010000 | /
(short unsigned)(Length)))

#define COPYRUN( Length ) /
((DWORD)(0x00020000 | /
(short unsigned)(Length)))

BYTE *CompressSprite(
   BITMAPINFOHEADER *pSourceHeader,
   BYTE *pSourceBits,
   BYTE TransparentColor ){
   int SourceWidthBytes = /
(pSourceHeader->biWidth + 3) & ˜3;
   void *pOutputBuffer = /
GlobalAllocPtr/
(GHND,pSourceHeader->biSizeImage);
   DWORD *pOutputRecord = /
(DWORD *)pOutputBuffer;
   BYTE *pOutputByte;
   int X, Y;

   assert(pOutputBuffer);

   for(Y = 0;/
Y < pSourceHeader->biHeight;Y++){
      int Width = /
pSourceHeader->biWidth;
      enum state { InSkipRun, /
InCopyRun } State;
      BYTE *pSourceByte = /
pSourceBits;
      DWORD *pNewlineRecord = /
pOutputRecord++;
      int LineLength = 4;
      int CurrentRunLength = 1;

      pOutputByte = /
(BYTE *)(pOutputRecord + 1);

      if(*pSourceByte ==/
TransparentColor){
      // we´re starting a skip run
         State = InSkipRun;
         LineLength += 4;
      }else{
      // source is data
         // we´re starting a copy
  run
```

Listing3.

```
         State = InCopyRun;
         *pOutputByte++ = *pSourceByte;
         LineLength += 5;
      }

      pSourceByte++;

      for(X = 1;X < Width;X++){
         if(*pSourceByte == TransparentColor){
            if(State == InSkipRun){              // still in skip run
               CurrentRunLength++;
            }else{                         // changing to skip run
               // write out copy record
               *pOutputRecord = COPYRUN(CurrentRunLength);
               pOutputRecord = (DWORD *)pOutputByte;

               CurrentRunLength = 1;
               State = InSkipRun;
               LineLength += 4;
            }
         }else{   // source is data
            if(State == InCopyRun){              // still in copy run
               CurrentRunLength++;
               *pOutputByte++ = *pSourceByte;
               LineLength++;
            }else{                         // changing to copy run
               // write out skip record
               *pOutputRecord = SKIPRUN(CurrentRunLength);
               pOutputRecord++;
               pOutputByte = (BYTE *)(pOutputRecord + 1);

               CurrentRunLength = 1;
               State = InCopyRun;
               *pOutputByte++ = *pSourceByte;
               LineLength += 5;
            }
         }

         pSourceByte++;
      }

      // finish off current record

      if(State == InSkipRun){
         *pOutputRecord = SKIPRUN(CurrentRunLength);
         pOutputRecord++;
      }else{ // InCopyRun
         *pOutputRecord = COPYRUN(CurrentRunLength);
         pOutputRecord = (DWORD *)pOutputByte;
      }

      *pNewlineRecord = NEWLINE(LineLength);

      pSourceBits += SourceWidthBytes;
   }

   return (BYTE *)pOutputBuffer;
}
```

Back

Ten Techniques for Faster Image Drawing

Once again, we are off on our quest for the fastest graphics performance possible. This time, we are going to take a look at image drawing routines and the factors that affect performance. In the process, we'll examine many factors that negatively affect performance and the techniques you can use to minimize them. While I am going to use Mode X image drawing as our test example, most of these factors apply to all graphics modes from EGA 16 color to Super VGA True Color.

For the techniques described in this article, let's suppose that we are writing a game that needs to redraw a tile-based background, like Ultima VI or Gauntlet. We are using 256-color Mode X at 320-by-200 pixels of resolution, and our background is 18-by-12 tiles in size. This makes for an update area of 288 pixels by 192 pixels or 55,296 total pixels. Tiles are stored in memory line by line from left to right, top to bottom.

For testing, I will time each routine on five different CPU and video card combinations to get a good cross-section of the machines a game designer would expect his or her programs to run on today. Representing the lower end of most systems is a 40MHz 386 with a slow ISA Trident VGA card. To contrast the variations between VGA cards is another 40MHz 386 with a fast ISA ATI Graphics Ultra Plus. Next, a 33MHz Hewlett Packard 486SX with an S3-based VGA on the motherboard and a 66MHz 486DX2 computer with a Diamond Stealth 32 on a VLB local bus. Finally, we will test on a 90MHz Pentium computer with a Diamond Stealth 64 PCI Bus video card. To reduce the impact of memory caches in our test results, each routine will be called 10,000 times in a loop using the same data buffers. The final results are summarized in Table 1.

We will start by using the Mode X drawing code from the article "Mode X Revealed" (Dec. 1994) to write a straightforward tile-drawing routine in Borland C, as shown in Listing 1.

Our tile-drawing routine is pretty simple and straightforward, wouldn't you agree? It's small, flexible, uses only integers, and has just three executing statements. It's also horribly slow and a perfect candidate for our first speedup technique.

Speedup Technique #1

Do not call a separate pixel-plotting routine in image-drawing code. Use inline code or functions instead. Think about how many calls it takes to draw one screen or image.

The crux of this technique can be summed up in one word: overhead. Every time you call a function or procedure, your program incurs the overhead of pushing parameters, executing a far call, setting up a stack frame, decoding parameters, cleaning up the stack, and executing a RETurn instruction. For something simple, like plotting a single pixel, the time spent handling the call can approach the time spent executing the core of the function.

Let's think about this: for each screen we redraw, 55,296 calls are made. Intel's timings show a perfect

Matt Pritchard

What affects image-drawing speed? Here are ten tricks to help you get the fastest graphics performance—in Mode X as well as EGA and Super VGA modes.

world time of 24 CPU cycles on a 486 for only the call, stack frame assignment, and return. In reality, that time can be more like 35 cycles per call. Using a call for each pixel can eat up nearly 1.3 to 2 million CPU cycles per screen! These are CPU cycles we want to use for other things.

Now we will rewrite our tile-draw routine, putting the set_point routine inline using Borland C's inline assembly language command, shown in Listing 2.

Although we added code to preserve registers, this version showed considerable improvement on our test machines. We recorded speed increases of 22%, 31%, 39%, 38%, and 10% for about a 30% average increase. Put another way, call overhead made up the difference between 16 frames per second and 20 frames per second! Call overhead is only the tip of the iceberg. Lurking in both examples is one of the biggest obstacles to graphics performance: the OUT instruction.

Speedup Technique #2

Minimize the number of OUT instructions to the video card. Switch video planes or memory banks as infrequently as possible.

OUT instructions are the only way to access the various control registers on a graphics card, so they can't be eliminated. You need them to set bit planes, write masks, and select banks in just about every mode except mode 13h and CGA modes. They are necessary, and they are *slow*.

According to Intel, an OUT instruction takes 10 CPU cycles on a 386 and 16 cycles on a 486. While it's not the fastest instruction, 16 CPU cycles doesn't seem that serious. This wouldn't be a problem if the timings told the whole story, but they don't. The 16 CPU cycles reflect only the CPU processing overhead. No mention is made of the I/O bus, or the VGA card itself. Even on a local bus video card, OUTs still have to go through the 8MHz bus protocol, which was designed for the original IBM PC-AT and PC/XT. In addition to the ISA bus, the VGA card itself has to respond to the OUT and signal the computer that it has properly processed it. Finally, as if this wasn't bad enough, under most memory managers or operating systems, the CPU has to check the OUT instruction for validity against the I/O permission bitmap, a process that can add 10 to 20 more cycles.

Given all these factors, the OUT instruction actually takes anywhere from 30 to more than 70 CPU cycles, with a wide variance due to differences in CPU, motherboard, I/O bus, and video card. With this in mind, how many OUTs do we really need to draw one of our tiles? Four. One OUT to select each of the four Mode X planes. If we rewrite our tile-drawing code to plot all pixels on one plane before going on to the next, it should be much faster since we will have cut the number of OUTs per time from 256 to 4, about a 98% reduction.

Listing 3 shows our tile-drawing code, rewritten to minimize OUTs. Timing this version, we see speed increases of 43%, 19%, 51%, 50%, and 127% over the last version. The scores on the

Listing 1. A Straightforward Approach in C

```
void draw_tile (char* theTileData, int Xpos, int Ypos, int Xwide, int Ywide)
{
     int   x, y;
     int   c = 0;

     for  (y = 0; y < Ywide; y++) {
          for (x = 0; x < Xwide; x++) {
               set_point (Xpos + x, Ypos + y, theTileData[c++]);
          }
     }
     return;
}
```

Listing 2. Putting the Pixel Plot Function Inline

```
void Fast_Draw_Tile (char * TheTile, int Xpos, int Ypos, int Xwide, int
Ywide)
{
    int x, y, z;
    int c = 0;

    for (y = 0; y < Ywide; y++) {
       for (x = 0; x < Xwide; x++) {
          asm {
            Push    SI                  /* Save SI & DI because */
            Push    DI                  /* the compiler is using them */

            Les     DI, dword ptr CURRENT_PAGE
            Mov     AX, y               /* Get Line # of Pixel */
            Add     AX, Ypos            /* Add Y position of Tile */
            Mul     SCREEN_WIDTH        /* Get Offset to Line Start */
            Mov     BX, x               /* Get X pos inside of tile */
            Add     BX, Xpos            /* Add X position of Tile */
            Mov     CX, BX              /* Save to get shift Plane # */
            Shr     BX, 2               /* X offset (Bytes) = Xpos/4 */
            Add     BX, AX              /* Offset = Offset + Xpos/4 */
            Mov     AL, 2               /* Select Map Mask Register */
            Mov     AH, 0x01            /* Start w/ Plane #0 (Bit 0) */
            And     CL, 3               /* Get Plane Bits */
            Shl     AH, CL              /* Get Plane Select Value */
            Mov     DX, 0x03C4          /* then Select Register */
            Out     DX, AX              /* Set I/O Register(s) */
            Mov     SI, TheTile         /* Point to Tile */
            Add     SI, c               /* Get current data byte # */
            Inc     c                   /* Advance to next byte */
            Mov     AL, byte ptr [SI]   /* Get Pixel Color */
            Mov     ES:[DI+BX], AL      /* Draw Pixel */

