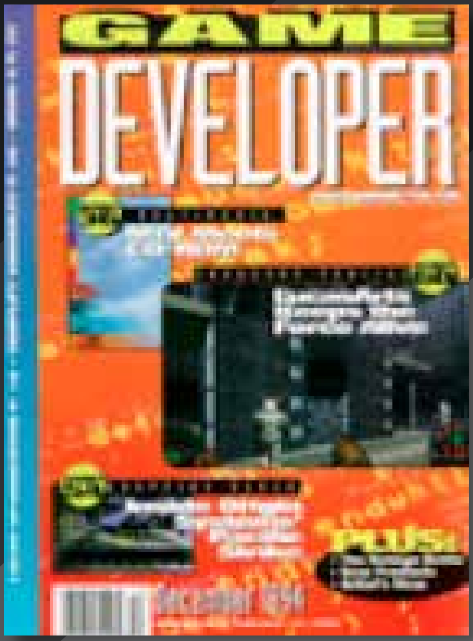
GAME DEVELOPER MAGAZINE

DECEMBER 1994



UBM Tech

A Rant on Ratings

"The true danger is when liberty is nibbled away, for expedience, and by parts."
—Edmund Burke

Rating systems are crap. With the entire entertainment industry rolling over whenever Congress calls a hearing, it's fallen on us to denounce these initiatives for what they are—cynical posturing and electioneering with no substance. Rating systems, whether for movies, television, video games, or any other form of communication, don't work, cost money, and impede creativity. Everyone at those hearings, politicians and witnesses alike, knows that. But there's nothing politicians love more than "standing up for the family" and blaming America's cultural violence on Hollywood. So the entertainment industry submissively pisses all over itself and proposes "voluntary" systems from the pathetic to the laughable.

What are rating systems supposed to do? Curb the glorification of violence and sex. But "glorification" cannot be quantified. Which of the movies *True Lies*, *Natural Born Killers*, *Schindler's List*, and *Wes Craven's New Nightmare*, glorifies violence and which denounces it? The rating system doesn't make a distinction—they're all rated R.

In the face of the failure of the MPAA rating system to control our national psyche, the ratings advocates don't make the logical conclusion that rating systems are bogus, but instead say that the problem is that not enough things are rated! If our culture's general level of sexual frankness and violence has risen despite the movie rating system, it's only because other forms of entertainment have not yet been purified.

Rating systems cost money. A ratings "board" (actually, a sizable and bureaucratic institution) and its cost will be borne by the software companies. So much for the garage-based entrepreneurs trying to pull themselves up by their bootstraps with shareware games.

Rating systems impede creativity. They do so in a perverse way, where brutalities are traded off against each other. I can personally attest to the kinds of discussions that happen all the time in Hollywood: "Page 48, you've got the forcible injection of drugs. That's an X, rewrite it. Page 72—the rape scene. That's O.K." This dehumanization is fine, that dehumanization isn't, let's do lunch.

But the refrain is always, "We need ratings systems to control what our children are exposed to." A parent in a video store once told me that her child might want to see R movies, but when she says no, the child knew not to argue. The corollary, I imagine, is that when she says no to a PG movie, the child argues. The favorite argument of "but all the other kids are watching it" is used by the child to subvert the very point of the rating system.

And ultimately, the incident holds the right answer to controlling what children are exposed to. Parents should decide. If parents don't want their kids to play X-Com or see *Terminator 2*, they should say "No" and put up with the ensuing argument. They don't need and shouldn't get a ratings system to supplement their authority. The government has no right to help parents say "No" at the video store if that governmental interference impedes your rights to develop whatever content you feel appropriate.

We all have responsibilities. To create responsibly, to control the viewing and gaming habits of our own children, and to call the government's ratings initiatives what they are. Cynical, ineffective, oppressive, and wrong-headed. ■

Larry O'Brien
Editor

GAME DEVELOPER

Editor Larry O'Brien
76702.705@compuserve.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
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 Constanza
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 Philip Payton *ppayton@mfi.com*
Newsstand Manager Pam Santoro
Reprints Krista Hiser (415) 905-2783

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

Client/Server Gaming?

by Our Readers

We've only been around a little while, but we're already hearing from readers. In this, the first installment of our forum for your opinions, we've got a sampling of queries, criticism, and suggestions.

Dear Editor:

I was very excited when I saw your new magazine. Oddly, it's not because I ever hope to write a game per se.

My job function is aptly described (using someone else's words) as a corporate cubbyhole programmer. I've been using Toolbook and Multimedia Toolbook for a couple years. I know it isn't as manly as C++ or even Visual Basic, but I can pretty much do all the things, in 60 scripting seconds, that take a *Dr. Dobb's* guy six nitty-gritty pages of text and code to describe.

Using Toolbook, calling Q+E as my database access and Pinnacle's Graphics Server SDK, I can really get fast access to CSV datafiles and plot everything from bar charts to interactive (hot spot) bubble charts. I create transformed reports (transports) from my customers that give good GUI, good drill-down, and good graphical constructs of their data.

Using Multimedia Toolbook, I capture mainframe screens and add neat postage-stamp video to accomplish CBT. My fuzzy-grey cubbyhole for a better compression algorithm (or a horse)!

So why am I telling this to *Game Developer* editors? Well, you've heard of infotainment and edutainment? Why not busutainment? Aside from its neat name?

My son plays Math Blaster for Windows. It rewards him with a spaceship launch or trash to zap after accomplishing certain goals. Adults may be more amused if surprised by some Easter egg they stumble across. I have been known to make a user chase a normal looking button that jumps away randomly on `mouseEnter`. If you are very quick, you can nail it.

Recently, a new game called Earth Invasion for Windows came out. It proves quite nicely that you can have arcade action in a "slow" graphical environment (I can spell that word because I was once a COBOL programmer). How much I would love to splice some of that action into my applications! Just to keep them eager to go on, of course.

Your article about OLE 2.0 ("Let's Go Embed," Premier 1994) was right on, and I hope someday it will allow me all sorts of opportunities to use interesting game-creating authoring tools. Multimedia has caused everybody to buy faster PCs. Certainly, they run Windows fast enough for something more than Solitaire now. Let's replace drag-and-drop with aim-and-fire. Twitch spreadsheets? Dbase thumb?

The more I read about desktop virtual reality, the more I see myself running through my relational database like in Wolfenstein 3D, past orthogonal buildings (disguised as three-dimensional bar charts) of my data looking for the evil odd piece of data. Not as compelling as I would like, but maybe Comptons has already patented it.

I get a small taste of this in Virtus VR, but they're more interested in showing off real buildings with real wire frame fireplaces and are not into anthropomorphizing bad data (with TrueType labels) into gun-toting Nazis—yet.

Edward Saur
Flemington, N.J.

Editor Larry O'Brien responds:
Heaven forbid that we talk about business software development in Game Developer*. But since you did bring it up: humans have a set of evolutionary talents*

that include competing, gathering, and hunting in a visually complex environment. The computer game industry makes billions off the fact that these are so wired into the wetware that humans will pay for the privilege of exercising them. A business program that took advantage of these evolutionary strengths would probably gain accolades for its "intuitive" and "friendly" interface.

How Do You Get to Carnegie Hall?

Dear, Editor:

Could you give me some pointers as to what a college graduate should do as far as looking for work and what I should learn on my own to get into the game developing field? I realize most game developers write games in their spare time or start their own companies, but I thought you might have some suggestions for recent grads on what kind of work they should look for to help expand their knowledge in the field. Do game development companies hire recent grads, and if so what would they look for?

Craig Tyler
via the Internet

Editor Larry O'Brien responds:
Your question is one that we get all the time, and we always give the same answer. There is no answer. Due to the relatively young age of the computer game industry, most of the game developers who are big names in the field started writing their own games themselves and companies grew up around them. Ask 10 game developers how they got into the business, and you'll get 10 different answers. We plan to focus on this in the future, but there's no simple answer that we can give you here.

Ratings Brouhaha

Dear Editor:

In response to Alex Dunne's editorial, "The Ratings Game" (Bit Blasts, Sept. 1994), I am sure we will see some form of ratings system imposed on the industry sooner than later, and I can only hope that the system's creators somehow keep simplicity in mind (that is, they should not focus too heavily on it).

I object to a simplistic level rating system (especially the one Dunne mentioned that was proposed by someone on CompuServe) because it does not work. Using the groups presented, Lemmings would possibly qualify as a 5. While laughing about how absurd this sounds, press the mushroom cloud to detonate the Lemmings on the screen and bits and pieces of stuff fly out of the figures. Even Super Mario Brothers would rate at least a 2—and possibly a 4, depending on which creature in which game you are considering.

Rather than try to put all of the acts in a game into one linear category, I suggest that the game industry use something similar to what I see now for many premium cable movie channels. The movies are rated with a simple letter code system that describes the content of the movies. AL means adult language, MV means mild violence, GV means graphic violence, AS means adult situations, N means nudity, BN means brief nudity, and so on. Of course no system is perfect, and certainly we need to work out what constitutes mild versus graphic violence. But the object of the rating system is to inform the potential buyer of the content of the product, not simply line up all of the games in an arbitrary linear scale.

John Durbetaki
Gaston, Ore.

More Praise, Please!

Dear Editor:

Caught your second issue and I loved it! As an aspiring game developer and local musician I found the articles deep and informative. It's good to know that some people know what guys like myself want in a magazine. I did notice some flaws in Jim Cooper's informative article ("Computer vs. Console," June 1994), though. The Sega Genesis runs at about 7MHz and has 128K of memory not 12MHz with 512K of RAM. I thoroughly enjoyed Cooper's article. All and all, I think *Game Developer* gives a unique perspective on the gaming industry rarely seen in other magazines.

Also, I was hoping *Game Developer* could get down and dirty on comparing the new consoles coming out and out already. For example, comparing the awesome Doom on the Jaguar vs. the Sega 32X version. I think readers would really like this.

Ray Rivera
Norfolk, Va.

A Real Plan

Dear Editor:

The boom in interactive entertainment is clearly reflected in the glut of new computer entertainment periodicals, but most developer's magazines followed other trends. And so it seems that the debut of *Game Developer* this year was timely and auspicious.

Although *Game Developer* is still thin, and has gotten pricey, each issue is more organized and carries the promise that later issues will be stronger. So far, the article topics have been appropriate, but unfortunately, much of the programming content lacks technical depth. Andre LaMothe's article, "The Mysterious Mode 13h" (Sept. 1994), was especially appalling. LaMothe says, "Mode 13h is an obscure graphics mode that few game developers have mastered." Whoa! If you want to increase readership, don't pursue the newbies who probably won't subscribe anyway.

Finally, I'm glad you gave standard names to your departments in the third issue, instead of article titles. The six departments do a lot to make *Game Developer* a "real" magazine.

Tom Park
via the Internet

Game Over!

We need your feedback! Send your cards, letters, and article suggestions to:

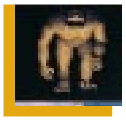
Game Developer
600 Harrison St., 4th Floor
San Francisco, CA 94107
Attn: Larry O'Brien

E-mail is even better:

Back

Child's Play

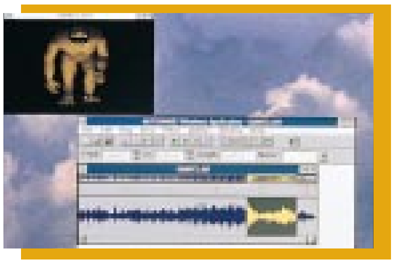
Diane Anderson and Nicole Claro



Autodesk and Microsoft offer two of four new animation options that promise to help you bring your graphics to life. Also, an update on the current game ratings trend.

PRODUCTS

Walt Disney, Watch Out!



Having recently broken my finger, I am more than ever a slave to my computer. I can't write anything with a normal implement, so I have to skip the "pen-and-paper" step and jump right to the keyboard. Not only that, but I'm obsessed with the fact that if I put my hair into a ponytail, it will always flop to the left (it's messing up my karma, I think). Oh how I wish I had a cool, new animation program to take me away....

Autodesk's award-winning, widely used two-dimensional animation software, Animator Pro, has been around for a few years now. Recently, the company announced the upcoming launch of Animator Studio Release 1, a Windows 3.1-based successor to the Animator Pro tools. The company calls Animator Studio a "powerful...2D animation production package with direct application to 2D film, TV, and video production."

With its new product, Autodesk seeks to break the boundaries between multimedia development venues and traditional graphic art studios. Animator Studio merges the two, allowing the user to create animation with "onion skinning," much like working on a light table; integrating 24-bit Truecolor paint and plug-in interfaces for Photoshop filters to be used as ink, which make the transition from still-image editing to animation even smoother; and providing a digital sound studio designed specifically for animation. The sound module lets you manipulate soundtracks in any way you need to, squeezing or stretching music to make it fit, recording from CDs and tapes and then layering the sounds, or adding special effects such as reverb or pitch shifting to music or vocals. While recording your sound or narration, you can view your animation in a separate video window simultaneously.

Autodesk Animator Studio reads and writes in many formats, including AVI, FLI and FLS, BMP, TGA, TIFF, JPEG, GIF, and PCX. Suggested list price is $795.

For More Information Contact:
Autodesk Inc.
2320 Marinship Way
Sausalito, Calif. 94965
Tel: (415) 332-2344

Marriage Announcement

3DLabs and Argonaut Software are joining forces.

3DLabs' GLINT two-dimensional processor is capable of 300K shaded, depth bufffered, anti-aliased polygons per second. It offers Gourard shading and texture mapping, 32-bit color, two- and three-dimensional acceleration, and an on-chip Peripheral Component Interconnect (PCI) local bus interface. According to 3DLabs, the GLINT processor is "ideal" for OpenGL or other 3D APIs.

Argonaut Software's BRender three-dimensional API provides scalable performance, true z-buffering, and perspective texture mapping across a wide range of computer and games platforms. BRender is intuitive and compact.

Combine GLINT and BRender and you've got high-performance rendering power. The partnership of the two companies promises to be a formidable force in the Windows and WindowsNT graphics market.

For More Information Contact:
3DLabs Inc.
2010 N. First St., Ste. 403
San Jose, Calif. 95131
Tel: (408) 436-3455

Winny the Toon

WinToon is Microsoft's latest product announcement. Addressing the strong demand for animated titles, Microsoft's new animation playback tool for Windows facilitates the creation of full-screen multimedia titles.

Its toolset includes the WinToon run-time engine. WinToon uses the Video for Windows architecture to allow developers blue-screen capabilities for superimposing images (remember *Jurassic Park*?). WinToon's scan-time engine digitizes traditionally animated characters, and its playback engine helps with design review. Microsoft promises the user "richer, more fluid, and more realistic animation" with WinToon.