            Pop     DI                  /* Restore SI & DI */
            Pop     SI
          }
       }
    }
    return;
}
```

386 machines illustrate how big the variation can be due to the video card alone. The speedy Pentium showed how factors outside the CPU can slow it down. Compared to the routine we started with, we are averaging about a 100% improvement.

The next couple of speedup techniques are less obvious, but share the same basic philosophy: reduce overhead by removing unnecessary or redundant portions of code.

Speedup Technique #3

For frequently drawn images, use specific routines with hard-coded values instead of general-purpose routines. You can choose the variables you encode as constants and remove functionality that you don't need.

You should think about the routines you use, especially if they come from third-party libraries. Ask these questions about your routines:

- Do they have inputs that will always be the same?
- Do they have features that will never be needed?
- Are they more flexible than my program will ever be?

If you answer yes to these questions, then you should consider writing custom versions of those routines for the specific task at hand.

Imagine a library routine that has neat features like a transparent color, or clipping the image to a rectangle. But what if we know that certain images are always solid or are never going to need to be clipped? In this case, it means every time we use them, the computer spends part of its time checking for things that will never happen. With a little up-front planning and awareness, a good game designer will avoid overhead by implementing alternate versions of those routines which don't have features that will go unused.

In our tile-drawing routine we pass in two variables, `Xwide` and `Ywide`, that tell us how big the tile image is. But, according to our specifications, the only size tile we will ever draw is 16-by-16. We can make a version specifically for 16-by-16 tiles and save the overhead of passing those variables every call. If we need to draw tiles in other sizes, we still have our general-purpose routine, or we could write another specific-sized routine.

While looking over the tile-drawing code with this in mind, we see another situation where the code is doing something that may be unnecessary. Why are we doing a MULtiply every time we plot a pixel? This leads us to yet another speedup technique.

Speedup Technique #4

Precalculate frequently needed information whenever possible. Use lookup tables instead of calculations. Image drawing should be about transferring data, not calculating it.

Taking a closer look at our code, we see that for every pixel plotted, one MULtiply instruction is performed when calculating the display address. That is not terrible, but the multiply takes anywhere from 10 to 26 CPU cycles, and we do it 55,296 times for every screen. If we stop and think about what is being multiplied, we are hit with this realization: we are multiplying the same 200 numbers over and over again. Because we know how wide the screen is going to be, and we know what Y values the pixels will be, why not do all the multiplying once and save the results in a table? This method is known as using a lookup table.

By using a lookup table in our tile-drawing routine, we can replace the MUL instruction that takes 10 to 26 cycles with a lookup sequence that takes two to four cycles. Even with timings that depend on the exact numbers multiplied, we should easily see savings of at least a half million cycles per screen drawn.

Our tile-drawing routine doesn't actually do justice to the benefits a lookup table can provide. More complex calculations such as square roots, sines and cosines, vectors, rotation, and scaling factors, can take hundreds of CPU cycles. But these, too, can be replaced with lookups that take only a couple of cycles. It should be no surprise that graphics-intensive games like Wolfenstein 3-D, Doom, TIE Fighter, Wing Commander, and many more make extensive use of lookup tables.

Table 1. Tile Draw Routing Timings

All times in BIOS Ticks/10,000 tiles
Smaller Number = Faster Routine

Machine:	386DX/40	386DX/40	486SX/33	486DX266	Pentium 90
Routine:	Trident VGA	ATI VGA	S3 VGA	Stealth 32	Stealth 64
Listing 1:	375	291	201	99	65
Listing 2:	307	222	145	72	59
Listing 3:	214	186	96	48	26
Listing 4:	179	152	72	35	19
Listing 5:	56	25	9	4	4

Now, let's redo our tile-drawing routine and apply speedup techniques 3 and 4, shown in Listing 4.

Timing this version again shows

Table 2. VGA Memory Timings

Timings are in Microseconds
Smaller Numbers = Faster Time

Machine:	386DX/40	386DX/40	486SX/33	486DX2/66	Pentium 90
Routine:	Trident VGA	ATI VGA	S3 VGA	Stealth 32	Stealth 64
8-Bit Writes:	1096	618	176	205	153
16-Bit Writes:	1094	618	176	205	153
16-Bit Odd Writes:	2180	1171	343	325	308

Listing 3. Minimizing the Number of OUT Instructions

```
void Faster_Draw_Tile (char * TheTile, int Xpos,
                   int Ypos, in Xwide, int Ywide)
{ int x, y, p, c;
     for (p = 0; p < 4; p++) { asm {
            Mov     AL, 2               /* Select Map Mask Register */
            Mov     AH, 0x01            /* Start w/ Plane #0 (Bit 0) */
            Mov     CX, p               /* Get Plane # */
            Add     CX, Xpos            /* Adust to Image Xpos */
            And     CL, 3               /* Get Plane Bits */
            Shl     AH, CL              /* Get Plane Select Value */
            Mov     DX, 0x03C4          /* then Select Register */
            Out     DX, AX              /* Set I/O Register(s) */
        };
        for (y = 0; y < Ywide; y++) {
           c = y * Xwide + p;
           for (x = p; x < Xwide; x+=4) { asm {
                Push    SI                  /* Save SI & DI because */
                Push    DI                  /* the compiler is using them */
                 Les     DI, dword ptr CURRENT_PAGE
                Mov     AX, y               /* Get Line # of Pixel */
                Add     AX, Ypos            /* Adjust to Image Y pos */
                Mul     SCREEN_WIDTH        /* Get Offset to Line Start */
                Mov     BX, x               /* Get X pos inside of tile */
                Add     BX, Xpos            /* Add X position of Tile */
                Shr     BX, 2               /* X offset (Bytes) = Xpos/4 */
                Add     BX, AX              /* Offset = Offset + Xpos/4 */
                Mov     SI, TheTile         /* Point to Tile */
                Add     SI, c               /* Get current data Byte # */
                Add     c, 4                /* Advance to next in plane */
                Mov     AL, byte ptr [SI]   /* Get Pixel Color */
                Mov     ES:[DI+BX], AL      /* Draw Pixel */
                Pop     DI                  /* Restore SI & DI */
                Pop     SI
               }
          }
       }
    }
    return;
}
```

healthy speed increases of 20%, 22%, 33%, 37%, and 37%. Overall, we have achieved about a 100% speed increase on the 386 machines, 180% on the 486, and 240% on the Pentium. It seems like we are running out of techniques that involve only modifying the code. It's time to turn our attention toward the actual display data and the VGA card itself. Understanding in detail how the CPU, memory, and VGA card work and interact with each other will allow us to uncover facts that we can exploit to further improve our routines.

Memory, it turns out, is the key. We know that video memory is slower than normal RAM, so what's the best way to access it? To study this, I've turned to a tool that you can find on your favorite bulletin board or online service: Michael Abrash's Zen Timer. I've used the Zen Timer to time how fast video memory can be written using various methods. The results are summarized in Table 2. You will notice that 16-bit writes take the same amount of time as 8-bit writes.

Looking at our tile-drawing code, we are using 8-bit writes to draw one pixel at a time when we could be using 16-bit writes to draw two pixels in the same amount of time. This brings us to our next speedup technique.

Speedup Technique #5

When writing data to the VGA card, use 16-bit writes whenever possible. They take the same amount of time as 8-bit writes, but transfer twice as much data.

You'll notice that I didn't list timings for 32-bit writes. Thirty-two bit writes bring up some other issues, such as bus type and `REP MOVSD` interfering with sound DMA on some 386 systems. We are going to save this issue for another time, when we can examine 32-bit graphics in detail.

In Mode X, writing a 16-bit value will plot two pixels that are four pixels apart instead of right next to each other. But our image data is stored linearly. We can either gather the two pixels together before each write or use this next speedup technique.

Speedup Technique #6

Arrange your image data in the form it will be drawn to video memory. For Mode X or 16-color modes, separate and store image data by video planes.

The way we store our image data is up to us, the game designers. Those decisions dictate how our graphic routines must work. This gives us another area where we can design our routines to work as efficiently as possible. Looking at the video memory timings some more, we come across a possible hitch with 16-bit writes. When a 16-bit value is written to an odd video memory address, it takes twice as long as an even address. The reason for that lies in how the VGA card and the bus work together. When the data written spans a 16-bit boundary, the bus splits it into two separate writes. Further testing shows that on the local bus video cards tested, 16-bit writes slowed down only on odd addresses that cross double word boundaries. Aligning write data to the size of the video bus (16 or 32 bits) gives us another speedup technique.

Speedup Technique #7

When writing images, align the data on word and double word boundaries whenever possible. It may be advantageous to have separate routines for even and odd image destinations.

This creates a problem for routines that can draw an image at any position on the screen. Some positions will start on even addresses, and some will start on odd addresses. The images drawn 16 bits at a time on odd addresses will be slower than those on even addresses. Depending on the circumstances, it may be advantageous to have two separate drawing routines, one for even destinations and one for odd destinations. In the case of our tile drawing, the positions we chose for the tiles are at even addresses only, so we can optimize our code for it.

While system RAM is faster than video RAM, data alignment works the same way. Most CPUs read 32 bits of aligned data at a time, even if only 8 bits are needed. A 16- or 32-bit read that spans two 32-bit blocks will be broken into two separate reads. This gives us another speedup technique that exploits memory alignment.

Speedup Technique #8

Try to store image data so that it will be aligned on 32-bit boundaries in system memory. Pad structures and tables so the data after it will be aligned in system memory.

This may seem like a repeat of Technique #6, but it's not. We could reorder our image data and store it in

Listing 4. Using Lookup Tables and Constants

```
void EF_Draw_Tile (char * TheTile, int Xpos, int Ypos)
{
    int x, y, p;
    int c = 0;

    for (p = 0; p < 4; p++) {
        asm {
            Mov     AL, 2               /* Select Map Mask Register */
            Mov     AH, 0x01            /* Start w/ Plane #0 (Bit 0) */
            Mov     CX, p               /* Get Plane # */
            Add     CX, Xpos            /* Adust to Image Xpos */
            And     CL, 3               /* Get Plane Bits */
            Shl     AH, CL              /* Get Plane Select Value */
            Mov     DX, 0x03C4          /* then Select Register */
            Out     DX, AX              /* Set I/O Register(s) */
        };
        c = p;                          /* we can do this because */
        for (y = 0; y < 16; y++) {      /* we know the tile width */
           for (x = p; x < 16; x+=4) {  /* and the tile height! */
              asm {

                Push    SI              /* Save SI & DI because */
                Push    DI              /* the compiler is using them */

                Les     DI, dword ptr CURRENT_PAGE
                Mov     BX, y                   /* Get Line # of Pixel */
                Add     BX, Ypos                /* Add Start Y position */
                Add     BX, BX                  /* Scale to word offset */
                Mov     AX, SCREEN_OFFSET[BX]   /* Lookup in Table */

                Mov     BX, x               /* Get Xpos */
                Add     BX, Xpos            /* Add in Image X Start */
                Shr     BX, 2               /* X offset (Bytes) = Xpos/4 */
                Add     BX, AX              /* Offset = Offset + Xpos/4 */
                Mov     SI, TheTile         /* Point to Tile */
                Add     SI, c               /* Point to correct byte */
                Add     c, 4                /* Advance to Next in plane */
                Mov     AL, byte ptr [SI]   /* Get Pixel Color */
                Mov     ES:[DI+BX], AL      /* Draw Pixel */

                Pop     DI                  /* Restore SI & DI */
                Pop     SI

              }
          }
       }
    }
    return;
}
```

Listing 5. Final Version (Continued on p. 31)

```
ASM_STACK    STRUC
                DW  ?,?,? ; saved BP, SI, DI
                DD  ?     ; Caller Return Address
    ADT_Ypos    DW  ?     ; Ypos of Tile to Draw
    ADT_Xpos    DW  ?     ; Xpos of Tile to Draw
    ADT_Tile    DW  ?     ; Near Prt to Tile Image
ASM_STACK    ENDS

ADT_DRAW_PLANE MACRO
    REPT 15
        MOV     AX, [SI]        ; Get 2 Pixels
        MOV     CX, [SI+2]      ; Get 2 More Pixels
        MOV     ES:[DI], AX     ; Write 2 Pixels
        ADD     SI, 4           ; Update Source Address
        MOV     ES:[DI+2], CX   ; Write 2 More Pixels
        ADD     DI, 80          ; Advance to next line
    ENDM
        MOV     AX, [SI]        ; Get 2 Pixels
        MOV     CX, [SI+2]      ; Get 2 More Pixels
        MOV     ES:[DI], AX     ; Write 2 Pixels
        ADD     SI, 4           ; Update Source Address
        MOV     ES:[DI+2], CX   ; Write 2 More Pixels
ENDM

ADT_ADVANCE_PLANE MACRO
        ROL     BH, 1       ; Rotate Map Mask
        ADC     BP, 0       ; Adjust Start Address
        MOV     AX, BX      ; Select New Video Plane
        OUT     DX, AX      ; By OUTing to Map Mask
        MOV     DI, BP      ; Start over at top
ENDM

ASM_DRAW_TILE   PROC    FAR

    PUSH    BP, DI, SI          ; Preserve Registers
    MOV     BP, SP              ; Set up Stack Frame

    MOV     DX, 03C4h           ; VGA Map Mask Register
    MOV     CX, [BP].ADT_Xpos   ; CX = Xpos
    MOV     AX, CX              ; AX = Copy of Xpos

    LES     DI, dword ptr CURRENT_PAGE
    MOV     SI, [BP].ADT_Tile          ; DS:SI - Tile Data
    MOV     BX, [BP].ADT_Ypos          ; BX = Ypos
    ADD     BX, BX                     ; Scale to Word Offset
    ADD     DI, SCREEN_OFFSET[BX]      ; Get Start of Line
    SHR     AX, 2                      ; Add in Xpos / 4
    ADD     DI, AX                     ; DI = Final Address
    MOV     BP, DI                     ; Save to start each plane
```

system memory on odd memory addresses. We have to look at where the image data is coming from as well as to where it is going. When our images are an odd width, it may be advantageous to add "padding" bytes in between each line of data, so the routine that draws a line can be assured of the fastest possible reads from system memory.

Taking all of this knowledge about how memory and the VGA card works, we can go back and rewrite our tile-drawing code again. Just because we did general purpose optimizations before, doesn't mean we shouldn't look at them again. Now that we have examined our needs in detail, we have more information to work with. For example, because we know the size of our images, we can apply a technique called "loop unrolling." This gives us another speedup technique.

Speedup Technique #9

Don't forget about normal assembly language optimizations. After applying other techniques, your code may have changed to where you can use standard techniques such as loop unrolling, branch elimination, and instruction substitution. Gear your optimizations toward the 486 and Pentium systems; 286 systems are all but dead now, but many optimizations are still oriented toward them.

Taking all we know into consideration, we can write our 16-by-16 tile-drawing code, shown in Listing 5. The code is rewritten completely in assembly language, with unrolled loops, image data stored by planes, and aligned 16-bit writes.

After testing, we see that image data and alignment techniques really do work. This time our results are even better, as shown in Table 1. The speed improvement over our original routine is amazing! We have achieved total increases of 500% and 1,000% on the 386 machines, 2,000% on the 486 machines, and 1,500% on the Pentium. But in terms of results, we are still doing exactly the same thing. At this point, some of our results are looking almost suspect. The test routines are somewhat idealized because of the attempts to nullify the cache's impact, but there is no denying that we can get huge performance increases on every system by applying all of our speedup techniques.

Are there more ways to improve our tile-drawing code? I am sure there are. The fact that we have not discussed them here doesn't mean they don't exist. With that thought, I give you a final speedup technique.

Listing 5. (Continued from p. 30)

```
    AND     CL, 3               ; Get Plane Bits
    MOV     BX, 1102h           ; Map Mask + Plane Select Bits
    SHL     BH, CL              ; Rotate into position
    MOV     AX, BX              ; Select video write plane
    OUT     DX, AX              ; By OUTing to Map Mask


   ADT_DRAW_PLANE              ; Draw 16 Lines of 4 pixels
    ADT_ADVANCE_PLANE           ; Select Next Mode X Plane
    ADT_DRAW_PLANE              ; Draw 16 Lines of 4 pixels
    ADT_ADVANCE_PLANE           ; Select Next Mode X Plane
    ADT_DRAW_PLANE              ; Draw 16 Lines of 4 pixels
    ADT_ADVANCE_PLANE           ; Select Next Mode X Plane
    ADT_DRAW_PLANE              ; Draw 16 Lines of 4 pixels


 POP     SI, DI, BP          ; Restore Saved Registers
    RET     6                   ; Exit and Clean up Stack