Because the new Windows operating system will have a WinG API and a display control interface (DCI) built in—along with 32-bit implementation of the Video for Windows architecture—multimedia soon may have set standards.

For More Information Contact:
Microsoft Inc.
1 Microsoft Way
Redmond, Wash. 98052-6399
Tel: (800) 426-9400

Industry News: More to Hate About Ratings

First we had to deal with our parents forbidding us from seeing R-rated movies, then came that music label identification scandal, now this. Now our games and the very digital frontier we hold sacred are being threatened. Cyberia is inherently anarchic, chaotic, and methodic in its madness. Down with censorship! See Alex Dunne's comments ("The Ratings Game," Bit Blasts, Sept. 1994) and Larry O'Brien's comments in this month's Game Plan.

The Recreational Software Advisory Council (RSAC), a group established to "implement and oversee a national ratings system," recently elected Robert Roden as its president. "So what!" you scorn. "Who's he?" you ask. Well, as it turns out Robert Roden is not only the president of RSAC, he's also a member of the LucasClub. He's the "general counsel and director of business affairs" for LucasArts Entertainment.

Coincidence? Conflict of interests? Ironically, RSAC's nine-member board is designed to "ensure the ratings system will be independent of the software industry's control." With a software bigwig as its president? Seems unlikely.

The RSAC rating system assigns to a product a numerical score between zero and four across three categories—violence, sex/nudity, and language. Any scores above zero are posted on the product's packaging. Developer participation is voluntary, but retailers are wary of carrying nonrated products.

"RSAC's system has nothing to do with censorship or with telling developers what to put in their games," Roden says. (But developers who get an unfavorable rating will obviously be stigmatized; such labeling may damage or boost sales.) Roden innocently claims the organization's "purpose is to inform consumers what's in the box." If parents are so concerned with what's in boxes they're buying for their kids, parents should consider checking out the games themselves instead of depending on an outside agency to babysit the floppy babysitter. What ever happened to quality time?

And why haven't *Tom Sawyer* and *Catcher in the Rye* been banned from all high school reading lists yet?

One Board, One World

Matrox Graphics Inc. is now shipping the MGA Impression Plus, a one-board, three-dimensional graphics accelerator that provides fast Windows, three-dimensional, and video on the PC. The MGA Impression Plus features a flexible architecture that provides complex drivers to optimize Windows, multimedia, CAD, and three-dimensional applications. The 64-bit acclerator uses Microsoft's DCI API for smooth video playback at resolutions up to 1,280-by-1,024 at 30 frames per second. It's also the only graphics accelerator that lets you upgrade to the VESA Media Channel (VM-Channel). The VM-Channel's open specification accepts up to 15 video streams, so you can add on any VM-based multimedia peripheral.

The MGA Impression Plus is shipping for $449 and includes the MGA three-dimensional-SuperPack, a three-dimensional CD-Rom with three games—Sentõ, Spectre MGA, and Ice Hawk—three-dimensional viewing files, and three-dimensional library demos.

For More Information Contact:
Matrox Graphics Inc.
1055 Saint-Regis
Dorval, Canada H9P 2T4
Tel: (514) 685-2630

Diane Anderson is editorial assistant for Game Developer *magazine. Nicole Claro is production editor for* Game Developer *magazine.*

Revenge of the Sequel Syndrome

by Alex Dunne

Like many current movies, popular games are spawning an explosion of sequels, look-alikes, and wannabes. Alex Dunne looks at the trend and makes his case for originality in concept and design.

What do *Friday the 13th*, *Rocky*, and *Nightmare on Elm Street* have in common? If you answered, "They're all movies that spawned a never-ending series of sequels," you got it. It's a formula that works for the studios: After the first movie is deemed a success, *Movie 2* or *Revenge of the Bad Guy From The Last Movie* invariably comes out the following year. At the same time, the rest of the movie industry latches onto the "look and feel" of the movie and releases look-alikes.

You've probably seen enough of these movies to know the formula. Perhaps you enjoyed a few of them, too. But how many of these sequels and look-alikes would you describe as unforgettable? If I saw a movie and came out of it thinking, "That was *Raiders of the Lost Ark* starring Michael Douglas," I wouldn't recommend it to anyone. Unfortunately, I'm feeling this way about a growing number of games.

I fear that this mentality is becoming prevalent within the game industry as well. How many different football and baseball games are on the market today? How many air-combat simulators? If I had to sum it up, I'd say, "a lot." Are there too many? At the rate that sequels and look-alikes are hitting the market, I'd hazard to say that most of them are financially successful enough to justify their development. But as a developer, you've got to ask yourself, "Should I ride the coattails of the latest game craze or create a completely new type of game and risk an unresponsive market?" In response to this dilemma, let's look at:

- Some reasons for developing a sequel or look-alike game
- What Windows '95 offers that will spur developers' imaginations
- A relationship between a hardware vendor and game developer that spawned a stunning new game.

O.K., There are Compelling Reasons

Don't get me wrong, there are sound reasons for developing look-alikes and sequels. Look-alikes tap into hot market crazes and can turn a great profit for their developers. The reception that Doom received has companies like Capstone negotiating with Id Software to use its technology. As a result, Capstone's Corridor 7 looks strikingly similar to Doom—but I'll venture that it does fairly well in the market.

Sequels tap into name recognition and their predecessor's story lines and are especially prevalent in adventure and role-playing games. The market knows the products already, so most of the marketing department's job is done. The look and feel of the game is established, so creative folks don't have to sweat out many conceptual changes. Perhaps much of the interface code can be reused, saving the developers' time. With reasons like this, who'd knock the idea of a sequel?

If it's done right, probably nobody. I am simply championing the idea that you might be rewarded an order of magnitude more by going out on a limb with a fresh game idea than by choosing a less risky game concept. Develop a truly groundbreaking game and suddenly you're all alone, and your competitors are scrambling to catch up. Although it's been talked up to death, Myst is the perfect example. I find that when I try to describe Myst to someone, I can't use another game as a reference. Myst, by virtue of being so different, has been a financial boon for both the Rand brothers and Broderbund.

Microsoft's New Game Platform

Among the most talked about events in the industry and one that has significant potential to change the face of gaming is Windows '95. Now that Microsoft has gone through two beta cycles of the operating system, developers are getting a feel for its abilities. The capabilities Windows '95 offers over DOS should lead to some incredible new games.

Windows '95 promises to be a competitive platform to DOS when it comes to game speed and adds some benefits not found in the old 16-bit operating system. Coupled with OLE, Microsoft envisions the ability to drag and drop game elements, the use of standard game interfaces that developers could use to extend a game's functionality, and the possibility of embedding game sessions into mail messages that would automatically connect you to another player over a network or modem.

The ability to develop network multiplayer games will be greatly enhanced with Windows '95. Developers won't have to write the networking code or work around the network's memory space. Using WinSockets, game developers will be able to write games for a wide range of networks including Novell, TCP/IP, Windows for Workgroups, Banyan Vines, and LAN Manager.

In what looks to be a boon for the phone companies, Windows '95 will provide support for a new modem technology called VoiceView, which will ship as a standard feature of many modems beginning in 1995. VoiceView will let modem users talk on the phone and use their modem over the same line. The technology is suited to games where each player takes a turn, such as chess, allowing the computer to take over the phone line to transmit game data and then relinquish it back to the callers for further conversation. Simple messages (such as a chess move) would take less than a second.

This will make modem-based games much more popular because players will be able to enjoy the benefits of coordinating team play (or cursing at each other) as they play. This of course will be a tremendous boon to game distributors who bundle phone cradles and chiropractic coupons with their games.

Perhaps the most significant aspect of the new Windows '95 system is the much touted WinG graphic library. I don't have to convince anyone how pitiful game performance is under Windows 3.1. Microsoft claims that its come close to matching DOS game speeds using the WinG library under Windows '95.

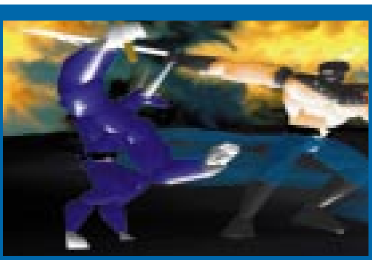
It's unclear when Windows '95 is going to be released. As of late October, I can only say (as Microsoft is) that it will be available sometime between April and June 1995. The acceptance rate of Windows '95 into homes is hard to project, however, so it's difficult to determine how large the market for Windows '95 games will be. Given Solitaire's popularity when Windows 3.0 came out, Microsoft is probably praying for some killer Windows '95 games to speed the system's adoption into family rooms. These "killer" games, I predict, will be innovative and make the most of Windows '95's enhancements.

Hardware Advancements Spur Innovation

Another angle developers are beginning to pursue in their quest for breakthrough games relates to hardware advancements. Back when sound cards first entered the market, card vendors courted developers to write games that supported their hardware. As a result of Creative Labs' diligence and marketing savvy, every major game today supports the Sound Blaster, and it has become the de facto standard.

Now, a video card manufacturer, Matrox, has come along with the powerful new MGA Impression Plus accelerator card that incorporates a three-dimensional polygon engine. Matrox is hoping to popularize the accelerator among game players and, to further this aim, has established a partnership with start-up game developer 47-Tek. 47-Tek is creating a battle game titled Sentõ that is optimized for the card and features very cool real-time animation. Sentõ's use of the card's engine creates a unique visual experience that makes other fighting games pale in comparison.

Unfortunately, Sentõ's performance under other video cards doesn't match the optimized conditions that the Matrox accelerator provides. It's possible to foresee a game so narrowly optimized for a particular computer configuration that a nonoptimal setup would bring the game to its knees. That, of course, would seriously affect the size of the game's market. 47-Tek has sidestepped this problem by bundling Sentõ with the hot-selling Matrox card, but of course that's not an option for every developer.


Other combat games pale in comparison to Sentõ, which uses the Matrox accelerator to its maximum potential.

Perhaps the Matrox card will become a de facto standard for three-dimensional polygon-based games—though I doubt it (the world can only handle so many standards). On the other hand, the symbiotic relationship between Matrox and 47-Tek might become a standard, one best suited for start-up developers looking for a niche.

So, as you begin brainstorming your next game, keep in mind the sequel syndrome. Experiment with new concepts and ideas, rather than rehashing old ones. Look around at new technologies the industry is offering, and capitalize on them. It comes down to this: Do you want your game to be the next groundbreaking hit or the next *Romancing the Stone*? Take the road less travelled. ■

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

Mode X Revealed

by Matt Pritchard

Frustrated by the limitations of Mode 13H? Try Mode X, the graphics mode that promises better screen resolutions, page flipping, hardware screen scrolling, and super rich color.

Graphics are one of the most important parts of any game today, yet the VGA standard comes up short in providing the features and power many games demand. "More colors!" That's what the VGA promised over its predecessors. "Pick any 256 from a quarter million!" it said. Unfortunately, as designed by IBM, only one mode actually delivered this promise of rich color: Mode 13h.

But there was a catch. Mode 13h did not provide many features game designers wanted—features they were accustomed to on other platforms. Screen resolution was limited. Hardware scrolling was out. Page flipping was a pipe dream. If you wrote a game using mode 13h, screen flicker was a constant enemy that limited your graphics and animation. Super VGA cards came out and provided more 256-color mode options, and some even provided page flipping. But there were no standards between brands and makes of VGA cards. Developers needed something that would work on any VGA card, from the first IBM VGA built into a PS/2 to the latest 4MB super duper VGA card. That something turned out to be Mode X.

Back in 1989, someone got down with the IBM VGA internals, took a close look at the internal registers and workings of Mode 13h, and discovered that nothing was etched in stone. This unsung hero determined that features from the EGA 16-color modes could be blended into Mode 13h to create a hybrid video mode with more accessible video memory and more usable hardware features. From that discovery we have what is commonly referred to as Mode X. Mode X gives us access to the following features that are lacking in Mode 13h:

- Higher screen resolutions
- Hardware screen scrolling
- Page flipping (the key for smooth animation).

The Technical Details

Game programmers everywhere have heard of Mode X, but for many it is still mysterious or unclear. The biggest reason for confusion is that Mode X is not a hard-and-fast video mode, but a label for any number of derived modes that have one thing in common: Unchained Video Memory Access. Let me explain.

A big limitation of VGA graphic modes is that there is only a 64K window of address space at segment `A000` through which to access graphics memory. EGA 16-color modes allow access to up to 256K of video memory by mapping four "video planes" over each other in the same 64K of address space. So, 4 bytes of video memory are actually mapped into each single byte of address space. Special control registers let the programmer set the four video planes which are actually mapped in at any given time. In Mode 13h, none of this exists. There is only one byte of video memory for each memory address, limiting this mode to 65,536 pixels maximum. With a screen resolution of 320-by-200, 64,000 of those pixels are used. Tragically, 192K of a standard VGA card's 256K of video

Listing 1. Modex.C (Continued on p. 27)

```
#include <stdio.h>
#include "setmodex.h"

void show_X_Mode(int Mode_Num, int X_Res, int Y_Res, int Scroll_Flag);
void modex_rect (int UL_X, int UL_Y, int LR_X, int LR_Y, int Color);
void modex_hline (int Left_X, int Right_X, int Y, int Color);
void modex_vline (int Top_Y, int Bottom_Y, int X, int Color);

void main (void)
{
    show_X_Mode(Mode_320x200, 320, 200, 0);
    show_X_Mode(Mode_320x240, 320, 240, 0);
    show_X_Mode(Mode_320x400, 320, 400, 0);
    show_X_Mode(Mode_320x480, 320, 480, 0);

    show_X_Mode(Mode_360x200, 360, 200, 0);
    show_X_Mode(Mode_360x240, 360, 240, 0);
    show_X_Mode(Mode_360x400, 360, 400, 0);
    show_X_Mode(Mode_360x480, 360, 480, 0);

    show_X_Mode(Mode_320x200, 512, 500, 1);

    set_text_mode();
    printf ("This Demo is finished\n");
}

void show_X_Mode(int Mode_Num, int X_Res, int Y_Res, int Scroll_Flag)
{
    int x, y;

    set_text_mode();
    if (Scroll_Flag == 0) {
        printf ("\n\nPress any key to see Mode X at %d by %d resolution\n\n",
                X_Res, Y_Res);
    } else {
        printf ("\n\nPress any key to see a Mode X Scrolling window\n\n");
    }
    printf("(When done, press any key to end the mode X display)");
    y = scan_keyboard();

    set_vga_modex(Mode_Num, X_Res, Y_Res);

    for (x = 0; x < 15; x++) {
        modex_rect(x*3, x*3, X_Res-x*3-1, Y_Res-x*3-1, x+1);
        modex_hline(32+x*4, X_Res-96+x*4, 20+x*10, x+1);
        modex_vline(32+x*4, Y_Res-96+x*4, 20+x*10, x+1);
    }

    if (Scroll_Flag) {
        for (x = 0, y = 0; x < 100; x++, y++)  set_window(x, y);
        for (y = 100; y >= 0; y--)             set_window(100, y);
        for (x = 100; x >= 0; x--)             set_window(x, 0);
    }

    y = scan_keyboard();
    return;
}
```

memory goes to waste here, in the only 256-color mode. The "secret" of Mode X is that it lets the programmer use the full 256K of VGA memory instead of a mere 64K by borrowing the EGA's multiple video plane system. This is accomplished by turning off a control register known as Chain-4 (unchaining) in the VGA card. Once this is done, 4 bytes of video memory are found at every memory location in the A000 segment.

With a quarter million pixels available to work with, the first thing people experimented with was changing the screen's displayed resolution. Mode 13h has no memory to spare for a larger screen, but Mode X has enough memory for a screen resolution of 512-by-512 pixels. However, just because there is enough memory doesn't mean the VGA card can display it. Reprogramming the registers that control the screen resolution is tricky. It is possible to play around with the VGA's CRT controller registers and come up with all sorts of strange resolutions. However, we want to stick with known settings that will work on any monitor and VGA card. Two horizontal resolutions and four vertical resolutions that follow established VGA modes fit our needs. These can be combined to create eight different screen resolutions for Mode X.

The standard Mode 13h resolution is 320-by-200 pixels and makes a good default for Mode X. This is the Mode X resolution used by games such as Doom and Ultima Underworld. The horizontal resolution can be safely increased to 360 pixels by setting the VGA card to use the 28MHz dot clock (used in text modes) instead of the 25MHz dot clock used in graphics modes. Some individuals claim that the higher frequencies damage their monitors. This is not true, and there has never been a known case of monitor damage because of it. Regrettably, higher horizontal resolutions such as 640 pixels are not available in a standard VGA because of video memory clock speed limitations.

Vertical resolution provides even more choices; 200-line modes are actually 400-line modes in disguise. These modes use the MSL register in the VGA's CRT controller to draw each scan line twice. Set that register to 0, and you have 400 lines of vertical resolution. By borrowing the vertical display settings from Mode 12h (640-by-480, 16-color graphics), we get 480 lines of glorious 256-color graphics, although the screen refresh rate slows from 70Hz to 60Hz. Finally, in this mode, setting the MSL register back to 1 gives you 240 pixels vertically.

Horizontal and vertical screen resolutions can be combined as desired, but one combination is worthy of special notice: 320-by-240 pixels. On a VGA monitor, this resolution has an aspect ratio of 1:1, which gives it perfectly square pixels and easily allows for true squares and circles on the screen.

Once you set a screen resolution, you can set a "virtual resolution." This lets you create a "virtual screen" larger than the displayed resolution. The screen display then becomes a window into the larger virtual screen, and the "window" can be moved and scrolled around. In Mode X, this is accomplished by setting the VGA card's Offset Register with the width of the desired virtual screen divided by 8. This creates the virtual screen, which can be wider than the 320 or 360 displayed pixels. The Start Address registers the position of the upper left corner of the screen display in video memory in the VGA's CRT controller control. Because each address has 4 bytes (4 pixels at 1 byte per pixel) mapped into it, when you change the Start Address register by 1, the screen display will shift by 4 pixels. To move a single pixel at a time, use the Horizontal Pixel Planning register in the VGA's Attribute controller. It can only shift the display up to 3 pixels in Mode X, but when you use it in conjunction with the Start Address registers, smooth, full-screen scrolling is possible.

Setting Mode X

So far, I have avoided getting very specific about the VGA registers and processes. At the end of this article, I have included an assembly language listing, Modex.asm (Listing 3), which

Listing 1. Modex.C (Continued from p. 26)

```
/* =====  Simple Mode-X Line draw routines ===== */

void modex_rect (int UL_X, int UL_Y, int LR_X, int LR_Y, int Color)
{
    modex_hline(UL_X, LR_X, UL_Y, Color);
    modex_hline(UL_X, LR_X, LR_Y, Color);
    modex_vline(UL_Y, LR_Y, UL_X, Color);
    modex_vline(UL_Y, LR_Y, LR_X, Color);
    return;
}

void modex_hline (int Left_X, int Right_X, int Y, int Color)
{
    int x;
    for (x = Left_X; x <= Right_X; x++) set_point(x, Y, Color);
    return;
}

void modex_vline (int Top_Y, int Bottom_Y, int X, int Color)
{
    int y;
    for (y = Top_Y; y <= Bottom_Y; y++) set_point(X, y, Color);
    return;
}
```

Listing 2. Modex.H

```
#ifndef __SETMODEX_H
#define __SETMODEX_H

    /* ===== SCREEN RESOLUTIONS ===== */

#define Mode_320x200  0
#define Mode_320x400  1
#define Mode_360x200  2
#define Mode_360x400  3
#define Mode_320x240  4
#define Mode_320x480  5
#define Mode_360x240  6
#define Mode_360x480  7

    /* ===== MODE X SETUP ROUTINES ===== */

int far pascal set_vga_modex (int Mode, int MaxXpos, int MaxYpos);
void far pascal set_text_mode (void);
int far pascal scan_keyboard (void);

    /* ===== BASIC GRAPHICS PRIMITIVES ===== */

void far pascal set_point (int Xpos, int Ypos, int Color);
int  far pascal read_point (int Xpos, int Ypos);
void far pascal set_window (int XOffset, int YOffset);

#endif
```

contains a routine that will set up Mode X at any of the eight resolutions that I described and create any sized virtual screen to order. Additional routines, which we will discuss, show how to plot pixels, read pixels, and scroll a virtual screen. I have also included two short demo programs, Modex.C and Modex.H, shown in Listings 1 and 2 and written in Borland C++ 3.1. They show each Mode X screen resolution and virtual screen scrolling. Both listings use the medium memory model.

Switching the computer's display into Mode X involves a few basic steps. First, you call the VGA's BIOS to set the display to Mode 13h. Some people have bypassed this step and do it manually, but I advise against it. Many super VGA cards have extra registers and features that are not part of the VGA standard. Calling the BIOS allows the VGA card to set any special or unique features it might have and ensures maximum compatibility. After Mode 13h is set, you disable the Chain-4 register in the VGA sequencer. Next, you will need to invoke an asynchronous reset on the VGA card. While the VGA is in a reset state, you can change the horizontal dot clock. After ending the reset, you then load the CRT Controller's registers with the values for the desired horizontal and vertical resolutions. Some registers are protected from accidental modification so a protection register must be turned off before starting and turned back on when finished. Finally, the virtual screen width must be set. At this point, the desired version of Mode X is set up and ready to go. Usually, the next step is to clear all the VGA's video memory completely, so any leftover garbage will not appear on the screen. To do this, you need to know how to write pixels in Mode X.

Mode X is not a hard-and-fast video mode, but a label for derived modes that have Unchained Video Memory Access.

Drawing Mode X Pixels

In Mode 13h, drawing pixels is very simple. You just write the desired byte value to address (320 * Y) + X. In Mode X, it gets a lot more complicated because there are four pixels at each memory address. This is where the Map Mask register comes in. As part of the VGA's sequencer, it acts like a control valve for each of the four video planes, determining if a byte written by the CPU will be passed through to each of the four video planes or ignored completely. While it is a hassle to set the Map Mask register for each pixel to plot, doing so can be beneficial if we want to write the same color value to more than one video plane at a time. If all video plane access bits in the Map Mask register are set to 1, a single byte written by the CPU will be simultaneously copied to all four video planes, setting four pixels with a single byte. Alas, this is a topic unto itself that we must save for later.

So, how do you determine the address of a given pixel that you want to write to location (X,Y)? The formula is similar to mode 13h but requires two more steps. First, you multiply the Y position by the virtual screen width (or display screen width if they are the same) and add the X position—just like Mode 13h, except that the width does not have to be 320. Then you take the result and divide it by 4. The whole number that results is the address in the A000 segment that you need to write to, and the remainder is the video plane number from 0 to 3, as shown here:

```
Raw_Address = (Y_pos * Screen_Width)
              + X_pos
Memory_Address = Raw_Address / 4
Plane_Number = Raw_Address MOD 4

"MOD" is the modulus/remainder function.
In C/C++ this is the "%" operator.
```

Because the numbers involved can exceed 65,536, most people first divide the video width and X position by 4 to avoid an overflow and get the plane number from the X position, as shown in this equation:

```
Memory_Address = (Y_pos * Screen_Width
                 / 4) + X_Pos / 4
Plane_Number = X_pos MOD 4
```

In either case, you wind up with a memory address and plane number. The four pixels at a given memory address will appear next to each other, left to right starting with plane #0. The very first pixel is at address 0, plane #0, followed by address 0, planes #1, #2, and #3. Then comes address 1, plane #0, and so on.

Before you can write the color value to the memory address, the Map Mask register must have the bit corresponding to the desired plane number set. You do this by loading the value 01h for plane #0, loading 02h for plane #1, loading 04h for plane #2, or loading 08h for plane #3 into the AH register, loading 02h into the AL register (to select the Map Mask register), and OUTing the AX register to port address 03C4h. Once that is done, any data you write to A000:Memory_Address will only go to the desired video plane. Examine the SET_POINT routine in Listing 3 for an example of Mode X pixel plotting.

Reading Mode X Pixels

Reading a pixel in Mode X is similar to writing one, except you can access only one video plane at a time. You still need

Listing 3. Modex.asm (Continued on p. 30)

```
    .MODEL Medium
    .286
    ; Macros to OUT 8 & 16 bit values to an I/O port
 OUT_16 MACRO Register, Value
    IFDIFI <Register>, <DX>      ; If DX not setup
        MOV  DX, Register        ; then Select Register
    ENDIF
    IFDIFI <Value>, <AX>         ; If AX not setup
        MOV  AX, Value           ; then Get Data Value
    ENDIF
        OUT  DX, AX              ; Set I/O Register(s)
ENDM
 OUT_8 MACRO Register, Value
    IFDIFI <Register>, <DX>      ; If DX not setup
        MOV  DX, Register        ; then Select Register
    ENDIF
    IFDIFI <Value>, <AL>         ; If AL not Setup
        MOV  AL, Value           ; then Get Data Value
    ENDIF
        OUT  DX, AL              ; Set I/O Register
ENDM
    ; macros to PUSH and POP multiple registers
 PUSHx MACRO R1, R2, R3, R4
    IFNB <R1>
        PUSH  R1                 ; Save Register
        PUSHx R2, R3, R4
    ENDIF
ENDM
 POPx MACRO R1, R2, R3, R4
    IFNB <R1>
        POP   R1                 ; Restore Register
        POPx  R2, R3, R4
    ENDIF
ENDM
    ; ===== VGA Register Addresses =====

    ATTRIB_Ctrl     EQU 03C0h   ; VGA Attribute Controller
    GC_Index        EQU 03CEh   ; VGA Graphics Controller
    SC_Index        EQU 03C4h   ; VGA Sequencer Controller
    CRTC_Index      EQU 03D4h   ; VGA CRT Controller
    MISC_OUTPUT     EQU 03C2h   ; VGA Misc Register
    INPUT_1         EQU 03DAh   ; Input Status #1 Register
    ; ===== VGA Register Index Values =======
    PIXEL_PAN_REG   EQU 033h    ; Atrb Index: Pixel Pan Reg
    MAP_MASK        EQU 002h    ; Sequ Index: Map Mask Reg
    READ_MAP        EQU 004h    ; GC Index: Read Map Reg
    START_DISP_HI   EQU 00Ch    ; CRTC Index: Disp Start Hi
    START_DISP_LO   EQU 00Dh    ; CRTC Index: Disp Start Lo
     ; ===== other VGA Register Values & Constants =====
    ALL_PLANES_ON   EQU 00F02h  ; Map Register + All Planes
    CHAIN4_OFF      EQU 00604h  ; Chain 4 mode Off
```

to calculate the memory address and plane number, but you will set the Read Map register in the VGA's Graphics controller instead of the Map Mask register. Unlike the Map Mask register, this register just needs the plane number from 0 to 3.

Specifically, you need to load AH with the plane number, AL with 04h (to select the Read Map register), and OUT AX to 03CEh. After that, you read the byte at A000:Memory_Address and it will return the value of the desired pixel. Examine the READ_POINT routine in Listing 3 for an example of Mode X pixel reading.

Scrolling the Screen

Once you know how to address pixels in Mode X, it is easy to scroll a window around a larger virtual screen. All you need to know is which pixel should appear in the upper left corner of the screen. With that, you compute the address and plane number of the pixel. The 16-bit address is loaded into the CRT controller's Start Address registers. You then multiply the plane number by 2 and load it into the Horizontal Pixel Panning register of the VGA's Attribute controller. Occasionally, this process may produce shearing of the display screen. To avoid this, the Horizontal Pixel Panning register is usually updated during the display's vertical blank period. Examine the SET_WINDOW routine in Listing 3 for an example of virtual screen scrolling.

Whew! This Mode X stuff is certainly more involved than Mode 13h, but the results are worth it. We have only scratched the surface of Mode X, and in future articles I hope to cover ways to use Mode X's unique features to produce fast and efficient graphics routines such as page flipping, block fills, and sprites. Until next time, happy coding! ■

Matt Pritchard is a software developer for Lacerte Software in Dallas Texas, and the author of MODEX110 a comprehensive freeware Mode X library. He can be reached at matthewp@netcom.com or through Game Developer.