ASM_DRAW_TILE   ENDP
```

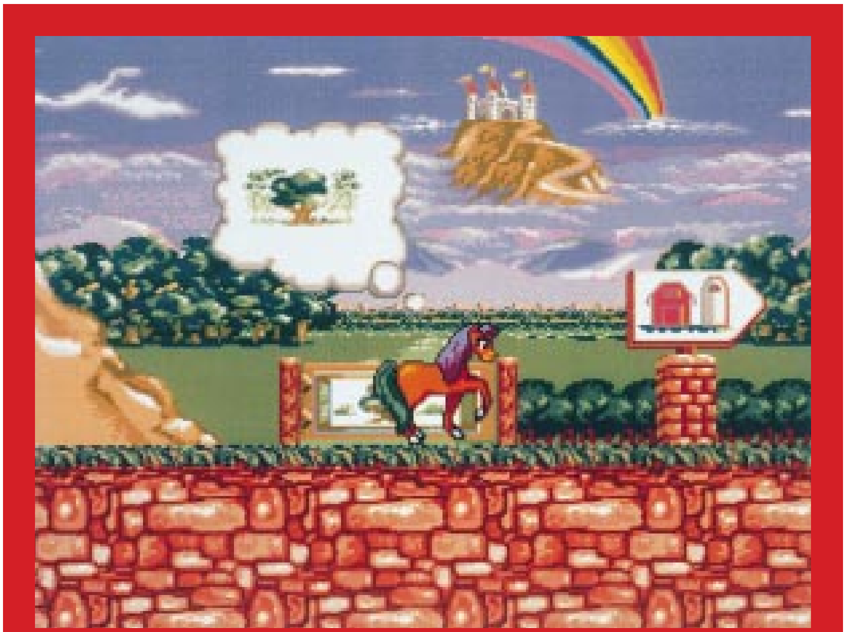
Speedup Technique #10

Never assume your routines are as fast as possible. Always keep your mind open for new ways to improve your code.

By keeping an open mind we learn more and discover more. Perhaps you have a speedup technique that I've overlooked. If so, why not drop a line to *Game Developer* and share it with us. Until next time, happy hacking! ■

Matt Pritchard is a software developer for Lacerte Software in Dallas, Texas, and the author of MODEX110, a comprehensive freeware Mode X library. You can reach him via e-mail at matthewp@netcom.com or through Game Developer.

Beyond Chrome and Sizzle



More pink, less guts? Crystal's Pony Tale, a game designed for girls under eight years of age by Sega's girls task force of women developers, is one of the first efforts to target female gamers. While little girls might think it's cute, others say it barely scratches the surface of what girls and women want in a game-playing experience.

Hey, professionals in the game development industry, I'm here, okay? I'm an adult woman somewhere between the 20-something generation Xers and the successful baby boomers, I own a computer, and I like to play computer games. I like Doom and Myst and Tetris, and other games too. But you people don't think I exist. I've even heard some of you say so. And frankly, I'm rather miffed about it.

But I guess I understand. Women haven't been big purchasers of computer games and neither have our younger sisters. If we do play games at all, we don't play them with the obsessive passion our male counterparts have shown, and you don't see us hanging out in the arcades jockeying for our turn at the joystick. You say we stop playing games at age 12, we're not interested in computers, some of you even say we're not competitive. So why bother making games that appeal to us. It's all *our* fault, not yours.

Well, I say things are changing. Many of us own and use computers now, and we are just beginning to get interested in computer games. So if you build them, we will play. But you've got to give us more than Barbie and pink interfaces, which is what some of you have done. While we'll play Mortal Kombat (and like it) it's going to take something else to really get us hooked—all of us, not just girls and women—but all of us "nontraditional" game players who aren't responding to the genre of shoot-em-up, beat the clock, flesh flying, space invading, dungeons and dragons, psycho macho, occasionally misogynistic stuff you've been creating for the past 10 years—the stuff that's geared to young boys and 18-to-35 year old males.

Computers at Home

As the computer moves into the hands of more women and families, and games move from boy-dominated arcades to the privacy of our own homes, the market of potential game players becomes larger. And perhaps because half of the general population is female, appealing to a larger population of game players has become a gender issue more than an issue of taste. As game developers look to women and girls to broaden their opportunities, the answer to the Freudian question, What is it that women want? is becoming a rather important topic. Some even call it the Holy Grail of the '90s.

Game developers have been looking to educational and psychological studies, market research, and their own hunches to answer this question, and what they've found suggests that women and girls want something different in a computer or video game than what boys and men want. Some research shows these different game play styles begin to emerge at around age eight, some research points to differences as early as age four. Here are some of the findings.

Girls and women like computer games more than video games and enjoy playing games at home.

A study conducted in 1993 by the University of British Columbia's computer science department and funded by Electronic Arts revealed that girls do enjoy playing computer games and that many of them are hip to the titles that their male schoolmates are into. In the study, researchers observed and interviewed game players aged 3 to 18 during their visits to a video and computer games display at Vancouver's Science World B.C. museum. Girls interviewed said they liked playing games at home more than in a boy-dominated arcade. If boys were swarming in front of a game console, girls didn't go up and ask for a turn.

Girls who were most comfortable playing computer and video games came from households with computers or game consoles in them. If they had a video game console and a computer in the house—and games on each—they liked computer games more than video games, often expressing that playing on the computer was more worthwhile.

Girls and women like characters and story lines more than fast action.

The University of B.C. study showed that girls were interested in story lines and character, preferring to discuss these things when talking about games instead of how high they scored, which is what boys liked to do. Girls often liked to discuss the relationships between the characters and often gave gender to androgynous characters.

"Girls pay much more attention to the real world," says Barbara Lanza, a game editor for Byron Preiss Multimedia. "Girls like to find out the story behind a game—what was really going on, what were the characters like, were the characters like them, would they have fixed that problem differently? Even girls' books are like that. If it's a story of a girl whose parents are going through a divorce, who's 13 years old and facing her very first crush, and who might possibly get kissed before the book is over, you have one hot item on your hands."

What kind of characters do girls like? They tend to like female characters their age. Young girls also like fuzzy, cute creatures. "We found that Sonic [the Hedgehog] is fairly gender neutral," says Diane Fornasier, group marketing director for Sega's Genesis and Game Gear games. "The girls like him a lot because he's cute and cuddly, and the boys like him a lot because he's fast." And both genders like him, Fornasier says, because he has an attitude.

Girls and women want the game playing to seem worthwhile.

While boys played video games to "beat the game," many female game play-

by Barbara Hanscome

Are games designed to appeal to women and girls a "separate but equal" kind of entertainment—or the beginning of a new game genre that will appeal to everyone?

ers in the University of B.C. study thought that wasn't the best use of their time. One girl interviewed in the study said she'd play "mindless" games, referring to Nintendo, only after playing "mind" games, referring to Where in the World is Carmen San Diego.

Other research points to this too, especially as girls get older. "At about age 16 or 17, girls are not all that interested in playing games for the sake of games," explains Solange Van Der Moer a multimedia marketing consultant. "By that age, young women start looking at computers as tools rather than as entertainment devices. They will play for diversion, they'll play to get information, they'll play something like Tetris, and they'll play simulation games like Sim City because it accomplishes something." Van Der Moer says creative games that involve story writing, poster making, and rock video design, are popular with this age group.

This desire for productivity or worthwhile play might be why girls aren't into the repetitive aspect of some games as much as boys are. "If a game is hard and it means they have to play it over and over again, it might bore the hell out of an adult woman," explains Lanza, "but a 10-year-old male doesn't mind this at all. This is the same way that he learns everything. With girls, it's like (sigh) what's the point?"

Girls and women don't like time limits. Game designers agree that most boys and men like the intensity of a time limit when playing a game, while most girls don't. They prefer to explore things at their own leisure. They like to be able to leave a game and come back to it, and to take the time they need to solve a puzzle or mystery or get to the next level. Games such as 7th Guest and Myst and many of the newer "gender neutral" games for children don't have time limits at all.

What Everybody Likes

But what's also interesting from this research is what boys and girls and men and women all like (and dislike) in computer and video games—things that stray from the tried and true, shoot-em-up-and-score-high formula most games have followed.

Collaboration and Socializing. Girls enjoy collaboration and socializing during game play and so do boys. In the University of B.C. study, girls enjoyed having other girls around them while they played video games and were highly social when playing a game together. The girls talked about many subjects and often encouraged each other, appearing to enjoy themselves just as much in between turns as when they were playing the game itself.

The University of B.C. study showed that this social aspect of game play is also important to boys. Boys might compete more aggressively in a group, but the boys they interviewed said they enjoyed playing games more when they played with friends, and they often collaborated on how to solve puzzles or get a high score.

Annie Fox, an independent game designer who has developed games for Electronic Arts and Humongous Entertainment, says that both boys and girls enjoy collaborating with characters within a game if given the opportunity. Fox often has the characters in her games turn to the screen and ask the player what to do. "It makes the game inclusive, it makes it more intimate." And boys seem to like this as much as girls.

Challenge Not Frustration. Both boys and girls in the University of B.C. study could articulate the difference between a game that was challenging and a game that was too difficult. Boys and girls interviewed said they liked mental challenge, but if the game was too hard, the children lost interest.

Confidence may be a bigger issue for girls. Girls in the study were very critical of their playing abilities, often saying things like, "I'm so terrible at this," even before they'd given a new game a try. They would stick to the games they mastered or had played before more often than they would venture forth and try a new game.

Several studies show that between the ages of 11 and 13, girls begin to question their self esteem, and socially conscious developers agree that games igniting a sense of accomplishment and victory are attractive to girls at this age, as well as important.

Nonviolence. Studies show that violence in games generally isn't appealing to girls, but that most boys like it. But the University of B.C. study pointed to several exceptions to this video game rule. Many of the violent games boys liked were also thinking games, and some boys abhorred violence altogether: one boy hated the fact that he had to blow up the fuzzy creatures in the game Lemmings to get to the next level, and he apologized to each one before he destroyed it.

Interactivity. Interactivity—the element that has practically become a buzzword in the industry—is something people of all ages and genders seem to like. Developers of children's games use interactive elements often. "It doesn't matter which gender it is, I think all kids love to feel empowered," says Fox. "To know that your choice changed the story is very exciting for kids."

In EA's game Madeline, for example, players create a backdrop for a puppet show. They enter a paint program that lets them mix and create colors and paint any number of backdrops. The backdrop they choose appears for the rest of the game, just the way they painted it.

Byron Preiss's Ultimate Haunted House—a game designed to appeal to both boys and girls—also involves interactivity. Each player gets a bag of items—gruesome things like severed hands and piles of guts—to help them find 13 missing keys in a house inhabited by quirky characters and monsters. Players drag their gory items to different places, and each item sparks a different—and appropriate— reaction, depending on what it is and where you drag it. The interactivity keeps the player intrigued throughout the game, and "winning" is almost secondary.

No Dead Ends. Regardless of gender, players find paths that lead nowhere in a game a real drag. They don't like to be told that they did it wrong and have to start over. In Sanctuary Woods' Hawaii High, Mystery of the Tiki, a game designed for girls, this never happens. If players make a wrong move, they go back to the "story map," a kind of central directory where they can click on one of several icons representing various scenes in the game and return to that point in the story to start again.

If players must "lose" the game, several choices or actions that the player takes should lead to that end—not one. “That way, you don’t feel like you’ve been clobbered by the game,” says Byron Preiss’s Lanza. “You can see why you lost. You took this risk and insulted that cop and stole this evidence and did all this stuff that you really shouldn't do, and it kind of makes sense in the end. Then, you play it over again and things happen differently because you’re acting differently and you get to a different ending.”

The Girl Game Formula

These gender-neutral / girl-appeal elements are popping up in many games for children under 12. They include female characters girls are familiar with, no violence, a story with an altruistic goal, no time limits, and creative play. EA’s new game Madeline, based on the beloved storybooks by Ludwig Bemelmans, is a story adventure featuring a gutsy French schoolgirl. Sega’s Crystal’s Pony tale, a game designed exclusively for girls by Sega’s all women development team, involves players in a story about a female pony and her quest to save her friends from a wicked witch. And Big Top Production’s Hello Kitty Big Fun Deluxe is a learning toy starring Sanrio’s cartoon cat, a popular and familiar character with little girls.

While the developers of these games—who are mostly women—hope for high sales figures, they intend to secure a place for women in the high-tech world by designing games to make girls comfortable with computers at an early age.

Sega’s girls’ task force is developing games exclusively for girls under 12. “One of our objectives is to equalize the opportunity for girls as well as boys in high technology,” explains Sega’s Diane Fornasier. “We found that by the time a child is about 12 years old, their role models and activities are quite well established. If they have not interfaced with computers by that age, they will be less likely to be comfortable around them, and less likely to go into future careers in science and technology.”

The Teenage Void

While there is a lot of talk in the industry about making games appeal to girls, teenage and adult women are still shopping in a void. Some game developers see a huge potential market here—especially in teenagers. “The 13 to 17 year old spends more money than anyone. They spend it on all forms of entertainment and sports and literature and toys,” says Laura Groppe, an independent game developer in Houston Texas. Last year, Groppe and her partner started their own company, Girl Games, to design games specifically for girls and women.

“The timing is so right. In literature, female authors are out there like crazy.—and they’re appealing to boys as well as girls. The music industry has never seen so many women-led bands, and they’re appealing to everybody. I mean, we know what girls like, it’s just a matter of pushing the technology to that level.” Groppe plans to enter the market with an educational game and an online multiplayer game that will hook up women gamers across the country.

Solange Van Der Moer, a marketing consultant for Sausalito, Calif. based

Infinity Marketing Group also thinks it's time for women's games. Unimpressed with the attempts of some companies to appeal to girls and women and concerned about the small number of teenage women enrolled in computer courses across the country, Van Der Moer started a non-profit game venture called Womensware that will publish games designed by teenage women, for teenage women.

Van Der Moer and her associates found several teenagers between the ages of 14 and 19—women with no computer skills whatsoever—and paired them up with some of the industry's hottest developers who tutor them in programming and game design. All communication is done via Internet, so the women can become comfortable with online services as well as computer programming.

Womensware hasn't released any games yet, because the designers work at their own pace and must juggle school work with game development. Van Der Moer provides the women with equipment, a salary, and educational scholarships in computer education. She is very protective of her designers. She won't reveal how she found them, who they are, who's working with them, or any of the games they're working on—for their own personal privacy and perhaps to protect her interests.

While the whole thing might smack of teen labor exploitation, Van Der Moer maintains that Womensware is nonprofit and is not a game company. "To be frank, if they never develop a game, I don't care. The point behind Womensware is the transference of skills and building confidence; to say, 'No, this is not a boys game, you too can play, you too can be just as good, if not better at this.' "

Don't Call It a Girl's Game

But other developers who have tried to design games for women have come up against a number of challenges. For one, publishers are reluctant to market a girl's or women's game. The market doesn't have a proven track record and a tried-and-true marketing path hasn't been established. Publishers prefer to sink the quarter of a million dollars it costs to develop a game into something they know works.

Independent game designer Danielle Bunten knows this as well as anyone. She has been trying to pitch ideas for women's games for some time with no luck. Bunten has been a game developer for 10 years, and she has a unique perspective on gender-specific game design. Three years ago Bunten changed her gender from male to female. She designed three successful war games—Modem Wars, Command HQ, and Global Conquest—as well as the nonviolent hit, Mule, as male game designer Dan Bunten.

Bunten says she never pitches a game she thinks will appeal to women as a "women's game." "That's the kiss of death as far as game publishers are concerned," she laughs. She instead uses the term "family game," which is gender inclusive and easier for publishers to swallow.

Still, even when developing such a family game, Bunten says it's difficult for publishers to break out of their fast-action formula. Last year, Bunten was working with 3DO on an updated version of her game Mule, a popular nonviolent game published by Electronic Arts in 1983. The game, which involves four players who land on a planet and work together to survive using robotic mules, was a proven hit with both men and women. It includes many elements that appeal to both genders: there's no time limit, it's collaborative, and players take turns, which allows socializing during game play.

But 3DO felt something was missing. They wanted intensity, and they asked Bunten if she would remove the turn-taking element in the game and place all players on the field at once. Reluctantly, Bunten agreed. "But as soon as I added the simultaneity, it instantly put in their head, 'Why can't we shoot at each other?' And I said, 'No, no guns.' And they said, 'What about bombs? Can we drop a bomb in front of you? It won't hurt you—it will be a cartoon thing, it will just slow you down.' And I said, 'You don't get it, it's changing the whole notion of how this thing works!' "

Their differences unresolvable, Bunten stopped working on the game just before entering the beta stage and left 3DO for home in Little Rock, Arkansas. Bunten feels that there might be more opportunities for alternative games in the new platforms, such as 3DO and Sony, but this hasn't been the case for her. "Here's one (3DO) that's staking its future on the idea of a new generation of hardware and therefore, you'd assume a new generation of software, but they said, no, our market is still 18 to 35, males. We need something with action, something with intensity. Chrome and sizzle. Ugh."

Juicy Issues

Annie Fox and Laurie Bauman are two other established game designers who have come up against resistance when developing games for women that push the genre envelope. The two design partners wrote the successful children's Putt Putt game series for Humongous Entertainment and wrote Counting on Frank and Madeline for Electronic Arts.

They also developed a prototype for a large publisher specifically for teenage women. The game was an interactive advice game, akin to Dear Abby and Ask Beth, featuring five teenagers—two male and three female—who ask the player for advice on a number of issues teens deal with today, such as interracial dating, drugs, and sex. The player would recommend various actions for the characters to take, and witness the outcome.

The game company that commissioned the prototype decided not to proceed with the game for a couple of reasons. Fox says that each problem had an almost endless number of options and outcomes, and the publisher felt the product would involve too much branching and become unrealistically large and complex.

But the issues presented in the game were complex, too. "As a game publisher, you have to decide whether or not you want to get into advocating a certain sense of morality vs. another sense of morality," says Fox. "And with this age group, do you talk about sex and drugs or do you whitewash it?"

Whitewashing was what the company preferred. Their research showed that younger girls like to peek ahead at what it might be like to be adults and often read literature meant for older readers. "They were thinking that 15 or 16 year olds were not going to play this game because they're halfway through it. It's like, who reads *Seventeen* magazine? Not 17-year-olds; 10-year-olds."

Fox and Bauman were not comfortable presenting these issues to a younger audience, but they also weren't comfortable whitewashing problems important to teens. There were so many potentially thorny issues that the publisher decided not to move forward with the project. "It doesn't mean we've given up on it," says Fox. "It's on the back burner."

Bombs Are Easy to Program

Danielle Bunten also wants to address important social issues in her designs. She'd like to create a game to help girls and women deal with sexual harassment. Bunten has yet to pitch this idea to a publisher because she admits it might be too alternative for the corporate game publishing world—and it would involve elements that have never been tackled in game programming and design—things like subtlety, responsibility, relationships, emotions, and negotiation. How do you transfer stuff like that into C++?

Bunten suspects the difficulty of creating the elements of a new genre is one reason one hasn't emerged. "The things that boys care about are so much easier to represent in a visual world. They are tangible things—anything that moves and moves fast, like projectiles. You can put a picture of them on the screen, and you can make them do the types of behaviors players expect them to do," she says.

"But if you want to create an adventure game where you have characters that you interact with and negotiate with—you'd have to invent a new level of AI for these characters to behave like a reasonable approximation of a human, and the heavy duty scientific types have yet to come anywhere close to creating artificial personality inside a machine. The best they can do is kind of digitize elements of people's superficial world, like voice maybe, pictures maybe. But they can't do what motivates someone to say "yes" or "no" to a question, and that's necessary if you want to interact emotionally with people. And that's what the rest of us care about."

Where's the Pink and Lace?

If designers can get past the hurdle of reluctant publishers and the limits of technology, they will often meet opposition from distributors and retailers. This group is often not receptive to the idea of a gender-specific game or clueless as to how to display a product that might not fit under the defined Adventure-Sports-Strategy subject matter slots in a typical Egghead store.

San Francisco-based Big Top Productions ran into this resistance with its Hello Kitty game. "When we first came out with this product, we were marketing it as a girl's product, and we met a lot of resistance in the retail channels for that," admits Big Top cofounder Lisa Van Cleef. "We had basically narrowed the retailer sellability by being that specific with it. I think it was a brave move, particularly by a young company, to specifically try to address this issue. But we were slapped down."

"They looked at our product and said, 'Where's the lace, where's the pink, where are the obvious signs of girlness,' " says Van Cleef's partner, Jim Myrick.

Gender stereotypes are something developers face at the design stage, too. Although Sanctuary Woods' game Hawaii High Mystery of the Tiki was lauded for being one of the first games targeted toward older girls, and while it features two young women who independently solve a mystery, tackle tough decisions on their own, and have professional career women as moms and role models, it was also criticized by some developers for featuring bikini clad, Barbie doll-esque characters and a segment where players help a character pick out her wardrobe.

Sega's Crystal's Pony Tale was also criticized for featuring too much pink in its graphics. Diane Fornasier says they chose pink because their research came back saying that girls prefer pastels—and pink in particular. "It's a tough line to walk because we find there are certain things girls naturally gravitate toward whether its based on biological or gender differences or social differences that are already innate to them by [a certain age]. It's been difficult because we want to make the games appeal to girls but at the same time, we don't want to be over stereotypic." Fornasier says that they've toned down the pink in the interface and packaging of the game's next release, changing it to a more macha magenta.

Is This Just a Stage?

Despite all the challenges, there are a few brave designers out there who are confident that all this resistance is just a stage in a growing industry. Laura Groppe of Girl Games says most publishers she has talked to are responsive to new market opportunities, but they are more comfortable outsourcing work to independent companies like hers than they are to the idea of launching their own women's games divisions.

Although the large, well-established game publishers are cautious about marketing games to women, the few that are trying, such as Sega, will be the most powerful of the pioneers. "We are working with the retailers to put the games in and give them the opportunity to sell," explains Fornasier. "And we are working to make the communication—the message—something that's appealing to both girls and boys and parents, and have that help drive the sales for the female products as well as the boys products." Sega recently began testing what might be the first TV commercial for video games targeted toward girls.

Still, reaching girls and women gamers is only part of it. A new game genre isn't necessarily a gender issue as much as it is an issue of creativity and ideas—of moving this medium into a new, wider direction that includes and appeals to all kinds of game players. It's really just a matter of being gutsy enough to break out of the chrome-and-sizzle formula. Myst is just the beginning. Game developers, don't give up. The nontraditional game players are out there, and we're waiting. ■

Barbara Hanscome is the managing editor of Software Development *magazine and production editor with* Game Developer. *Don't be fooled—she likes Doom just as much as the next person and hates pink. You can contact her at 73611.633@compuserve.com or through* Game Developer *magazine.*

Back

Tie Fighter, Part 1

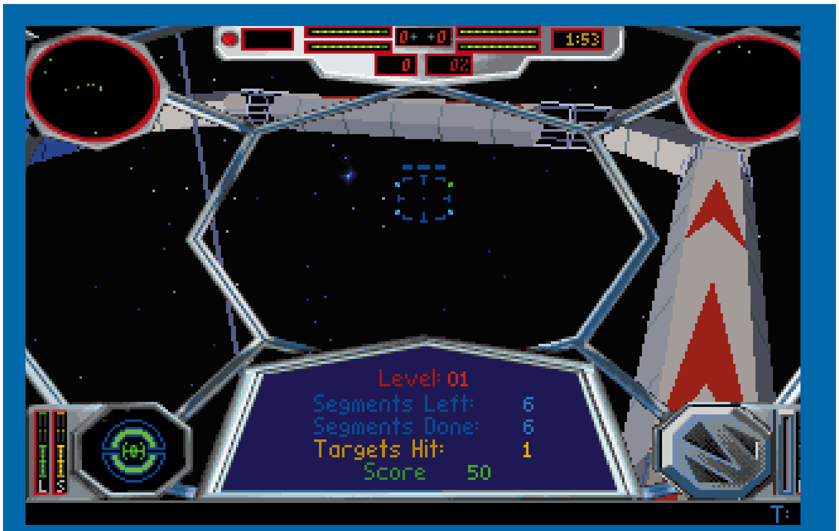
by Wayne Sikes

Peer into the depths of the TIE Fighter game engine from LucasArts Entertainment and discover a world of complex data files, file storing, and file-naming conventions.

On the chopping block this month is TIE Fighter by LucasArts Entertainment Company. TIE Fighter is the second game in LucasArts' Star Wars series, X-Wing is the first. The TIE Fighter game engine is an improved version of the X-Wing engine. The most noticeable improvements were made in the graphics rendering, artificial intelligence, and cockpit data display areas.

The TIE Fighter engine and its associated data structures are somewhat complex; therefore, this review will span two editions of the Chopping Block. In this issue, I will broadly summarize the executable and data files and focus on the pilot data file. In the next issue, I will delve into mission construction. I used TIE Fighter version V1.0 (06/15/94) for this review.

TIE Fighter initially requires about 14MB of hard disk space. A minimum of 572k conventional RAM and 900k of expanded memory are required to run the game. Ideally, the game wants 2MB of expanded memory. During installation the \CP, \MISSION, and \RESOURCE subdirectories are created under the primary \TIE directory. The \CP subdirectory contains most of the data files for vehicles the player is allowed to fly. The \MISSION directory is loaded with battle and historic mission data, and \RESOURCE contains the "generic" pieces of the game, including battle summary information; music, sound effects, and speech data; menu screen graphics; the game logo and credits; and registration (copy protection) data. The primary game subdirec-


Level: 01
Segments Left: 6
Segments Done: 6
Targets Hit: 1
Score 50

A defender training mission in TIE Fighter.

Listing 1. Data Storage Files

FILE NAME SUFFIX	DATA TYPE
TIE	All Historical and Battle mission files.
LFD	The most common data storage format for various types of data.
PNL	Appears to contain raw graphics data.
INT	Contains data for vehicles that the player can fly. Primarily tabular in format.
TFR	Pilot data file.

Listing 2. LFD Storage File Structures

```
struct  LFDRECORDHEADER
   {
   char   RecordName[12];     // Record name string
   long   RecordSize;         // Size of Record data
   };

struct LFDRESOURCETAG
   {
   char   ResourceTag[12];    // "RMAPresource" string
   long   FirstRecordOffset;  // Offset of the first Record
   };
```

Listing 3. Spacecraft Name References

SHIP NAME	LETTER REFERENCE	NUMERICAL REFERENCE
TIE Fighter (T/F)	F	1
TIE Interceptor (T/I)	I	2
TIE Bomber (T/B)	B	3
TIE Advanced (T/A)	A	4
Assault Gunboat (GUN)	G	5
TIE Defender (T/D) (also known as the TIE Deluxe)	D	6

tory, \TIE, contains the game executable files, user configuration information, system setup help routines, display fonts, and various sound card drivers.

Tie Fighter Executables

TIE Fighter was written using the Microsoft C development system. The executable code consists of three files: FLIGHT.OVL, FRONT. OVL, and TIE.EXE. The game is started by running TIE.EXE, which subsequently loads the FRONT.OVL and FLIGHT. OVL overlay routines when required. The various game functions are logically organized into these three files.

The TIE.EXE routine performs game start up. An initial part of the start up includes memory allocation and data validity checks. Once memory has been verified, the iMUSE (LucasArts' proprietary Interactive Music and Sound System) engine is started. Another initialization function involves setting up the flight combat filming system. While examining the TIE.EXE code, I noticed the unusual character string "heidirobinyali" buried in the data. Purusal of the *Starfighter Pilot Manual's* list of game credits showed that this string was the first names of three of the "Invaluable Support" personnel. It is possible that this data string was used by programmers as a trigger for debug or developmental mode operation. Although TIE.EXE is 368365 bytes in length, only about 59943 bytes are used for code and data. The remaining 308422 bytes have a 0 value and are apparently used when the Microsoft C run-time system loads the FRONT.OVL or FLIGHT.OVL overlay routines.

The FRONT.OVL module provides the front end for the game and provides the overall, nonflight gaming environment. This environment includes most of the main menu Concourse functions, cut scenes and other battle-related animations, award functions, funeral scenes, Tech Room and Film Room operations, and the registration and copy protection modules. To avoid violating LucasArts' copyrights, I will not tell you how to override the game's copy protection. I can tell you that the copy protection consists of standard C, null-terminated character strings that begin at file offset 3E42A (hex) and the last string ends at offset 3E5F1 (hex).

The FLIGHT.OVL overlay module contains most of the flight data. This file appears to contain the artificial intelligence used during combat and other space flight sequences. There are a number of possibilities for most combat situations. For example, a cursory scan of the "intelligence" given to Rebel vehicles found references to over 70 presumed different actions or activities. Data structures that define the general layout (weapons, shields, hull strengths, etc.) of all game vehicles are located in FLIGHT.OVL.

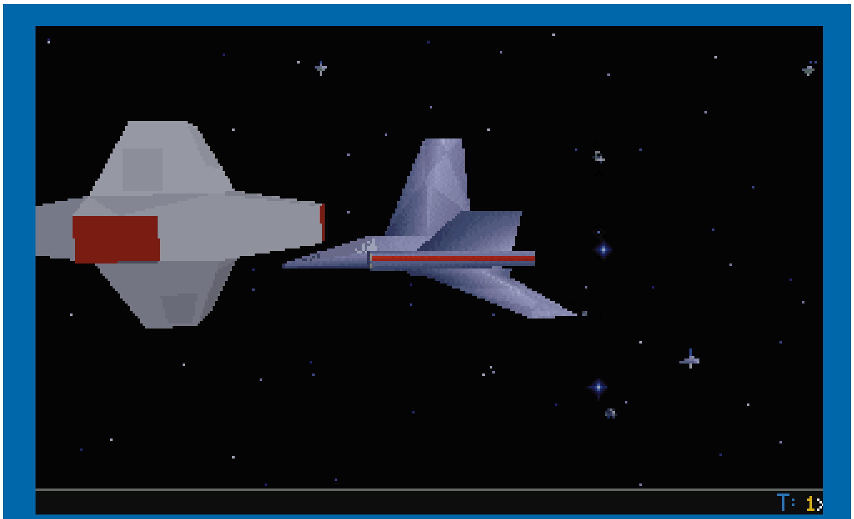
How Is the Data Stored?

TIE Fighter data is scattered among numerous files, and I observed several storage formats. Listing 1 gives a summary of several data storage file types. Most mission data is stored in .TIE files and the bulk of the general data is stored in .LFD files. (I speculate that the LFD suffix is a mnemonic for “Lucas File Data,” and I will refer to the files suffixed with LFD as “LFD files.”) Pilot data is stored in files suffixed with TFR.

The LFD file format allows for the storage of single or multiple Records. I observed two flavors of LFD files, but the Record data is stored basically in the same manner between the two. I will refer to these two types of files as Type I and Type II. The difference between the two LFD file types is that Type II has a tabular header at the top of the file, which indexes all of the Records in the file.

In both LFD file types, each Record consists of a 16-byte header followed by the Record data. The 16-byte header is composed of a 12-byte Record name followed by the size of the Record data expressed as a 4-byte (32-bit long) value. The Listing 2 structure, `LFDRECORDHEADER`, gives an example Record header. The actual size of the entire Record would be calculated by adding 10 (hex) to the `RecordSize` value.

The Type II tabular header consists of a 16-byte “Resource Tag” followed by 16-byte headers for all Records in the file. These 16-byte headers are duplicates of the Record headers previously discussed. The individual Records immediately follow the tabular header. The Resource Tag consists of a 12-byte character string, “`RMAPresource`”, followed by the offset of the first Record expressed as a 4-byte value. The `LFDRESOURCETAG` structure in Listing 2 gives an example Resource Tag. (The actual offset of the first Record following the tabular header is calculated by adding 10 (hex) to the `FirstRecordOffset` value.)



An assault gunboat flies by a buoy, one of several space objects in TIE Fighter.

Mission File Naming Conventions

As previously mentioned, the mission files have a TIE suffix. Examination of the \TIE\MISSION subdirectory reveals many similar file names. Battle and Historical mission files are named using a standard convention. The Battle mission files are named using the format shown in Figure 1.

Using the diagram in Figure 1, the file named B7M3AW.TIE would reference Battle 7 Mission 3, where the player is flying a TIE Advanced vehicle in the Wingman position. Listing 3 shows the spacecraft name references. Historical mission file names are slightly different, as shown in Figure 2.

Using the Historical mission file naming diagram, a file named HG3M.TIE would reference Assault Gunboat Historical Mission 3, where the player flies an Assault Gunboat and is the

Figure 1. Battle Mission File Naming