Listing 3. Modex.asm (Continued on p. 31)

```
    ASYNC_RESET     EQU 00100h  ; (A)synchronous Reset
    SEQU_RESTART    EQU 00300h  ; Sequencer Restart
    VGA_Segment     EQU 0A000h  ; VGA Memory Segment
    VERT_RETRACE    EQU 08h     ; INPUT_1: Vert Retrace Bit
    PLANE_BITS      EQU 03h     ; Bits 0-1 of X = Plane #
    nil             EQU 00h     ; Used to mark end of list
    wptr            EQU WORD PTR    ; Shorthand text
    dptr            EQU DWORD PTR   ; macros for pointers
     .DATA?     ;==== DGROUP STORAGE NEEDED (10 BYTES) ====
CURRENT_PAGE     DW  0        ; Offset of current Page
CURRENT_SEGMENT  DW  0        ; Segment of VGA memory
SCREEN_WIDTH     DW  0        ; Width of a line in Bytes
MAX_XOFFSET      DW  0        ; Current Display X Offset
MAX_YOFFSET      DW  0        ; Current Display Y Offset
    ; Mode X Mode list data table format...
 Mode_Data_Table STRUC
    M_MiscR      DB  ?,?      ; Value of MISC_OUTPUT register
    M_XSize      DW  ?        ; X Size Displayed on screen
    M_YSize      DW  ?        ; Y Size Displayed on screen
    M_XMax       DW  ?        ; Maximum Possible X Size
    M_YMax       DW  ?        ; Maximum Possible Y Size
    M_CRTC       DW  ?        ; Table of CRTC register values
Mode_Data_Table ENDS
    .CODE       ; Data Tables put in CS for easy access
     ; CRTC Register Values for Various Configurations
 MODE_Single_Line:   ; CRTC Data for 400/480 Line modes
    DW   04009H      ; Cell Height (1 Scan Line)
    DW   00014H      ; Dword Mode off
    DW   0E317H      ; turn on Byte Mode
    DW   nil         ; End of 400/480 line CRTC Data
 MODE_Double_Line:   ; CRTC Data for 200/240 Line modes
    DW   04109H      ; Cell Height (2 Scan Lines)
    DW   00014H      ; Dword Mode off
    DW   0E317H      ; turn on Byte Mode
    DW   nil         ; End of 200/240 Line CRTC Data
 MODE_320_Wide:      ; CRTC Data for 320 Pixels
    DW   05F00H      ; Horz total
    DW   04F01H      ; Horz Displayed
    DW   05002H      ; Start Horz Blanking
    DW   08203H      ; End Horz Blanking
    DW   05404H      ; Start H Sync
    DW   08005H      ; End H Sync
    DW   nil         ; End of 320 pixel CRTC Data
 MODE_360_Wide:      ; CRTC Data for 360 Pixels
    DW   06B00H      ; Horz total
    DW   05901H      ; Horz Displayed
    DW   05A02H      ; Start Horz Blanking
    DW   08E03H      ; End Horz Blanking
    DW   05E04H      ; Start H Sync
    DW   08A05H      ; End H Sync
    DW   nil         ; End of 360 pixel CRTC Data
```

Listing 3. Modex.asm (Continued on p. 32)

```
MODE_200_Tall:
MODE_400_Tall:      ; CRTC Data for 200/400 Line modes
    DW  0BF06H      ; Vertical Total
    DW  01F07H      ; Overflow
    DW  09C10H      ; V Sync Start
    DW  08E11H      ; V Sync End/Prot Cr0 Cr7
    DW  08F12H      ; Vertical Displayed
    DW  09615H      ; V Blank Start
    DW  0B916H      ; V Blank End
    DW  nil         ; End of 200/400 Line CRTC Data
MODE_240_Tall:
MODE_480_Tall:      ; CRTC Data for 240/480 Line modes
    DW  00D06H      ; Vertical Total
    DW  03E07H      ; Overflow
    DW  0EA10H      ; V Sync Start
    DW  08C11H      ; V Sync End/Prot Cr0 Cr7
    DW  0DF12H      ; Vertical Displayed
    DW  0E715H      ; V Blank Start
    DW  00616H      ; V Blank End
    DW  nil         ; End of 240/480 Line CRTC Data
    ; Table of Display Mode Components & Index Table
MODE_TABLE:
    DW  offset MODE_320x200, offset MODE_320x400
    DW  offset MODE_360x200, offset MODE_360x400
    DW  offset MODE_320x240, offset MODE_320x480
    DW  offset MODE_360x240, offset MODE_360x480
MODE_320x200:       ; Data for 320 by 200 Pixels
    DB  063h, 0     ; 400 scan Lines & 25 Mhz Clock
    DW  320, 200    ; Displayed Pixels (X,Y)
    DW  1302, 816   ; Max Possible X and Y Sizes
    DW  offset MODE_320_Wide, offset MODE_200_Tall
    DW  offset MODE_Double_Line, nil
MODE_320x400:       ; Data for 320 by 400 Pixels
    DB  063h, 0     ; 400 scan Lines & 25 Mhz Clock
    DW  320, 400    ; Displayed Pixels X,Y
    DW  648, 816    ; Max Possible X and Y Sizes
    DW  offset MODE_320_Wide, offset MODE_400_Tall
    DW  offset MODE_Single_Line, nil
MODE_360x240:       ; Data for 360 by 240 Pixels
    DB  0E7h, 0     ; 480 scan Lines & 28 Mhz Clock
    DW  360, 240    ; Displayed Pixels X,Y
    DW  1092, 728   ; Max Possible X and Y Sizes
    DW  offset MODE_360_Wide, offset MODE_240_Tall
    DW  offset MODE_Double_Line , nil
MODE_360x480:       ; Data for 360 by 480 Pixels
    DB  0E7h, 0     ; 480 scan Lines & 28 Mhz Clock
    DW  360, 480    ; Displayed Pixels X,Y
    DW  544, 728    ; Max Possible X and Y Sizes
    DW  offset MODE_360_Wide, offset MODE_480_Tall
    DW  offset MODE_Single_Line , nil
MODE_320x240:       ; Data for 320 by 240 Pixels
    DB  0E3h, 0     ; 480 scan Lines & 25 Mhz Clock
    DW  320, 240    ; Displayed Pixels X,Y
    DW  1088, 818   ; Max Possible X and Y Sizes
    DW  offset MODE_320_Wide, offset MODE_240_Tall
    DW  offset MODE_Double_Line, nil
MODE_320x480:       ; Data for 320 by 480 Pixels
    DB  0E3h, 0     ; 480 scan Lines & 25 Mhz Clock
    DW  320, 480    ; Displayed Pixels X,Y
    DW  540, 818    ; Max Possible X and Y Sizes
    DW  offset MODE_320_WIDE, offset MODE_480_Tall
    DW  offset MODE_Single_Line, nil
MODE_360x200:       ; Data for 360 by 200 Pixels
    DB  067h, 0     ; 400 scan Lines & 28 Mhz Clock
    DW  360, 200    ; Displayed Pixels (X,Y)
    DW  1302, 728   ; Max Possible X and Y Sizes
    DW  offset MODE_360_Wide, offset MODE_200_Tall
    DW  offset MODE_Double_Line, nil
MODE_360x400:       ; Data for 360 by 400 Pixels

    DB  067h, 0     ; 400 scan Lines & 28 Mhz Clock
    DW  360, 400    ; Displayed Pixels X,Y
    DW  648, 816    ; Max Possible X and Y Sizes

    DW  offset MODE_360_Wide, offset MODE_400_Tall
    DW  offset MODE_Single_Line, nil
;==========================================================
; int SET_VGA_MODEX (int Mode, int Max_XPos, int Max_Ypos)
;
; Sets Up the specified version of Mode X.  Allows for
; the setup of of a virtual screen which can be larger
; than the displayed screen.
;
; ENTRY: Mode_Type = Desired Screen Resolution (0-7)
;          0 =  320 x 200    4 =  320 x 240
;          1 =  320 x 400    5 =  320 x 480
;          2 =  360 x 200    6 =  360 x 240
;          3 =  360 x 400    7 =  360 x 480
;
;         Max_Xpos = The Desired Virtual Screen Width
;         Max_Ypos = The Desired Virtual Screen Height
;
; EXIT:  AX = Success Flag:  0 = Failure, -1 = Success

SVM_STACK   STRUC
    SVM_Table   DW  ?     ; Offset of Mode Info Table
                DD  ?,?,? ; DI, SI, DS, BP, Caller
    SVM_Ysize   DW  ?     ; Vertical Screen Size Desired
    SVM_Xsize   DW  ?     ; Horizontal Screen Size Desired
    SVM_Mode    DW  ?     ; Display Resolution Desired
SVM_STACK   ENDS
    PUBLIC  SET_VGA_MODEX
```

Listing 3. Modex.asm (Continued on p. 33)

```
SET_VGA_MODEX   PROC    FAR
    PUSHx   BP, DS, SI, DI      ; Preserve Registers
    SUB     SP, 2               ; Allocate workspace
    MOV     BP, SP              ; Set up Stack Frame
    ; Make Sure Mode, X and Y Sizes are legal
    MOV     BX, [BP].SVM_Mode   ; Get Requested Mode #
    CMP     BX, 8               ; 8 Modes max: Is it 0..7?
    JAE     @SVM_BadModeSetup   ; If Not, Error out
    SHL     BX, 1                   ; Scale BX to word
    MOV     SI, wptr MODE_TABLE[BX] ; CS:SI -> Mode Info
    MOV     [BP].SVM_Table, SI      ; Save ptr for later
    AND     [BP].SVM_XSize, 0FFF8h  ; X size Mod 8 Must = 0
    MOV     AX, [BP].SVM_XSize  ; Get Logical Screen Width
    CMP     AX, CS:[SI].M_XSize ; Check against Displayed X
    JB      @SVM_BadModeSetup   ; Report Error if too small
    CMP     AX, CS:[SI].M_XMax  ; Check against Max X
    JA      @SVM_BadModeSetup   ; Report Error if too big
    MOV     BX, [BP].SVM_YSize  ; Get Logical Screen Height
    CMP     BX, CS:[SI].M_YSize ; Check against Displayed Y
    JB      @SVM_BadModeSetup   ; Report Error if too small
    CMP     BX, CS:[SI].M_YMax  ; Check against Max Y
    JA      @SVM_BadModeSetup   ; Report Error if too big
 ; Make Sure there is Enough memory to Fit it all
    SHR     AX, 2               ; # of Bytes:Line = XSize/4
    MUL     BX                  ; DX:AX = Total mem needed
    JNO     @SVM_Continue       ; Exit if Total Size > 256K
    DEC     DX                  ; Was it Exactly 256K???
    OR      DX, AX              ; (DX = 1, AX = 0000)
    JZ      @SVM_Continue       ; if so, it's valid...
 @SVM_BadModeSetup:
    XOR     AX, AX              ; Return Value = False
    JMP     @SVM_Exit           ; Normal Exit
 @SVM_Continue:
    MOV     AX, 13H             ; Start with Mode 13H
    INT     10H                 ; Let BIOS Set the Mode
    OUT_16  SC_INDEX, CHAIN4_OFF         ;Disable Chain 4
    OUT_16  SC_INDEX, ASYNC_RESET        ;(A)sync Reset
    OUT_8   MISC_OUTPUT, CS:[SI].M_MiscR ;Set New Timings
    OUT_16  SC_INDEX, SEQU_RESTART       ;Restart Sequencer
    OUT_8   CRTC_INDEX, 11H     ; Select Retrace End Reg
    INC     DX                  ; Point to Data
    IN      AL, DX              ; Get Data, Bit 7=Protect
    AND     AL, 7FH             ; Mask out Write Protect
    OUT     DX, AL              ; And send it back
    MOV     DX, CRTC_INDEX      ; Vga Crtc Registers
    ADD     SI, M_CRTC          ; SI->CRTC Parameter Data
    ; Load Tables of CRTC Parameters from List of Tables
@SVM_Setup_Table:
    MOV     DI, CS:[SI]     ; Get Ptr to CRTC Data Table
    ADD     SI, 2           ; Point to next Ptr Entry
    OR      DI, DI          ; A nil Ptr means that we have
    JZ      @SVM_Set_Data   ; finished CRTC programming
@SVM_Setup_CRTC:
    MOV     AX, CS:[DI]         ; Get CRTC Data from Table
    ADD     DI, 2               ; Advance Pointer
    OR      AX, AX              ; At End of Data Table?
    JZ      @SVM_Setup_Table    ; If so, go get next Table
    OUT     DX, AX                ; Reprogram VGA CRTC reg
    JMP     short @SVM_Setup_CRTC  ; Get next table entry

    ; Initialize Page & Scroll info, DI = 0
@SVM_Set_Data:
    MOV     CURRENT_PAGE, DI    ; Offset into VGA memory=0
    MOV     AX, VGA_SEGMENT     ; Segment for VGA memory
    MOV     CURRENT_SEGMENT, AX ; Save for Future LES's
    ; Set Logical Screen Width, X Scroll and Our Data
    MOV     SI, [BP].SVM_Table  ; Get Saved Mode Info Ptr
    MOV     AX, [BP].SVM_Xsize  ; Get Display Width
    MOV     CX, AX              ; CX = Logical Width
    SUB     CX, CS:[SI].M_XSize ; CX = Max X Scroll Value
    MOV     MAX_XOFFSET, CX     ; Set Maximum X Scroll
    SHR     AX, 2               ; Bytes = Pixels / 4
    MOV     SCREEN_WIDTH, AX    ; Save Width in Pixels
    SHR     AX, 1               ; Offset Value = Bytes / 2
    MOV     AH, 13h             ; 13h=CRTC Offset Register
    XCHG    AL, AH              ; Switch format for OUT
    OUT     DX, AX              ; Set VGA CRTC Offset Reg
    ; Setup Data table, Y Scroll, Misc for Other Routines
    MOV     AX, [BP].SVM_Ysize  ; Get Logical Screen Height
    MOV     CX, AX              ; CX = Logical Height
    SUB     BX, CS:[SI].M_YSize ; CX = Max Y Scroll Value
    MOV     MAX_YOFFSET, CX     ; Set Maximum Y Scroll
    ; Clear all of VGA Memory
    OUT_16  SC_INDEX, ALL_PLANES_ON ; Select All Planes
    LES     DI, dptr CURRENT_PAGE   ; ES:DI -> A000:0
    XOR     AX, AX              ; AX = 0
    CLD                         ; Block Xfer Forwards
    MOV     CX, 8000H           ; 32K * 4 * 2 = 256K
    REP     STOSW               ; Clear video memory!
    MOV     AX, 0FFFFh          ; Return Success Code -1
@SVM_EXIT:
    ADD     SP, 2               ; Deallocate workspace
    POPx    DI, SI, DS, BP      ; Restore Saved Registers
    RET     6                   ; Exit & Clean Up Stack
SET_VGA_MODEX   ENDP
;==========================================================
; void SET_POINT (int X_pos, int Y_pos, int Color_Num)
;
; Plots a single Pixel on the active display page
;
; ENTRY: Xpos     = X position to plot pixel at
;        Ypos     = Y position to plot pixel at
```

Listing 3. Modex.asm (Continued on p. 34)

```
;       ColorNum = Color to plot pixel with
;
; EXIT:  No meaningful values returned
SP_STACK    STRUC
                DD  ?,? ; BP, DI, Caller
    SETP_Color  DB  ?,? ; Color of Point to Plot
    SETP_Ypos   DW  ?   ; Y pos of Point to Plot
    SETP_Xpos   DW  ?   ; X pos of Point to Plot
SP_STACK    ENDS
        PUBLIC SET_POINT, READ_POINT

 SET_POINT   PROC    FAR
    PUSHx   BP, DI               ; Preserve Registers
    MOV     BP, SP               ; Set up Stack Frame
    LES     DI, dptr CURRENT_PAGE  ; ES:DI -> A000:0
    MOV     AX, [BP].SETP_Ypos  ; Get Line # of Pixel
    MUL     SCREEN_WIDTH        ; Get Offset to Line Start
    MOV     BX, [BP].SETP_Xpos  ; Get Xpos
    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, MAP_MASK        ; Select Map Mask Register
    MOV     AH, 0001b           ; Start w/ Plane #0 (Bit 0)
    AND     CL, PLANE_BITS      ; Get Plane Bits
    SHL     AH, CL              ; Get Plane Select Value
    OUT_16  SC_Index, AX        ; Select Plane
    MOV     AL,[BP].SETP_Color  ; Get Pixel Color
    MOV     ES:[DI+BX], AL      ; Draw Pixel
    POPx    DI, BP              ; Restore Saved Registers
    RET     6                   ; Exit and Clean up Stack
SET_POINT       ENDP
;==========================================================
; int READ_POINT (int X_pos, int Y_pos)
;
; Gets the color of a pixel at (X_Pos, Y_Pos)
;
; ENTRY: X_pos = X position of pixel to read
;        Y_pos = Y position of pixel to read
;
; EXIT:  AX   = Color of Pixel at (X_pos, Y_pos)
RP_STACK    STRUC
            DD  ?,? ; BP, DI, Caller
    RP_Ypos DW  ?   ; Y pos of Point to Read
    RP_Xpos DW  ?   ; X pos of Point to Read
RP_STACK    ENDS

READ_POINT      PROC    FAR
    PUSHx   BP, DI               ; Preserve Registers
    MOV     BP, SP               ; Set up Stack Frame
    LES     DI, dptr CURRENT_PAGE  ; ES:DI -> A000:0
    MOV     AX, [BP].RP_Ypos     ; Get Line # of Pixel
    MUL     SCREEN_WIDTH        ; Get Offset to Line Start
    MOV     BX, [BP].RP_Xpos    ; Get Xpos
    MOV     CX, BX              ; Save to get shift count
    SHR     BX, 2               ; X offset (Bytes) = Xpos/4
    ADD     BX, AX              ; Offset = Offset + Xpos/4
    MOV     AL, READ_MAP        ; GC Read Mask Register
    MOV     AH, CL              ; Get Saved Xpos
    AND     AH, PLANE_BITS      ; mask out Plane #
    OUT_16  GC_INDEX, AX        ; Select Plane to read in
    XOR     AX, AX              ; Clear Return Value
    MOV     AL, ES:[DI+BX]      ; Get Color of Pixel
    POPx    DI, BP              ; Restore Saved Registers
    RET     4                   ; Exit and Clean up Stack
READ_POINT       ENDP
;==========================================================
; void SET_WINDOW (int X_pos, int Y_pos)
;
; Since a Logical Screen can be larger than the Physical
; Screen, Scrolling is possible.  This routine sets the
; Upper Left Corner of the Screen to the specified Pixel.
; When called, this routine syncronizes the display to
; the vertical blank.
;
; ENTRY: X_pos       = # of pixels to shift screen right
;        Y_pos       = # of lines to shift screen down
;
; EXIT:  No meaningful values returned
SW_STACK    STRUC
                DW  ?   ; BP
                DD  ?   ; Caller
    SW_Ypos     DW  ?   ; Y pos of UL Screen Corner
    SW_Xpos     DW  ?   ; X pos of UL Screen Corner
SW_STACK    ENDS
        PUBLIC SET_WINDOW

SET_WINDOW  PROC    FAR
    PUSH    BP                  ; Preserve Registers
    MOV     BP, SP              ; Set up Stack Frame
    ; Check if our Scroll Offsets are Valid
    MOV     AX, [BP].SW_Ypos    ; Get Desired Y Offset
    CMP     AX, MAX_YOFFSET     ; Is it Within Limits?
    JA      @SW_Exit            ; if not, exit
    MOV     CX, [BP].SW_Xpos    ; Get Desired X Offset
    CMP     CX, MAX_XOFFSET     ; Is it Within Limits?
    JA      @SW_Exit            ; if not, exit
    ; Compute proper Display start address to use
    MUL     SCREEN_WIDTH        ; AX=YOffset * Line Width
    SHR     CX, 2               ; CX / 4 = Bytes into Line
    ADD     AX, CX              ; AX=Offset of UL Pixel
    MOV     BX, AX              ; BX=Desired Display Start
    MOV     DX, INPUT_1         ; Input Status #1 Register
```

Listing 3. Modex.asm (Continued from p. 33)

```
    ; Wait if we are currently in a Vertical Retrace
@SW_WAIT0:
    IN      AL, DX              ; Get VGA status
    AND     AL, VERT_RETRACE    ; In Display mode yet?
    JNZ     @SW_WAIT0           ; If Not, wait for it
    ; Set the Start Display Address to the new window
    MOV     DX, CRTC_Index      ; We Change the Sequencer
    MOV     AL, START_DISP_LO   ; Display Start Low Reg
    MOV     AH, BL              ; Low 8 Bits of Start Addr
    OUT     DX, AX              ; Set Display Addr Low
    MOV     AL, START_DISP_HI   ; Display Start High Reg
    MOV     AH, BH             ; High 8 Bits of Start Addr
    OUT     DX, AX              ; Set Display Addr High
    ; Wait for a Vertical Retrace to smooth out things
    MOV     DX, INPUT_1         ; Input Status #1 Register
@SW_WAIT1:
    IN      AL, DX              ; Get VGA status
    AND     AL, VERT_RETRACE    ; Vertical Retrace Start?
    JZ      @SW_WAIT1           ; If Not, wait for it
    ; Now Set the Horizontal Pixel Pan values
    OUT_8   ATTRIB_Ctrl, PIXEL_PAN_REG  ; Select HPP Reg
    MOV     AX, [BP].SW_Xpos    ; Get Desired X Offset
    AND     AL, 03              ; Get # of Pixels to Pan
    SHL     AL, 1               ; Shift for 256 Color Mode
    OUT     DX, AL              ; Fine tune the display!
@SW_Exit:
    POP     BP                  ; Restore Saved Registers
    RET     4                   ; Exit and Clean up Stack
SET_WINDOW       ENDP
;==========================================================;
 void SET_TEXT_MODE (void)
; int  SCAN_KEYBOARD (void)
;
; Routines for the Demo program, MODEX.C
; SET_TEXT_MODE - Sets the VGA to MODE 3 (80 col text)
; SCAN_KEYBOARD - Gets a Key from BIOS
        PUBLIC  SET_TEXT_MODE, SCAN_KEYBOARD

SET_TEXT_MODE   PROC    FAR
    MOV     AH, 0               ; 0 = Set Mode Function
    MOV     AL, 3               ; Mode to Set = 3
    INT     10h                 ; Call the VGA BIOS
    RET                         ; That´s it!
SET_TEXT_MODE   ENDP

SCAN_KEYBOARD   PROC    FAR
    MOV     AH, 00H             ; Function #0= Read key
    INT     16H                 ; Call Keyboard BIOS
    RET                         ; That´s it!
SCAN_KEYBOARD   ENDP
    end
```

Back

MTV Meets CD-ROM!

With the interface of Peter Gabriel's *Xplora 1* CD-ROM, you can interact, watch, resume, or quit your play. The color stripe on the side is the signature of Real World Records.

If you are part of the MTV generation, you are accustomed to playing Nintendo or Sega and alternately listening to music or watching music videos. It's not surprising, then, that many musicians and game developers are joining forces and flocking to the CD-ROM playground—rushing to get a corner of the market.

Old cartridge games are like new CD-ROM games; you see the landscape of Marioland much like you see the island of Myst. Likewise, popular two-dimensional action games parallel new interactive music games; you hear bullets and punches much like you play with an interactive mixing board. Games aim for a multisensory impact; interactive music CD-ROMs feature puzzles and challenges for players. In both, you feel you are a part of the artificial environment. You leave your desktop behind and enter a world of vivid imagination and carefully crafted creation to envision another three-dimensional realm.

A gaming element can bring music to life and entice people to experience music in a new, visual way. The music CD-ROMs available today enable players to interact with their favorite artists' clips and turn the pages of a musical memorabilia scrap book.

Interactive music CD-ROMs are difficult to create, design, program, and develop. Multimedia means there are a lot of things a game maker must address, polish, and then integrate together—and, like most development projects, it all comes together under tight deadlines.

Two small production houses created particularly successful music CD-ROM products: Brilliant Media in San Francisco, Calif., developers of Peter Gabriel's *Xplora 1*, and Graphix Zone in Irvine, Calif., developers of Prince's (☥) *Interactive*. Let's take a look at these two game-like CD-ROMs.

Gabriel's *Xplora 1*

When you load *Xplora 1*, you hear world music—mystical flutes in snake-charmer tones. Then you see a suitcase open; Peter Gabriel emerges from its depths to give you an overview of the landscape and explain the controls you will need on your exploration. He generously gives you your own suitcase with which to collect goodies along the way. (If you fill your suitcase, you earn icons and screen savers.)

You remember a suitcase motif from his recent tour and realize this is no flat, dull video. The screen seems alive. Along the top of the screen are pictures you recognize from his *US* album—each piece of art represents one of the songs from the album and each is done by a different artist. You begin to feel very at home with this interface.

Along the side of the screen are icons that act as verbs—you control what happens. You can interact, watch, resume, or quit by pointing on the image and clicking. You click on pictures which correlate to songs, and you are transposed to another interface with a new menu. For example, you can access the song "Digging in the Dirt" by clicking on the appropriate icon—a painting of a shovel—and get

by Diane Anderson

Using multimedia technology and game elements, two gutsy and innovative development houses wooed music's biggest stars— and gave the MTV generation a new medium to rock to.

a bio on the artist who created the painting. You can watch the video, hear from the team who worked on the video, or watch an interview with Peter (by this time you are on a first-name basis with the man) who offers his reflections on the particular song.

⚥'s *Interactive*

Imagine playing Myst. Only it's star-trekky. And purple. Very purple. Welcome to ⚥'s *Interactive*. While Gabriel's game is more of an "interface game," ⚥'s *Interactive* is more of a "place game," in which the player has the feeling of being placed in a futuristic environment. Instead of watching actors act, the player acts and plays. You use the cursor to motor around. The cursor is, of course, a symbol like this: ⚥. It is purple and zipple-like in the Macintosh version, white and rotating in the PC version.

You can turn, go forward, up, down, and all around. Instead of being presented with options, you are presented with a mystery, and you must discover the solution on your own. It's very three-dimensional. Instead of a static feel, *Interactive* has motion and a geographical dimension—a Myst-like environment with rich three-dimensional textures. You feel like you are on a voyage, a discovery.

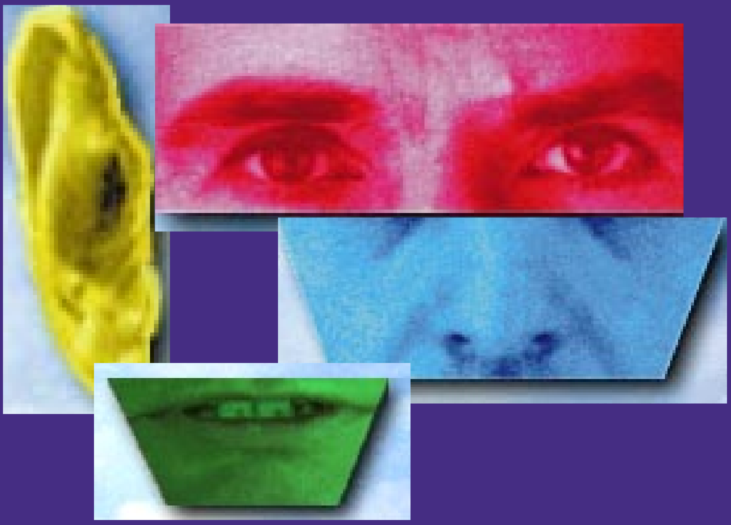
Peter's People

Despite Peter Gabriel's acute interest in computer technologies, he didn't have his own CD-ROM project until a smart, plucky developer came along. Meet Steve Nelson who, while at Claris (then a division of Apple) in Santa Clara, Calif., developed an object-oriented authoring tool for in-house users called Digital Montage. After working with Apple in 1991, he realized the entertainment field was ripe and ready to harvest. With his licensing fee from Apple for Digital Montage, Nelson founded Brilliant Media.

His first project was a CD-ROM demo, which combined elements from Gabriel's *So* album with available art and text. Nelson had never met Gabriel, but he managed to show him the demo—and he loved it.

Gabriel, who has always been on the proverbial cutting edge, was definitely ready for something interactive. His concerts, his killer videos, his philanthropic benefits are all major productions where lights, sound, and action come together for an in-your-face and blow-your-mind sensory overload. Remember the music videos for his hits "Sledgehammer" and "Big Time"? (Reportedly, some unfortunate epileptic in Boston suffered a seizure from one of his videos. I'd call that using a medium to touch your audience.)

Gabriel's involvement gave Brilliant Media access to a world of resources, such as Real World (which films many of Gabriel's concerts and videos) and *The Box* magazine, another British-based company closely associated with Gabriel. Such a consortium let Brilliant Media use much of Gabriel's existing media—song lyrics, interviews, images, movies, and graphics, as well as the several paintings Gabriel had commissioned by individual artists to capture the essence of



Welcome to Gabriel's secret world. Each interface element in *Xplora 1* is a doorway to a different experience. The ear takes you to Gabriel's World of Music and Dance, the eyes let you into his personal file, the nose takes you behind the scenes to a recording studio and the Grammy's, the mouth lets you enter the *US* database, which includes music, videos, artwork, and interviews.

each song on the *US* album. These paintings are what add the unique visual element to the audio and turn each song into a multimedia experience. James Johnson, who started as an intern at Brilliant Media and soon became the company's digital production supervisor, jokes that even if Brilliant Media were to have produced a "crappy interface," the CD-ROM still would have been a success because of Gabriel's incredible material. As it turned out, the interface received accolades and awards, including the Interactive Media Festival's Sparky Award.