```
B  [1-7]  M  [1-6]  V  [MW] . TIE
|    |    |    |    |   |  |W = player is the Wingman
|    |    |    |    |   | M = player is the Flight Leader
|    |    |    |    | Vehicle letter (see Listing 3)
|    |    |    | Mission Number
|    |    | M for Mission
|    | Battle Number
| B for Battle
```

Figure 2. Historical Mission File Naming

```
H  V  [1-4]  [MW] . TIE
|  |    |    |  | W = player is the Wingman
|  |    |    | M = player is the Flight Leader
|  |    | Mission Number
|  | Vehicle letter (see Listing 3)
| H for Historical
```

Listing 4. Mission Files

```
BATTLE MISSIONS

BATTLE      FILE NAME (*.TIE)

Battle 1    B1M1FM B1M2FM B1M3BM B1M4IM B1M5GM B1M6GM
Battle 2    B2M1FW B2M2BW B2M3IW B2M4GW B2M5FW
Battle 3    B3M1BM B3M2BM B3M3FM B3M4GM B3M5BM B3M6GM
Battle 4    B4M1FM B4M2BM B4M3BM B4M4IM B4M5GM
Battle 5    B5M1IW B5M2GW B5M3GW B5M4AW B5M5GW
Battle 6    B6M1AW B6M2AW B6M3GW B6M4GW
Battle 7    B7M1AM B7M2AM B7M3AW B7M4DW B7M5DM


HISTORICAL MISSIONS

VEHICLE          FILE NAME (*.TIE)

TIE Advanced     HA1W   HA2W   HA3M   HA4M
TIE Bomber       HB1W   HB2W   HB3M   HB4M
TIE Defender     HD1W   HD2W   HD3M   HD4M
TIE Fighter      HF1W   HF2W   HF3M   HF4M
Assault Gunboat  HG1W   HG2W   HG3M   HG4M
TIE Interceptor  HI1W   HI2W   HI3M   HI4M
```

Flight Leader for the mission. Listing 4 gives the TIE Fighter mission file names.

Does Your Pilot Need Help?

If you are like me, your computer pilot needs help every now and then. Your pilot may die unexpectedly (the "I didn't see that one coming" scenario) and games such as TIE Fighter will penalize the player when his or her character is revived. In this section I discuss how some of the data in the pilot file is organized; you may find the information useful for pilot survival.

As mentioned previously, TIE Fighter stores pilot data in files suffixed with TFR. Pilot files are 3,856 bytes in length, but the size is somewhat misleading. During game play you can press the "escape" key to bring up the Personal Datapad, which can then backup your pilot file. The backup copy of your pilot data is stored in the same file as your primary pilot data. The first 1,928 bytes of the file are your active pilot data, and the last 1,928 bytes are backup data. When manually editing pilot files, avoid editing the backup data.

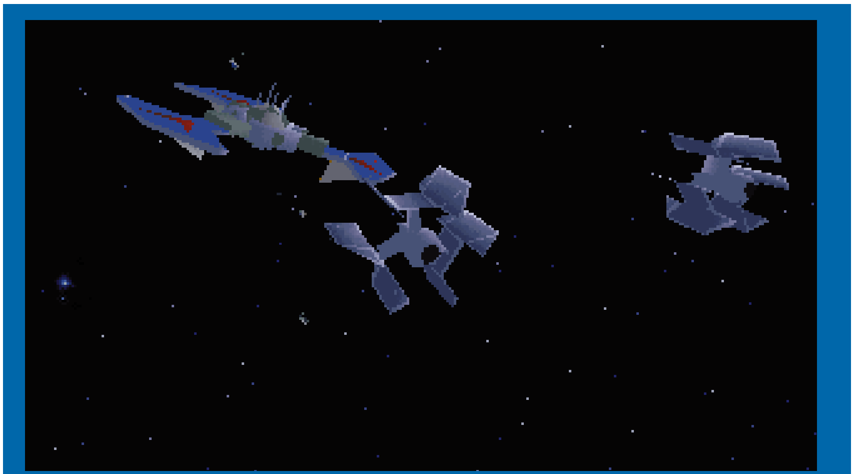
Listing 5 gives a summary of several important data parameters and where they are located in your pilot file. (The FILE OFFSET references in the listing assume the first byte in the file is "byte 0".) The pilot file has open or add-on slots for future game expansion. There are slots for eight Historical missions for each vehicle you can fly, but the current game only has four Historical missions per vehicle. There are eight available mission slots for each Battle, but all Battles currently have less than eight missions.

Feel free to experiment with your pilot file (after backing it up, of course). It is very easy to change your skill level, scores, or reduce the number of vehicles you have lost. Personally, I usually edit the mission completion flags to mark a difficult mission as completed when my "computer pilot" gets a little too frustrated.

Until We Meet Again at the Imperial Starbase

As with many of LucasArts' gaming products, TIE Fighter is well written and executed. The improved graphic engine and artificial intelligence make the game a true leader in the space combat and simulator genre of games. The clean layout of the mission and pilot files makes tailoring the game much easier and lots of fun.

In the next Chopping Block column, we will dig into the TIE Fighter mission data structures. If you are interested in editing mission files or creating new missions, I would strongly urge you to pick up a copy of *The TIE Fighter Official Strategy Guide*. The mission statistics tables located in the back of the book were extracted directly from the mission file data. Once you understand how the mission



TIE defenders approaching a space platform.

Getting ready to dock, cockpit view.

More TIE Fighter action.

data is stored (by reading the next Chopping Block column), the *Strategy Guide* will make mission file editing much smoother. ■

Wayne Sikes has been a computer hardware and software engineer for the last 10 years. He has an extensive background in C, C++, and assembly language programming. He also has several years experience as a computer systems intelligence analyst, where he specialized in deciphering and disassembling computer code while working on classified government projects. He has written numerous computer gaming help utilities. You can reach him via e-mail at 70733.1562@compuserve.com or through Game Developer.

Listing 5. Pilot File Data

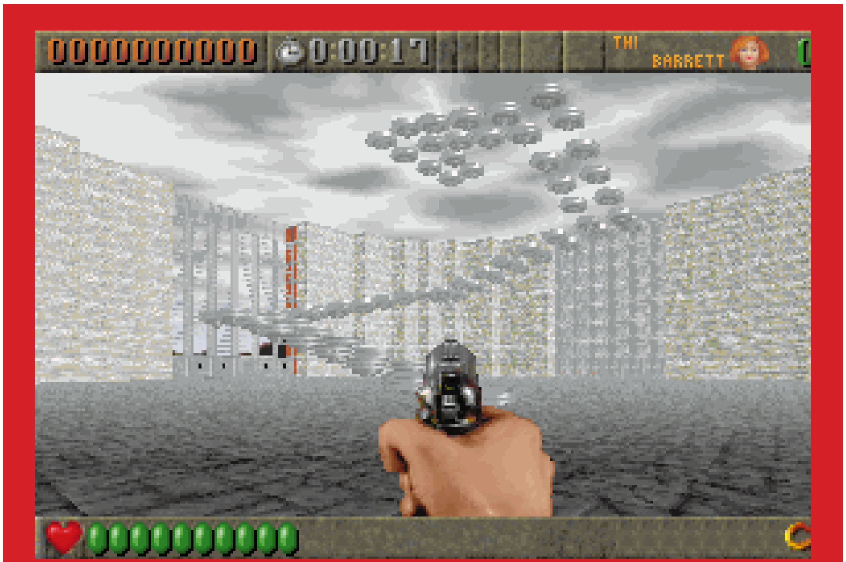
```
FILE OFFSET  DATA     DESCRIPTION
(DECIMAL)    TYPE*