A major part of the collective design effort was the group of independent artists and contractors who were hired for a time to scan, color, and clean up images. Troy Daniels, a Bay Area graphic artist, was hired to create the game screen. He remembers his time on the project. "Gabriel was very cool to meet and working with him made the experience unforgettable. He inspired me. We all wanted to finish with a quality product."

Gabriel not only incited enthusiasm and generated the music, video, and art, he oversaw the entire project. Nelson met often with Gabriel while he was on tour to show him Brilliant Media's progress. For the most part, Gabriel approved of Nelson's work—he just added creative elements to the production. Gabriel's unbelievable breadth of talent and interests, including WOMAD (World Of Music And Dance), Amnesty International, and Witness (a project that equips those who suffer human rights violations with cameras and other technology to expose those violations) were included in the game. Thus, *Xplora 1* is not just a rock video but a close-up of the artist and his passions.

⚥'s People

The concept for *Interactive* was initiated by Prince—excuse me, ⚥—but the design, programming, and porting of the game were the work of Graphix Zone.

In contrast to Gabriel, ⚥, and his production company, Paisley Park, were looking for developers. "Prince got the bug to be an interactive artist who wanted to express himself in a product like this," explains Dave Nichols, project director. ⚥ and Paisley Park put the word out that they were looking for developers to propose ideas and show presentations for possible approaches. Nichols and others at Graphix Zone worked hard to show Paisley Park what they could do. After looking at Graphix Zone's stuff, Paisley Park decided to go with them for the project.

Stop! In the name of...Prince (or rather, ⚥). In ⚥'s *Interactive*, the artist formerly known as Prince is the master of ceremonies, and players must solve puzzles to progress through the game's purple, interactive maze.

SGI KICKS BUTT

So, you want your game to have killer graphics. You've settled on something interactive, but aren't sure what the best development route is. It looks like this will be a CD-ROM project, and there is the prospect of making it an entertainment title. Where should you begin? Well, if all were for the best in this best of all possible worlds, you'd have the money to invest in one of the Indy systems from Silicon Graphics (SGI) and take advantage of the UNIX multiuser operating system. Then, you'd have a gigabyte of hard disk space and multitasking would be painless. So would porting to Macintosh and PC platforms. However, if this is on your wish list, make sure you have at least $5,000 to $25,000 for hardware alone and another $50,000 for necessary software: Alias's Eclipse ($5,995), Mentalix's PixelFX ($1,600), Visigenic's Creative License 1.1 ($1,495), and Barco Graphics's Creator 5.2 ($17,000) and Strike ($18,000). Granted, applications will run very quickly, but for that price they better.

Wake up, you were dreaming. Unfortunately, you don't have the money to buy such extravagant dreams. And since CD-I and 3DO are not yet the standard, you've decided to stick to more traditional (and affordable) means. The PC looks like a good pick, but you remind yourself you want graphics and good ones. You want a three-dimensional world where sound and video combine to anesthetize the player. For your project, budget, and goals, Macintosh is the choice.

Surprisingly, some great games were developed on the Macintosh this year. *Xplora 1* and *Interactive* are not isolated anomalies. They aren't the only great Macintosh-developed games. Myst is a perfect example. But there is stiff competition from DOS and PC game makers. Doom has shared more wares than anyone would care to count. And Lucas's amazingly lucrative product Rebel Assault was developed on the PC. It sold 500,000 individual copies by the summer of 1994—not even a year after its late November 1993 release. Myst sold 200,000 copies by April 1994—but that is way before its PC version even had time to pick up any steam. Give Myst some time selling to both Macintosh and PC consumers, and I'm sure its commercial success will be overwhelming. Despite its bugs, the Macintosh-made Myst is a multisensory, virtual-reality envelope pusher.

However, this is not a comparison or contrast of games, but an examination of developing on the Macintosh. And I must agree with Wayne Sikes that "the three-dimensional animation graphics in Rebel Assault are excellent" ("Breaching the Rebel Base," Chopping Block, Sept. 1994). However, the relatively affordable graphics capabilities of Macintoshes have enabled creators of games to realize the aesthetic environments of their imaginations. The pixels on a Macintosh screen are what make the difference. Even if PCs are more tenable, the presentation on Macintosh is superior. Programming on PCs is unequivocally smoother, but visual aspects of a game are crucial as well.

Macintoshes have simply been the best three-dimensional performers. Who else has killer applications like QuickTime, Photoshop, Illustrator, Electric Image, and Strata? Now programs like Live Picture by HSC Software let users work with really large files with FITS (Functional Interpolating Transformation System) technology.

Many people involved in such visual creation are artists (see David Siek's "Artist vs. Artisan" in Artist's View, p. 63), not programmers; the Macintosh is more user-friendly to such people. Of course, many computer novices get involved in the production of such titles. Musicians, animators, and marketers are all important players.

Since Apple infused the market with so many CD-ROM drives and Adobe first introduced its graphics applications to the Macintosh community, it seems logical that Macintoshes would be a natural place for CD-ROM developers to congregate. Competition from IBM with its new Holiday '94 multimedia computers is good for the industry.

Nichols explains that after the initial hand-off from Paisley, Graphix Zone took over all the design and production of the project—they were the creative brains and the technical muscle. Of course, Graphix Zone showed Paisley the stages of progression and got feedback. It was a collaborative effort. Paisley supplied information, music tracks, and other material. Since ⚥ is media-oriented, the "talent" in this case was very willing to cooperate. ⚥ recorded a song and shot a video specifically for this project—something many developers hope artists will continue to do in this medium, instead of simply recycling old stuff.

Obviously, *Interactive*'s design needed to match ⚥'s persona—and it does. You sense ⚥ throughout the experience. The game is seductive and mysterious—and purple.

The design and product were fundamentally done at Graphix Zone, after the initial hand-off from Paisley. (Money makes things happen!) Graphix Zone completed the job in six months—just in time for ⚥'s birthday on June 7.

Peter's Machines

The video "Kiss that Frog" on the *Xplora 1* CD-ROM lucidly illustrates that Gabriel and the Brilliant Media team had the help of human and technological resources. Mindblender Rock Motion Theatre, a "ride" featuring quadraphonic sound and hydraulic seats, joined forces for the video aspect with video director Brett Leonard (*Lawnmower Man*). A.E. Bunker's sketches were scanned into Silicon Graphics workstations and three-dimensionally animated at Angel Studios in Los Angeles. Leonard preconceived ideas on the computer using three-dimensional wire frames. They shot footage with Sony's HDTV camera.

Viewers can reinterpret and reexperience "Kiss that Frog" over and over again. It is a powerful blending of psychedelic graphics, organic textures, and primitive emotion. The video makes viewers feel as if they are moving. Other videos on *Xplora 1* don't have motion, but do contain morphs and other tricks.

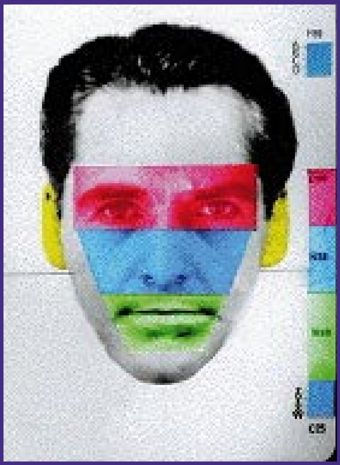
Although Brilliant Media didn't shoot the videos or write the songs, the company assembled them to create a cohesive and rich product. Programming this material into a tapestry required patience and skill. To put the whole thing together and add the interactive element, Brilliant Media used its proprietary authoring system, Digital Montage, a program based on HyperCard and developed by Nelson. Think of Digital Montage as Quark or Pagemaker for multiple media. It lets the user lay out various digital information—text, images, video—into a seamless song. It allows multiple users to work on a file and coordinates "asset management" for a whole project.

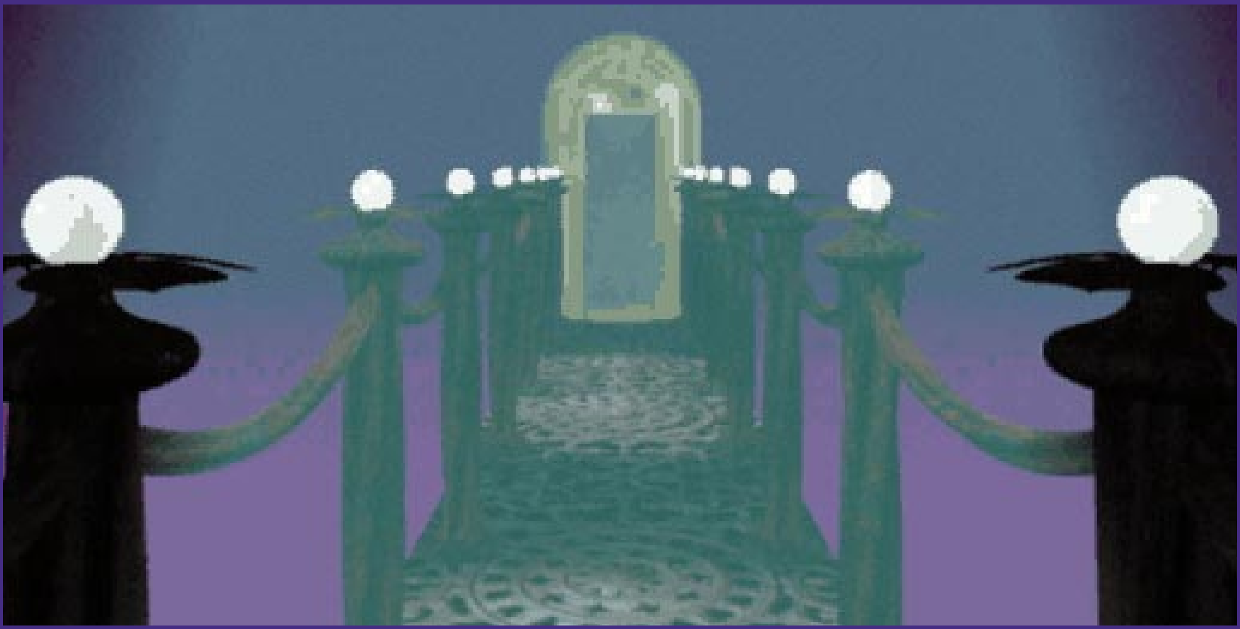
Because Digital Montage is based on Hypercard, it provided extensibility. For the trippy morphs and sleights of hand, Hypercard let Brilliant Media control the architecture of the game. We all know multimedia is about communication—not just between people but between programs. Digital Montage took advantage of Hypercard's capabilities. Other important tools Brilliant Media used included:

- Digital Film Board by SuperMac (to digitize video)
- VideoFusion by VideoLake (for creating effects)
- Photoshop by Adobe (to create and manipulate PICT, TIFF, and EPS files)
- Premiere by Adobe (to digitize and edit digital video)
- AfterEffects by CoSa
- QuickTime and MovieShop by Apple
- SoundDesigner by Digidesign
- Video Vision and PLI
- A 1.07GB turbo drives.

These products helped Nelson and his crew tackle the most Herculean task of all—the particular challenge of dealing with video footage. In England, the video standard is PAL; in the U.S., it is NTSE, which made for some difficulty. However, James Johnson managed to be a competent liaison; he coordinated communications

Prince represents himself with this symbol—which is seen throughout his *Interactive* CD-ROM.


Choosing from diverse facial features, players of *Xplora 1* must put together Peter Gabriel's passport photo.


Cross the bridge and enter another realm. The amazing three-dimensional rendering of *Interactive* lets you get inside your computer.

Watch an image magically change before your very eyes. Morphs for *Interactive* were done in Elastic Reality.

between Real World and Brilliant Media and managed the vast sound files, photo PICTS, and video clips involved. It was imperative these resources were not damaged, destroyed, or lost, and so archiving was vital. Not only did Johnson get an education in management; he learned a lot about technology. Fortunately, Digital Montage allows for easy tracking of changes—an important benefit for such collaborative projects.

⚥'s Machines

From a creative and technical standpoint, Nichols says the people at Graphix Zone feel pleased with their work on *Interactive*. Design and programming aspects were nearly painless with their authoring tool, Apple Media Tool. They needed Apple Media Tool to create the main part of the program and then extend into other areas, something possible from other programs like Hypercard. However, other script languages aren't as powerful as C is and will only allow you to do limited things "in addition to y'know making a movie player, being able to turn left or right or whatever," says Nichols.

The first part of the Apple Media Tool works like Director—only simpler. You sacrifice some control but you get the same mapping or staging. The second part of the tool is a variation of the C programming language called Apple Media Programming Language. This is a custom language that allowed the team to design what it wanted to as far as navigation, graphics, and the integration of audio and video in a real simple interface. Because the designers wanted to do custom things like adding a mixer (a DJ booth where you can select CDs and play them in a player), the team used Apple Media Kit, which enabled them to code in their own features when they wanted to do something above and beyond Apple Media Tool.

Other code-intensive projects required programming—a puzzle in the virtual video room, morphs in the church, and so on. Morphing was done in Elastic Reality. Nichols claims Apple Media made it much easier on the development team.

The Pain of Porting

Porting is perhaps the biggest challenge developers face today, and one they must face if they want to cater to a wide audience. Many developers use Macromedia's Director, which enables developers to port their games to both PC and Macintosh platforms.

Digital Montage enabled the Brilliant Media team to complete the project in nine months, but it didn't provide a straight PC link. Brilliant Media released *Xplora 1* for the Macintosh only and went to an outside company in England to provide the CD-I and PC ports. At press time, a PC version of *Xplora 1* was about to be released.

Graphix Zone's Apple Media Tool compiles for Macintosh and Windows platforms, which reduced their porting worries. Not only did *Interactive* ship on time (thanks to the tool), but it was able to run on both platforms.

What's Next?

Big names like Prince and Peter Gabriel mean big bucks for developers who can create great games that work. Brilliant Media is now working on other multimedia projects for Disney and Warner Bros., and a CD-ROM movie project featuring Richard Dreyfuss. The exposure Brilliant Media received for its pioneering work on the collaborative *Xplora 1* title couldn't have hurt its chances of winning such bids.

Graphix Zone's next project is for Bob Dylan—quite a change from ⚥. Nichols says that coordinating with talent is challenging, but his real interest is getting inside the minds of stars and figuring out their styles. Where does Nichols see the future? "I want to try to continue to do that same thing —extend the use of both technology and the ability to communicate something that represents the artist accurately and in an entertaining way."

The egomaniacal ⚥ is at it again. This time he's back with a male-female-arrow-mirror-symbol-deal. Graphix Zone made the new symbol its navigating cursor. I'd call that an accurate representation—and an entertaining one.

Modern Jukebox. Click on an icon, get a song. *Xplora 1* pleases your eyes, your ears, and your mind.

CD-ROMs this year. Conversely, Doom and Mortal Kombat will be starring in movie theaters near you soon. Television commercials are more cyber-aware these days. Technology (represented by the image of a laptop screen), coupled with an icon of counterculture, is used to advertise tennis shoes of all things. Intel Inside pops up between Letterman skits. Conversely, one of our favorite Saturday Night Live personalities is featured in a commercial advocating the CD-I platform. Strange, but big-name designers can be found not only on the pages of fashion magazines, but on the pages of our favorite techno-savvy, slick, and oh-so-very-90s 'zines.

This proves a very symbiotic relationship—entertainment need be technical and technology is entertaining in and of itself. ■

Break on Thru to the Other Side

Not only have multimedia merged, but industries have as well. Making transcendent computer experiences is not easy. The aim of games and the prospect of virtual reality promise the possibility of transporting a person sitting in front of a computer into another environment. Players should forget they are using a computer and be unaware of the mouse or keyboard as they attempt to grasp another dimension.

For so long, stories have helped people enter new worlds, and music has aided in soothing the stress of life by connecting us with a spiritual aspect of sound. Now interactive CD-ROMs offer us an illusion of control. Interactive multimedia is cool. A lot of things are multimedia—meals, poetry readings, concerts, movies—but the interactive element is what intrigues our human nature.

Hollywood, Silicon Valley, Siliwood. It looks like information and entertainment are coming together more and more. Technology didn't only make the MTV generation, it is now responsible for Kurt Cameron, Tia Carrere, Dennis Hopper, and Stephanie Seymour showing up in

Diane Anderson is the editorial assistant at Game Developer.

PORTING

M*acromedia User Journal* #27 gives this advice: "To distribute Director 4 movies to Windows users, you need to create a projector using the Windows version of Director 4. To distribute Director 4 movies to Macintosh users, you need to create a projector using the Macintosh version of Director 4. In both cases, the projector file can then call the same Director 4 movie files. The only restriction on movie files is that they are named in accordance with PC naming conventions." Another word on Director: Since Microsoft seems to enjoy seeing its Windy City operating system repeatedly renamed (Windows 4 or Windows 95), there has not been OLE 2.0 support for Director's Windows version. But Macromedia has good news for gamers, it looks like OLE 2.0 support will happen if Microsoft ever comes through.

Beware of differences in color—a display on Windows has lower color depth, and so colors don't look the same. Some VGAs can show 16 colors and some can show 256. Making a movie look the same on all platforms is no easy task. Sound is also better on a Macintosh. These individual aesthetic elements (mediums) are the individual parts that, when assembled together, make the *multi* the operative part of multimedia.

Macromedia's Director is a good tool to use if you are concerned with making a game that runs on both PCs and Macintoshes. Neither platform has complete dominance (especially with all these strange, rumored mergers and buyouts), so it seems smart to put your game out as many ways as possible. Visibility means marketability for your company—competition is the name of this multimedia game we're playing.

Back

Bandits at 0x1200 High!

Pacific Strike sends the player—acting as fighter pilot—on battle missions based on real-life occurrences (Pearl Harbor, Midway, and the like). Here, a Japanese Val plane has just been targeted.

Up next on the chopping block is Pacific Strike by Origin Systems Inc. Pacific Strike is an action-oriented aircraft simulation based in the Pacific theater during World War II and was written using Origin's RealSpace Technology. Although this review centers on Pacific Strike v. F1.19, I included the speech pack (sold separately) in my analysis.

Once installed, Pacific Strike uses about 20MB of hard disk space and requires 583K of conventional memory as well as at least 4MB of XMS (extended) or EMS (expanded) memory. Ideally, the game wants expanded memory, so the JEMM.OVL expanded memory manager, included with Pacific Strike, will automatically run whenever the user's computer only has XMS memory available. (Origin's JEMM.OVL was written by Jason Yenawine—JEMM is probably a mnemonic for "Jason's Expanded Memory Manager.") If you have trouble running the game, the included OSITEST.EXE routine, a carryover from Strike Commander, will diagnose your system and make recommendations for running Pacific Strike. (I saw no mention of this routine in the game literature.)

How it Works

Pacific Strike's primary executable, PACIFIC.EXE, is about 1MB in size and is not compressed or encrypted. It was written using the Borland C++ Development System. This file contains essentially all the executable code in the game. My analysis showed the presence of a built-in cheat mode (although I haven't yet figured out how to access it), plus lots of internal diagnostics. PACIFIC.EXE requires a minimum of a 80386 processor and runs the game in protected mode. Comparisons of PACIFIC.EXE and the INSTALL.EXE file show that the developers made provisions for two add-on mission packages. Of particular interest was my discovery of embedded PKWARE data compression functions. (I later discovered that most of the graphic terrain data was stored in ZIP format, so the presence of a "pkunzip" function makes sense.)

Essentially all game data is contained in the PACIFIC.DAT file. This large file (about 16MB in size) contains 600 data files. I will refer to these embedded data files as File Records. Listing 1 gives a

by Wayne Sikes

What makes the best games fly? Flexible data structures. Wayne Sikes presents a navigation plan to steer you through the ins, outs, ups, and downs of Pacific Strike.

general summary of how PACIFIC.DAT is organized. Basically, you can break PACIFIC.DAT into three parts:

- An 8-byte header
- A list of 74-byte File Description Records.
- A list of File Records.

The header gives the total number of File Records and the file offset of the first File Record (see the `PACIFICDATHEADER` example in Listing 1). Following the header are File Description Records. Each 74-byte File Description Record contains information about a File Record, such as the path name of the original data file making up the File Record, where the File Record is located in PACIFIC.DAT (expressed as an offset from the top of the file), and the size (in bytes) of the File Record. In other words, each File Description Record "points" to its corresponding File Record—as in the `FILEDESCRIPTIONRECORD` structure in Listing 1. (In this review, I will frequently refer to File Records using their original data file name. For example, the Corsair aircraft data is contained in the CORSAIR.IFF File Record.)

The File Records constitute most of the space in PACIFIC.DAT. These records vary in size from a few to several hundred thousand bytes and contain the actual "pieces" of game data. Many different types of File Records are stored in PACIFIC.DAT. Most File Records have an IFF file name suffix; out of 600 File Records, 511 are of the IFF type.

What is an IFF File Record?

The IFF file name suffix is possibly a mnemonic for "Information Form File." An IFF File Record is a structured, variable-length file whose design allows for an open-ended, expandable format. (Origin Systems has used the IFF File Record format in several games including Privateer and Strike Commander. My PREDIT, PREASY, and SCEASY "help" utilities modify various IFF File Records in these games.) Basically, an IFF File Record is made up of one or more `form`s, with each `form` containing one or more `record`s. Following are a few general rules for IFF File Record construction.

- All `form`s have a header consisting of the "`form`" text string followed by a 4-byte number. This number is the number of data bytes in the `form`.
- All `record`s have a header consisting of a name (that can be up to 8 bytes of text characters) followed by a 4-byte number. This number is the number of data bytes in the `record`.
- `Record`s can be located both inside and outside a `form`.
- *Never* change the `form` or `record` header name, and *never* change the total number of data bytes in these structures. Violation of these rules will most likely cause a system crash. (Actually, you can change the number of `record` data bytes but be *very* careful when restructuring any `form`s that may enclose the `record`.)

I extracted most mission and aircraft data discussed in this article from IFF File Records. The next two sections discuss how missions and aircraft are "built" by the game engine, using the IFF File Records.

Mission Anatomy

Essentially all mission data is stored in IFF File Records. The name of a mission

File Record contains the geographic location of the mission and the number of the mission at that location. When the game engine chooses a mission to play, it opens and reads the IFF File Record corresponding to the mission. The first campaign mission is centered around the Pearl Harbor area. The mission data is located in IFF file PRLH-M1.IFF (PRLH-M1 is a mnemonic for Pearl Harbor Mission 1). The game engine first scans the File Description Records in PACIFIC.DAT until it finds a reference to the PRLH-M1.IFF File Record. Once the correct File Description Record is located, the offset and size data are used to locate and read the PRLH-M1.IFF File Record.

The mission IFF File Record contains several internal `record`s describing the various mission parameters. The mission structure is somewhat complex, so I will only describe a few of the more interesting `record`s.

The `WRLDFILE record` gives the name of the IFF File Record containing the terrain data for the player's location. The primary geographical areas the player will interact with are in the `AREA record`. This record contains the name and XYZ-coordinate data of each important location—for example, airfields, ships, enemy encampments, navigation waypoints, and the like.

The game's three-dimensional XYZ-coordinate system is interesting. Pacific Strike references all world objects using a 24-bit coordinate system. If you imagine the world (or an area of it) displayed as a flat map, movement in the X-axis direction goes east-west with positive (+X) movement going east. Movement in the Y direction goes north-south and positive (+Y) movement heads north. The Z-axis goes above-below the map and can be considered as altitude referenced to sea level. Positive (+Z) movement means increased altitude. Using cockpit altitude measurements and data from Area Records, I calculated a map scale factor of 1 XYZ unit equals 1 meter. I used this scale factor to calculate the size of the Pacific Strike "world." This world measures about 10,500 miles on a side for a total of about 110 million square miles. The center of this world, at XYZ coordinates (0, 0, 0) is the center of the terrain map currently being used. An example C structure for the `AREA record` data is given in Listing 2.

The `CAST record` lists the names of the IFF File Records containing dialog box text and other information used by the cast members. Each cast member has a Cast Member Number. Rather than referencing cast members by name, the game engine references each cast member using a predefined Cast Member Number.

Each cast member plays a part in the mission, and the `PART record` defines each member's role. See Listing 2 for an example C structure for `PART record` data. Each cast member has a data structure in the `PART record`. The cast member structure references the cast member by its Cast Member Number and contains IFF File Record names for files describing the member and any associated weapon loads as well as XYZ-coordinate data giving the starting location of the cast member relative to the `AREA` location in which this member belongs. Finally, this structure contains various control bytes specifying the actions of the member.

Aircraft Construction

Aircraft data is stored in IFF File Records. These records generally have a file name similar to the type of aircraft being described, for example, the F4F3 Wildcat data is stored in the F4F3.IFF File Record. The aircraft IFF File Record contains `record`s describing various aircraft parameters. Because the overall aircraft description is rather complex, I will again only discuss a few of the more interesting `record`s.

The `JETPHNME record` (mnemonic for "JET Pilot NaME") is interesting because

Listing 1. The PACIFIC.DAT File Structure

```
HEADER
Bytes 0-3    Number of embedded File Records.
Bytes 4-7    Offset of the first File Record.

FILE DESCRIPTION RECORDS
Bytes 8-XX   File Description Records. Each record is 74
             bytes long.  XX = 7 + (74 * Number of File Records)

FILE RECORDS
Bytes (XX+1)-EOF    File Records.



EXAMPLE HEADER AND FILE DESCRIPTION RECORD C STRUCTURES
#define   BYTE   unsigned char
struct PACIFICDATHEADER
   {
   long   NumFileRecords;           // offsets 0-3
   long   FirstRecordOffset;        // offsets 4-7
   };

struct FILEDESCRIPTIONRECORD
   {
   BYTE   Unidentified1;            // offset 0, always value of 1
   char   OriginalPathFileName[32]; // offsets 1-32
   BYTE   Unidentified2[33];        // offsets 33-65
   long   FileRecordOffset;         // offsets 66-69, PACIFIC.DAT
                                    // FILE RECORD OFFSET
   long   FileRecordSize;           // offsets 70-73, FILE RECORD
                                    // SIZE
   };
```

Pacific Strike's dynamic animation capabilities depicts a shot of several aircraft sitting on a carrier. This image appears in the games opening animation sequence.

it shows that much of the Pacific Strike game engine came directly from Strike Commander. After all, why would a World-War-II aircraft name be in a record referencing jet aircraft?

The flight modeling is contained in the TOFF, LAND, DYNMDYNM, STBL, ATMO, THRS, and JDYN records. The DYNMDYNM record (see Listing 2 for an example C structure) contains the approximate weight of the aircraft in increments of 600 pounds. For example, the weight of a 4,200-pound aircraft would be given as 7. Obviously in this flight model, the exact weight is not very important. The STBL record contains a data byte representing the overall stability of the aircraft. These values ranged from an unstable Helldiver (40 decimal) to a very stable Bearcat (131 decimal). If you reduce the value of this parameter to less than 20, your aircraft will never be stable enough to fly. Increase this value to get better handling.

The ATMO record appears to contain a measure of the overall atmospheric drag or total air resistance of the aircraft. The Baka (a very sleek, rocket-powered Japanese aircraft) had an ATMO record data-byte value of 51 (decimal), and the large and heavy Japanese Betty had a value of 230 (decimal). If the ATMO record data value is too high, your aircraft will have so much drag that you will never get off the ground. Conversely, very small values will give you almost no air resistance—that is, they will create inertia similar to flying in space vacuum conditions. It will take you a long time to bleed off enough speed for landings.

The THRS record contains aircraft thrust and propulsion data. Refer to Listing 2, which gives an example C structure for THRS record data. Byte offsets 5 and 6 in this structure form a 16-bit value that represents aircraft engine horsepower. This horsepower value sometimes differs from the values referenced in the game literature (probably due to programmer "tweaking" to make the game more playable). Of course, changes in engine horsepower ratings will drastically affect aircraft performance. Drag factors are located at byte offsets 12 and 16. The drag factor at byte offset 12 appears to be used in calculating the amount of lift required for take-off, while the value at offset 16 is a drag element used for calculation of stall speed. Increasing the value at offset 12 results in higher takeoff speeds, and increasing the value at offset 16 results in higher stall speeds. Generally, the two drag factors have almost identical values.

The WPNS record data consists of one or more 10-byte structures. Each structure references one weapon type. Listing 2 gives an example WPNS structure. The first two bytes in the structure contain the weapon load (number of bullets, bombs, and the like), and the last eight bytes give the name of the IFF File Record that contains detailed weapon information. The names of the weapon File Records correspond with the weapon type, for example, 20MM.IFF (20-mm cannon), 50CAL.IFF (.50-caliber machine gun), and B1000LB.IFF (1,000-pound bomb).