1            byte     Duty Status.  0=Alive, 1=Captured, 2=Killed
2            byte     Rank.  0=Cadet -> 5=General
3            byte     Difficulty Level.  0=easy -> 2=hard
4            long     Score.
8            word     Skill Level.  0=Novice -> 65535=Super Ace
10           byte     Secret Order Ranking.  0=None -> 6=Emporer´s Hand
29-34        byte     Next Training Level.  off 29=T/F -> off 34=T/D **
42-62        long     Training Scores.  off 42=T/F -> 62=T/D **
90-95        byte     Total Training Levels Completed. off 90=T/F -> 95=T/D **
136-164      long     T/F Historical Scores. 136=Mission1 -> 164=Mission8 ***
168-196      long     T/I Historical Scores. 168=Mission1 -> 196=Mission8 ***
200-228      long     T/B Historical Scores. 200=Mission1 -> 228=Mission8 ***
232-260      long     T/A Historical Scores. 232=Mission1 -> 260=Mission8 ***
264-292      long     GUN Historical Scores. 264=Mission1 -> 292=Mission8 ***
296-324      long     T/D Historical Scores. 296=Mission1 -> 324=Mission8 ***
520-527      byte     T/F Historical Completion Flags. 0=Not Done, 1=Done ***
528-535      byte     T/I Historical Completion Flags. 0=Not Done, 1=Done ***
536-543      byte     T/B Historical Completion Flags. 0=Not Done, 1=Done ***
544-551      byte     T/A Historical Completion Flags. 0=Not Done, 1=Done ***
552-559      byte     GUN Historical Completion Flags. 0=Not Done, 1=Done ***
560-567      byte     T/D Historical Completion Flags. 0=Not Done, 1=Done ***
617          byte     Battle 1 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
618          byte     Battle 2 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
619          byte     Battle 3 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
620          byte     Battle 4 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
621          byte     Battle 5 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
622          byte     Battle 6 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
623          byte     Battle 7 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
637          byte     Battle 1 Last Mission Completed. 0=None, 1=Mission1...
638          byte     Battle 2 Last Mission Completed. 0=None, 1=Mission1...
639          byte     Battle 3 Last Mission Completed. 0=None, 1=Mission1...
640          byte     Battle 4 Last Mission Completed. 0=None, 1=Mission1...
641          byte     Battle 5 Last Mission Completed. 0=None, 1=Mission1...
642          byte     Battle 6 Last Mission Completed. 0=None, 1=Mission1...
643          byte     Battle 7 Last Mission Completed. 0=None, 1=Mission1...
986-1014     long     Battle 1 Scores. off 986=Mission1 => 1014=Mission8 ****
1018-1046    long     Battle 2 Scores. off 1018=Mission1 => 1046=Mission8 ****
1050-1078    long     Battle 3 Scores. off 1050=Mission1 => 1078=Mission8 ****
1082-1110    long     Battle 4 Scores. off 1082=Mission1 => 1110=Mission8 ****
1114-1142    long     Battle 5 Scores. off 1114=Mission1 => 1142=Mission8 ****
1146-1174    long     Battle 6 Scores. off 1146=Mission1 => 1174=Mission8 ****
1178-1206    long     Battle 7 Scores. off 1178=Mission1 => 1206=Mission8 ****
1626         word     Total Kills.
1628         word     Total Captures.
1926         word     Number of Craft Lost.

*      "byte" references an unsigned character.
       "word" is a 16-bit unsigned value.
       "long" is a 32-bit signed value.
**     Vehicles are ordered as in Listing 3.
***    There are currently four historical missions for each flyable craft.
       The pilot file has storage slots for eight historical missions per craft.
****   The pilot file has provisions for eight missions per battle.
```

Back

The Sultans of Shareware Go Retail

by Alexander Antoniades

In an attempt to become more than "those other shareware guys from Dallas," Apogee Software makes its move to dominate the realm of 3D gaming.



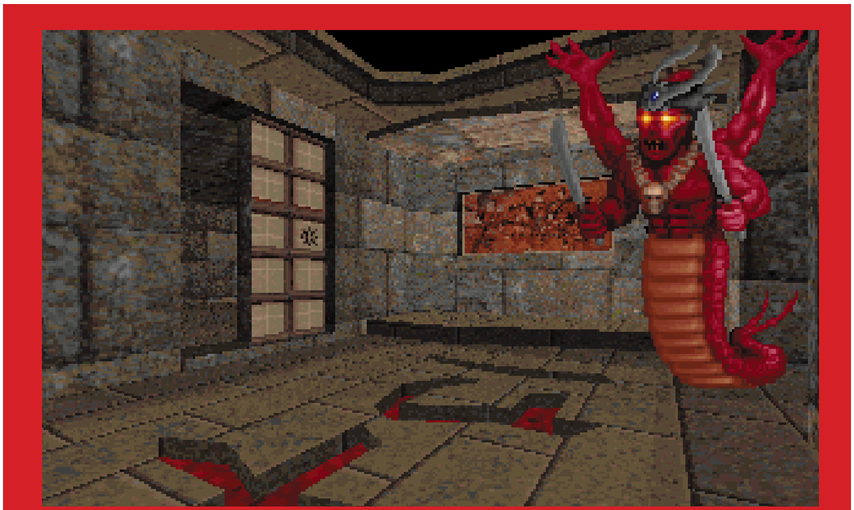
Rise of the Triad is Apogee's first post-Doom, three-dimensional game release. The game engine isn't sophisticated enough to be released under Apogee's new 3D Realms game label, but it is a good example of the kind of three-dimensional game released under Apogee's name.

Only in America, the legend goes, does the little guy have a chance to hit the big time. This story is true for a couple of childhood friends from Dallas, Texas, whose company, Apogee Software, has come to dominate the shareware game industry and is now looking to become a major player in the retail channel.

It all started in 1987, when Scott Miller was working as a computer consultant and wrote one of the early shareware games for the PC, Kingdom of Kroz. Although the game was a simple ASCII text adventure, it became so successful that Scott quit his job and formed Apogee Software. (The name Apogee came from a band that he had been with in 1982 and fit in with his interest in astronomy.) He repeatedly tried to convince his friend, George Brousard, author of the shareware game Pharaoh's Tomb, to join him, but George kept his day job until 1991, when he eventually joined Scott as partner in Apogee.

Apogee's mission was simple: find cool games and distribute them. Scott and George scoured the BBSs looking for cool games that just needed that finishing touch to become hits. Once they found a cool game, they contacted the authors, signed them up, and handled the distribution and fee collecting.

Todd Raplogle was the first person they signed up. His game, Caves of Thor, was a prime example of the kind of cutting-edge game Apogee was look-



Shadow Warrior is one of four games that will be released under Apogee's 3D Realms title in 1995. The games, each developed by a different design team, will use Apogee's Build engine.

ing for. Eventually, Todd left his home in Santa Cruz, Calif., for Dallas, where he worked with Apogee on its first really big hit, Duke Nukem.

One reason behind Apogee's success is Scott's "trilogy approach" to game marketing and distribution. This method consists of making the first third of the game, which contains a subset of the features, unconditionally free. To get the remaining two thirds, players must register for the game. He developed this style by accident after he regained the rights to three games he had written for Soft Disk. To test the software market, he released the first game as freeware and charged for the other two. This approach helped Apogee become *the* shareware game company.

But it wasn't just creative marketing that made Apogee successful—it was also the ability to spot great talent, such as Id software. Before the makers of Doom were the masters of all they surveyed in the gaming community, they worked for Apogee. Scott wooed them away from the company they were working for, Soft Disk, by sending them fan letters under fake names. Each letter ended with the message "contact me" and Scott's phone number. (See the article "Monsters from the Id: The Making of Doom," Premier issue, 1994.)

At the peak of their relationship, Id accounted for about 20% of Apogee's total revenues. Id cranked out hit after hit, first with the Commander Keen games, a series of side scrollers in the same vein of Sigeru Miyamoto's Super Mario games (see the article "Miyamoto's World" by David Sheff, June 1994). They next created the then state-of-the-art Wolfenstein 3-D, one of the first faux three-dimensional games to capitalize on texture mapping and fast bitmap manipulation.

Id's success gave them enough name recognition and money to distribute its own games. So, after a very successful three years, Id and Apogee split up to seek their own fates, but they continued to work on a few projects together.

The first game, BioMenace by Jim Norwood, was the only finished product to come from a Commander Keen cloning workshop that Id taught to other Apogee developers. A second project was Blake Stone and the Aliens of Gold, made by JAM productions, which used the Wolfenstein 3-D game engine.

Their last project together was Wolfenstein II. During this time, Doom was becoming a huge hit, and the Id developers broke away from the Wolfenstein II project, saying that they were too busy to continue with it. Apogee was having second thoughts about the project, anyway. Both companies felt it was heading in the wrong direction.

Apogee wasn't left completely in the lurch when Id parted. One of Id's founding members, Tom Hall, who had left Id due to creative differences at the beginning of Doom, moved over to Apogee and became the leader of its first in-house development team.

Goodbye Wolfenstein II

Tom was heading up the Apogee side of the Wolfenstein II project, but he wasn't happy with it. His main problem was that the iconography of the game was too confining. He wanted to make a game that had a wide variety of characters and creatures, so when Id was too busy to continue working on the project, Tom took the opportunity to start from scratch.

A new game, Rise of the Triad, started out with legacy artwork from Wolfenstein II. Because the artwork had taken six months to complete, Tom didn't want it to go to waste. In a moment of Roger Corman B-movie inspiration, Tom dreamed up a storyline in which a super secret U.N. SWAT team stumbles upon a terrorist plot to destroy Los Angeles. The terrorist's cover is an old movie studio that looks like a Nazi fortress.

To finalize the divorce from Id, Tom plugged the data into a new game engine. The engine that Wolfenstein II was designed around was an enhanced version of the original Wolfenstein engine, which had texture-mapped floors and ceilings. Tom enlisted the aid of Mark Dochterman to build a new engine that would expand the capabilities of the game while using the current artwork. The new engine included support for multiple heights (such as three-story buildings), translucent walls (sheets of glass), and light sourcing. While the final engine wasn't quite up to par with Doom (walls had to be at right angles and the graphics tiles were bigger), it did have a couple of things that Doom didn't, such as bullet marks on the walls and support for more network players.

To take advantage of these new features, Tom incorporated some ideas he was originally going to put into Doom had he continued working on it. One concept was to have different characters, similar to Street Fighter II, who would have different appearances and characteristics. Another theme was environmental dangers such as spinning blades, crushing walls, and giant rolling balls to add another element of chance when there were no living enemies around.

Other touches show the depths of Tom's imagination. My favorite of the power-ups is the dog mode, which switches the perspective to a lower level and places a dog snout where your weapons were. Other playability extenders include springboards that catapult the player tens of feet in the air, random actors (roughly the equivalent of wandering monsters in Dungeons and Dragons), and the ability to generate completely random levels.

Network Heckling

Another aspect that Apogee didn't want to overlook with Rise of the Triad was network play. The game can support up to 11 network players as well as a "remote ridicule" mode that can transmit the players voice through a sound card over a LAN to be played back on other players' machines.

Apogee considers projects of this size the minimum for future development. While Scott and George have gotten rich by releasing six to eight small games a year, with up to 22 projects going at one time, their goal is to become big-time developers working on four to six games a year.

Their first step is to use in-house development teams, similar to Tom Hall's nine-person crew, that will be able to flexibly build games in a reasonable time frame. After working with many small developers all over the U.S., Apogee has found that long-distance relationships generally haven't worked for them. The communication gap between what Apogee wanted and what the developer wanted often resulted in so many revisions that by the time some of Apogee's games got to market, their technology was too old to be competitive.

A new branding strategy is another plan Apogee has to become more competitive. Apogee is launching a different company this year that will only do three-dimensional games. The new company, called 3D Realms, will be a sister company separate from Apogee that will release games using the latest three-dimensional technology. Apogee wants to retain name recognition for making general action games.

The key component to this strategy is a new game engine called Build, developed by Ken Silverman, author of the shareware game Ken's Labyrinth. This engine is capable of rendering a 640-by-480 screen, and, according to Apogee, it matches or surpasses the Doom engine feature for feature. Four games are currently planned for release from 3D Realms in 1995 using this engine.

Although shareware is important to Apogee over the long run, one goal is to break into the retail market. Apogee will team up with FormGen, the distributors it used for Wolfenstein, for all of its currently planned ventures. It will release the shareware and retail versions of its products as close together as possible, unlike Id, which staggers its retail releases a great deal behind its shareware.

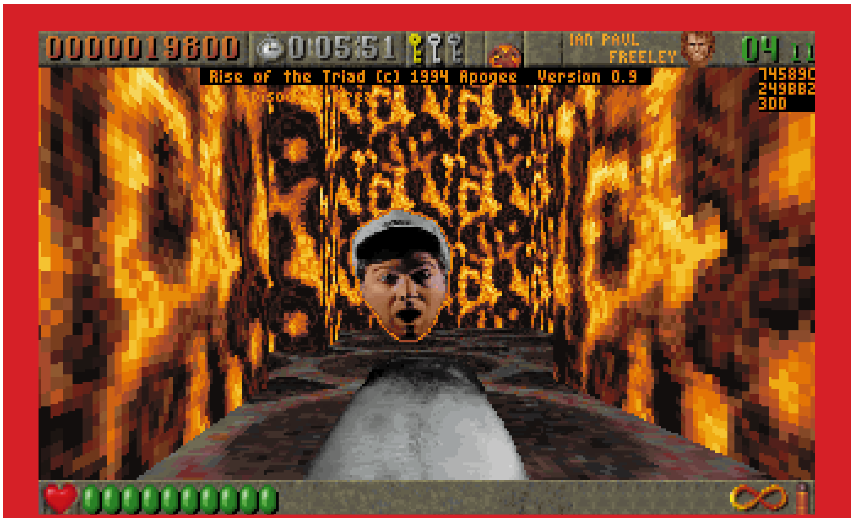
Apogee's shareware roots left the company particularly well positioned in the online market. Because of the distribution model that shareware uses (duplicate early and often), Scott and George had to establish a presence online from the very beginning. The early days involved a week of 20-hour days once a game was released to make sure that it got on the shareware community's 100 BBSs.

Today, after investing more than $200,000, Apogee has the Software Creations BBS, which features more than 100 lines and services 3,500 distribution points. Apogee also has its own section on America Online and will soon open an electronic software store on CompuServe. To manage these products, Apogee employs two people, whose sole job is to offer support online.

This infrastructure hasn't changed one thing, however: Scott and George still work at home. A short distance from their main offices where their 25 employees work, the principal partners in Apogee find that they still keep hours too irregular to be confined to an office routine. They believe that this nontraditional working style and hands-on management will keep them competitive as they alternate between taking on other shareware vendors and fighting for shelf space with the big boys.

Still looking for hot technology, they get 20 proposals a week from developers eager to become the next shareware millionaires. If Apogee can make the transition from shareware moguls to retail darlings, they will have made a new model for game companies to use and further validated shareware's impact on the game market. ■

Alexander Antoniades is Game Developer*'s editor at-large.*



It's creative touches such as Dog Mode and other surprises that Apogee hopes will make ROTT a hit in 1995. (The guy in the Gilligan Hat is Scott Miller of Apogee.)

Back

ARTIST'S VIEW

3D or Not 3D—Is That the Question?

First, let's get this straight: "Doom-style graphics" is not, technically speaking, the correct term to describe a game's look. It is more proper to say "Wolfenstein-style graphics." No, wait, that's not right either.

Many of the terms required for a discussion of game graphics have entered the vernacular with subtly but significantly altered meanings. Visual perception is central to the way in which most of us relate to the world, yet translating those perceptions into words or rendered images can prove to be less than intuitive.

Software may largely obviate the need for real technical drawing skill, but to make the best use of graphics possibilities and discuss them knowledgeably, it helps to understand the real building blocks of a scene. In this article, I'll cover various so-called projection methods used to create an illusion of space, and framing considerations that further define the look and feel of a game. Along the way, I'll try to clarify terms to help alleviate some of the misinformation already bogging down this topic.

The Look

Of course, a game's look is defined by a number of factors. Of principle importance is view, commonly but not quite accurately referred to as a game's perspective. View is the element most often invoked when a game is said to be like another, familiar game. Other factors may contribute greatly to achieving the overall look of a game and be of equal or greater importance to its success, but view is definitive.

As an example, in Halloween Harry, gameplay consists of running around toasting monsters and looking for power-ups. The plot sounds a lot like Doom, but you'd be unlikely to compare Apogee's four-way scroller to Id's first-person shoot-em-up. On the other hand, Quarantine, a first-person driving-and-shooting romp from Gametek, is routinely compared to Doom because of its similar visual style, despite significant differences in plot and gameplay.

View is readily understood on a visual level; it presents an illusion that the mind is easily able to interpret. However, like most illusions, the behind-the-scenes preparations require a more specialized knowledge. To understand the workings of a view, we must first understand dimensionality, projection method, and framing.

Dimensionality

Dimensionality refers, not surprisingly, to the dimensions represented in an image. A two-dimensional scene represents only height and width (and technically is not considered a view, though personally I feel that may be slicing the terminology a bit thinly). A scene rendered in three-dimensions additionally represents depth, thereby placing objects in relation to one another in a spatial context for a far more realistic appearance.

Two-dimensional images are common in many arcade-style games, such as two-way or four-way scrollers á là Mario Bros. or vertical elevation (bird's-eye) shooters. Though many two-dimensional games use simple shading effects to create the illusion of form, objects exist only on an X,Y axis; space does not enter the equation.

This is a less sophisticated, less convincing method for presenting a scene, yet for certain game types, it is preferable to a more realistic, three-dimensional depiction. If you made it to the secret Pac-Man level in Castle Wolfenstein, you know what I mean (Pac-Man never would have made it as a three-dimensional game). Suffice it to say that though it may be less of a visual feast, two dimensions have a well-established place as an electronic gaming format.

"3D" is now officially a buzzword, which means it can be dropped meaningfully by people who don't really know what they're talking about. Games like Doom and its predecessor, Castle Wolfenstein, are commonly described as being three-dimensional as though that tucks them in a neat little cubbyhole. Three-dimensional they are, but so are Origin's Ultima 8, LucasArt's Sam & Max Hit the Road, Infogrames' Alone in the Dark, and a great number of other games on the market today. More definitive than the tri-dimensionality of a game is the method by which those three dimensions are projected onto the viewing plane.

Projection

Isometric projection is one such method—and another term that tends to be misused. It really only refers to a specific view in which the sides of a rectilinear object are each at a 30-degree angle to the horizontal axis. The impression achieved is of looking down on the scene from a modest height. Ultima 8 uses isometric projection, as does Mystic Tower from Apogee.

Cabinet projection—often mistakenly referred to as isometric projection, to which it is similar—is another simple system for suggesting space. The main difference is that in cabinet projection the face of an object lies parallel to the horizontal plane while the sides are at a 45-degree angle to it. This results in a seemingly less elevated vantage point. A recent example of a game that makes use of cabinet projection would be Theme Park, by Bullfrog/Electronic Arts.

These two projection methods create an effective enough illusion of space when used to depict a scene of limited scope. They are not suited to presenting vistas, however, nor is the sense of depth created especially convincing. This is because in both systems, the scale of an object remains constant regardless of its relative position in space; a figure shown in the extreme foreground at the bottom of the screen appears the same size as a figure positioned in the farthest background at the upper limit of the screen. What's lacking is *perspective*.

Though, again, the term is misused frequently, perspective is itself a projection method. Properly known as central projection or scientific perspective, this is a more convincingly realistic system for creating the illusion of space. Distant figures are depicted as

David Sieks

Software may obviate the need for technical drawing skill, but to discuss graphics knowledgeably it helps to understand the real building blocks of a scene.

appearing smaller, and parallel lines, such as railroad tracks, converge at the horizon. This is more readily accepted by our minds as the way things are supposed to look.

I might add here that any number of arcade-type action/adventure games use backgrounds that are rendered in perspective (more or less), while the figures are really only depicted in two dimensions and move on an X,Y axis, which does not represent the third dimension of depth. This is not yet another projection method, just a hodgepodge. But, hey, it's just a game.

Vantage Point

Critical to an understanding of perspective is the fact that the viewer's eye-level lies along the horizon line, regardless of the height at which the viewer is stationed. This raises another issue, which is not a consideration with projection methods other than scientific perspective: vantage point, also known as point of sight or point of station. ("Point of view" also sounds right, but that term is widely used to describe a *specific* vantage point, which we'll get to shortly.)

Whereas with isometric or cabinet projection the apparent position of the viewer is dictated by the prescribed angles used in the rendering, a perspective view is infinitely flexible; the viewer's vantage point must be positioned relative to the scene. This positioning is no light consideration. Rather, it is one of a view's most important characteristics.

Which brings us once again to Doom. We've established that the game is three-dimensional; there is certainly an illusion of depth. Further, we know Doom is rendered in perspective; distant objects appear smaller, and parallel lines converge toward the horizon. Since it's in perspective, it has a vantage point: in this case, that vantage point is known as first-person or, in Hollywood, point of view (POV).

First-person perspective puts the viewer in the driver's seat, so to speak. In cinema, the POV shot purports to show the scene through one character's eyes. In gaming terms, the player is looking through the eyes of the avatar, the player's gaming persona.

The effect of the first-person view can be extremely immersive. It very closely approximates the way we see the world and so can really seem to put the player in the middle of the action. This works great for fluid, action-oriented games. In addition to the myriad offspring of Castle Wolfenstein, almost all flight sims are first-person.

First-person is not the answer for all game types, though. Some games, for one reason or another, rely on the player being able to see the avatar within the context of the scene. This calls for the player's vantage point to be somewhere other than inside the avatar's head.

It is helpful to borrow some terminology from the cinema to cover the general range of options: A straight-on view approximates eye-level. This is quite common in film, where it presents the image in a neutral fashion (high or low views are thought to impart subliminal meanings to a scene). The straight-on view is used in some games, but action can tend to appear crowded from this angle if there are many figures moving at once. One Must Fall from Epic Megagames is well suited to a straight-on view; action is well depicted, and with only two figures onscreen the view does not appear cluttered.

For the purposes of some electronic games, a high angle view is often used. This vantage point positions the player on a somewhat higher plane than ground level, so that the player appears to be looking down at the scene as though viewing it from a balcony or other elevated point. The result is a good overall view of the scene, showing the avatar in relation to its surroundings.

Perspective can also be projected from a low angle (below eye-level) or an oblique angle (canted at an angle to the horizontal plane, that is, crooked). These are not generally useful for game play, but can make a very effective view for transitional animations.

A further consideration is the shifting vantage point. With mobile framing, the view tracks or jumps to follow the action. This is used with some sports games when play moves from one end of the playing area to the other. Front Page Sports: Baseball '94 from Dynamix is one example.

Another use of a shifting vantage point is with what is known as cinematic perspective. As its name suggests, cinematic perspective mimics the frequent cuts and varied angles of motion pictures. It can have tremendous affect on the mood of a scene and is an asset to highly plotted games that have a story to convey. Action-intensive gameplay can be difficult in some views, though, which is a consideration to make in choosing angles to use. Origin's Bioforge makes use of cinematic perspective, and it is also used throughout the Alone in the Dark series of games.

Isometric, perspective, point-of-view. For me, forgetting the definitions I thought I already knew was the hardest part of understanding the technical aspects of view. The rest is not really all that complicated (well, not if the computer does all the rendering for you).

Finally, there's absolutely nothing wrong with drawing comparisons between games. It's natural and inevitable. But it's nice to not have to resort to comparison when describing your own new game design. I'd much rather be able to say, "It's like nothing you've ever seen before!" ■

While a student at the Massachusetts College of Art, David Sieks was chastised for frequent absences from his technical drawing class. He was probably out playing video games. Dave can be reached via e-mail at dsieks@arnarb.harvard.edu or through Game Developer *magazine.*