The HPTS record data defines where the weapons referenced in the WPNS record are mounted on the aircraft. See Listing 2 for an example of the `HPTS` data structure. Each mount point on the aircraft is called a "hardpoint" and is defined by a 13-byte structure. One or more of these structures make up the `HPTS record` data. The first byte in the structure gives the hardpoint type. Surveys found the following hardpoint types:

- *Type 0*: .50-caliber machine guns, 20mm cannons, and 7.7mm machine guns
- *Type 1*: High-Velocity Aerial Rockets
- *Type 2*: 100- and 551-pound bombs
- *Type 3*: 500- and 1,000-pound bombs, Napalm, MK-13 Torpedoes
- *Type 4*: 12.7mm machine guns

The last 12 bytes in each `HPTS record` data structure give the location of the hardpoint expressed in aircraft XYZ coordinates. Aircraft coordinates are represented as 32-bit signed numbers. The X-axis stretches from wingtip to wingtip, the Y-axis stretches from the nose of the plane to the tail, and the Z-axis extends from the top to the bottom of the aircraft. Origin's aircraft coordinate system follows the standard "right-hand rule" (for those of you who remember a little bit of physics). The 0 or center reference position (0, 0, 0) is approximately the location of the pilot seat in the cockpit. Using references to aircraft dimensions given in the game literature and various `HPTS record` data samples, I calculated a rough XYZ-coordinate scale factor of 12 XYZ coordinate units equals 1 inch of distance on the aircraft or 144 XYZ coordinate units equals 1 foot. For example, assume you are sitting in the cockpit of a Corsair and you increase the X-axis coordinate of a gun mount hardpoint from 400 to 544. Looking out the right side cockpit window onto the wing, you would notice the gun mount had moved about one foot away from you.

Undocumented Keyboard Features

Several useful keystrokes are not documented in Pacific Strike's game literature. Note that some of these are active only in certain portions of the game. During game startup and while onboard the carrier, the Alt-V keystroke reads out the game version and the state number. The game

Listing 2. Several Types of RECORD Data (Continued on p. 59)

```
#define      BYTE        unsigned char
#define      WORD        unsigned int

// Data in AREA Record
// Note there are Type C and S AREA RECORD data. The type is determined
// by the contents of AreaType.  The Type C and S data are different sizes.
struct AREA_TypeC
   {
   BYTE   AreaType;          // offset 0, C char
   char   AreaName[33];      // off 1-33, text or 0x2E
   long   XAxis;             // off 34-37, X pos of object
   long   YAxis;             // off 38-41, Y pos of object
   long   ZAxis;             // off 42-45, Z pos of object
   WORD   AreaWidth;         // off 46-47
   WORD   Blank0;            // off 48-49
   WORD   AreaHeight;        // off 50-51
   BYTE   Blank1;            // off 52
   };

struct AREA_TypeS
   {
   BYTE   AreaType;          // offset 0, S char
   char   AreaName[33];      // off 1-33, text or 0x2E
   long   XAxis;             // off 34-37, X pos of object
   long   YAxis;             // off 38-41, Y pos of object
   long   ZAxis;             // off 42-45, Z pos of object
   WORD   AreaWidth;         // off 46-47
   BYTE   Blank0;            // off 48
   };

// Data in DYNMDYNM Record
struct DYNMDYNM
   {
   BYTE   unknown0;          // off 0
   BYTE   unknown1;          // off 1
   BYTE   AirplaneWeight;    // off 2, real wt in lbs/600
   BYTE   unknown2;          // off 3
   };

// Data in HPTS Record
struct HPTS
   {
   BYTE   Type;              // off 0, Hardpoint type 0-4
   long   XAxis;             // off 1-4,  X pos of hardpoint
   long   YAxis;             // off 5-8,  Y pos of hardpoint
   long   ZAxis;             // off 9-12, Z pos of hardpoint
   };

// Data in PART Record
struct PART
   {
   WORD   MemberNumber;      // off 0-1, Cast Member Number
   char   MemberName[16];    // 2-17,  IFF File Record name
   char   WeaponLoad[8];     // 18-25, IFF File Record name
   WORD   Unknown0;          // 26-27
   WORD   Unknown1;          // 28-29
   long   XAxisRelative;     // 30-33, X pos rel to AREA
```

Listing 2. Several Types of RECORD Data (Continued from p.58)

```
    long   YAxisRelative;     // 34-37, Y pos rel to AREA
    WORD   ZAxisRelative;     // 38-39, Z pos rel to AREA
    BYTE   Controls[22];      // 40-61, various control bytes
    };

// Data in THRS Record
struct THRS
    {
    BYTE   Blank0[5];         // off 0-4
    WORD   HorsePower;        // off 5-6, Engine HP
    BYTE   Blank1;            // off 7
    BYTE   Flags0;            // off 8
    BYTE   Flags1;            // off 9
    BYTE   Unknown0;          // off 10
    BYTE   Blank2;            // off 11
    BYTE   Drag1Factor;       // off 12, 1st drag factor
    BYTE   Blank3[3] ;        // off 13-15
    BYTE   Drag2Factor;       // off 16, 2nd drag factor
    BYTE   Blank4[2];         // off 17-18
    };

// Data in WPNS Record
struct WPNS
    {
    WORD   WeaponLoad;        // Number of bullets, bombs, etc
    char   WeaponName[8];     // Weapon IFF File Record name
    };
```

engine references missions using predefined State Numbers. When you are in the cockpit during a mission, this same keystroke reads out the Version and the Mission name. The Mission name is essentially the name of the IFF File Record that contains the mission data. Pearl Harbor Mission 1 has an IFF File Record name of PRLH-M1.IFF and would be displayed with Alt-V as "PRLHM1."

The game also includes some built-in diagnostics you can activate via the Shift-D and Shift-F keycodes. Both keycodes are active only while the player is in the cockpit. The Shift-D keycode toggles a readout containing display and memory usage parameters, including frame rate (in frames per second). The Shift-F keycode toggles a readout of the display frame rate only. The Alt-F keycode displays the game score.

Experimenting with Pacific Strike

During the course of this analysis, I prepared a "Pacific Strike Experimenter's Kit" to experiment with game data. The kit includes PSDAT, an analysis and data modification tool you can use for extracting and inserting File Records into Pacific Strike. It also includes several X-Wing aircraft that I built using aircraft File Records. (I refer to these aircraft as X-Wings because I repositioned the gun mount hardpoints on the wingtips so that they appear to fire in an X-Wing manner.) These experimental X-Wing aircraft are much faster and have more firepower than the original game vehicles. The Experimenter's Kit (PSKIT.ZIP) is in the Flight Simulation Forum (GO FS FORUM) on CompuServe, Historic Air Combat Library.

Does It Fly?

One of the messages that Origin released regarding Pacific Strike was that the game tries "to raise the standard in terms of technology, graphics, sound and music—every aspect of the gaming experience." My analysis of Pacific Strike shows this to be the case, but as with all evolving technology, numerous problems exist. Overall, the RealSpace game engine worked well but it could still use some improvement in the graphics rendering and updating areas.

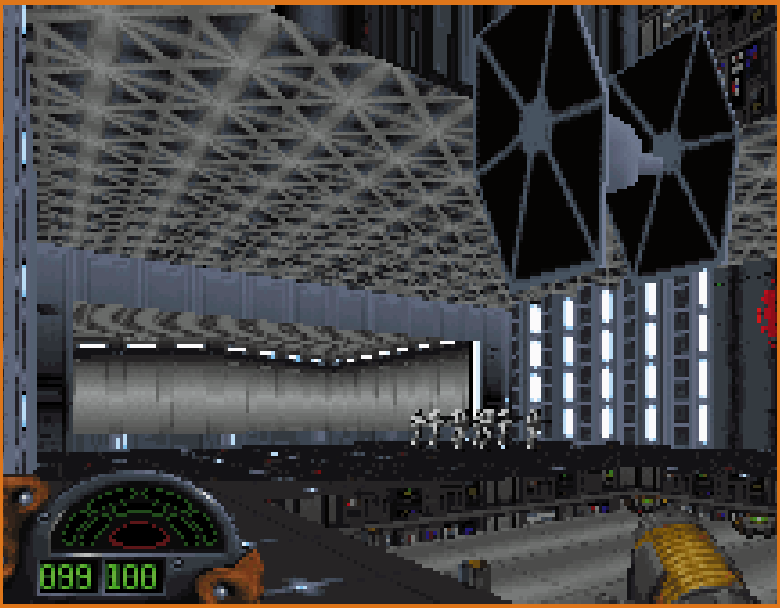
RealSpace was also used in Strike Commander, and I noticed a definite carryover of old Strike Commander data in Pacific Strike. Some Pacific Strike File Records appeared to be littered with data such as unused pieces of Strike Commander missions. While doing the analysis for my SCEASY and SCEDIT Strike Commander utilities, I observed that the Strike Commander data was very cleanly organized and grouped. Pacific Strike seems to be a product that was hurriedly thrown together using the Strike Commander engine. The presence of unused Strike Commander mission data is a good example of the "we are behind schedule and on a tight budget, so just make it run" philosophy many game companies currently operate with.

The philosophy behind the RealSpace file formats used for data storage, that is, IFF File Records, allows for an open-ended, expandable gaming environment. The price for this technology is longer time delays during game startup and updating (while the various File Records are being processed), and more stringent requirements regarding computer system speed and memory usage. (I turned off all the sky and water texturing options while doing this analysis and I often achieved a frame rate on my 486DX2/50 of only five frames per second.) I guess its about time for that 90MHz Pentium upgrade. ■

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 authored numerous computer gaming help utilities. He can be reached via e-mail at 70733.1562@compuserve.com or through Game Developer *magazine.*

Back

Reinventing the Death Star



The Dark Forces game engine is capable of displaying both one-dimensional rotated bitmaps and true three-dimensional polygon objects. In this case, the characters walking around on the deck are two-dimensional bitmaps while the spaceship and surrounding spacestation are polygon-based with bitmaps wrapped around them.

In an era where most movie companies are trying to get into the computer game industry, one company has been there for a while. LucasArts Entertainment and its sister company, Lucasfilm, both formed by *Star Wars* creator George Lucas, have been in the thick of both industries for almost a decade. While the companies are totally separate, they have found a creative synergy in LucasArts games based on Lucasfilm movies.

Although both George Lucas and LucasArts CEO Randy Komisar have publicly criticized the digital Hollywood movement, the close relationship between these two companies has helped LucasArts reap the benefits of George Lucas's moviemaking vision.

Using the Force

The influence of *Star Wars* can be felt throughout many computer and video games—from the hyperspace button in Asteroids to the space dogfights in Wing Commander. The *Star Wars* story first appeared in games in a series of coin-operated arcade games developed by Atari in the early 1980s, which featured sequences loosely based on scenes from the three *Star Wars* movies: *Star Wars*, *The Empire Strikes Back*, and *Return of the Jedi*. The first two were three-dimensional space combat games using vector-based, wire-frame graphics to show action. The third game, Return of the Jedi, used more conventional, sprite-based color graphics and featured game play from an isometric three-dimensional perspective.

To capitalize on the exploding video game market, in 1982 Lucasfilm formed a computer game division called Lucasfilm Games. From 1982-1987, acting solely as a developer, Lucasfilm Games turned out games such as Rescue at Fractalis and Ball Blazer. In 1987, Lucasfilm Games became a publisher, and in a corporate restructuring move eventually became what it is today, LucasArts Entertainment.

After developing coin-operated arcade games, LucasArts wanted to better establish itself in the game market and distance itself from the shadow of the Lucas empire. And so, the company began developing games for home computers that had nothing to do with *Star Wars*. Its first titles were the point-and-click adventure games

by Alexander Antoniades

LucasArts is keeping Darth Vader's name alive around the office by continuously coming up with new games using iconography from the Star Wars film series.

Maniac Mansion, Zak McKracken and the Alien Mindbenders, Loom, and the Secret of Monkey Island, most of which were created using its own SCUMM (Script Creation Utility for Maniac Mansion) game engine.

LucasArts next entered the flight simulator genre, with games such as Battlehawks 1942 and Secret Weapons of the Luftwaffe. It was from this experience making flight simulators that *Star Wars* reemerged in the computer game world.

A License to Develop

LucasArts' computer game division had first crack at the license to create games based on all of George Lucas's films. This relationship worked well on films such as *Indiana Jones and the Last Crusade,* which had multiple computer games released in conjunction with it. This partnership even produced Indiana Jones and the Fate of Atlantis, a computer game-only episode of the Indiana Jones saga. Unfortunately, by the time this licensing arrangement was established, the *Star Wars* film series had come to an end.

But Star Wars games returned in a big way in 1991, with the release of X-Wing, a game featuring *Star Wars* characters and themes. In X-Wing, players pilot an X-Wing fighter plane (similar to Luke Skywalker's in the original *Star Wars*) from training missions up to destroying the Death Star. The X-Wing design team integrated the repair and management of the X-Wing's resources into the game, which added to the game's realism and improved long-term playability. The combination of LucasArts' experience with flight simulator games, a game engine similar to Origin's Wing Commander series, and *Star Wars* iconography, made X-Wing a smash hit.

X-Wing was so successful that one supplemental mission disk made PC Data's best seller list by itself. The game eventually spawned a sequel called Tie Fighter.

After this success, LucasArts investigated making more games with the *Star Wars* license and the result was another hit, Rebel Assault. This game was more episodic and had more linear game play than X-Wing, and it relied more on captured video images, including some taken directly from the original *Star Wars* film. Rebel Assault was also among the first games to come out on CD-ROM for the PC and became an early success story for this product category.

Dark Forces

Continuing its trend of using the *Star Wars* license to showcase cutting-edge technology, LucasArts joined the ranks of Doom-style games with its latest entry, Dark Forces. In this game, the player fights storm troopers as a secret agent for the Rebel Alliance.

While the game's lead programmer, Daron Sinnett, admits his team studied Doom's design, he insists that the Dark Forces game engine is 100% original. He speculates, however, that the data structures might be close enough that a conversion program could convert one file structure into the other.

A clear difference between the two games is the design environment that the Dark Forces design team used. While the Doom development team used a NextStep environment with its own tools, the Dark Forces design team used a DOS-based design environment with Watcom's C compiler 10.0 and AutoCAD release 12.

The Dark Forces team chose AutoCAD over other rendering packages because most of the designers were former architects more accustomed to using CAD software. A dedicated programmer customized the AutoCAD interface using extensions to add elements such as lighting and wall textures. In the end, developers used the base AutoCAD program mainly for the geometry of building the space.

For the games X-Wing and Tie Fighter, LucasArts used a general-purpose, polygon-based engine that could render pretty much any object from the Death Star to a space buoy. The Dark Forces engine is more specific in design. It combines a true three-dimensional engine to render the objects in the game and a flat bitmap manipulation engine to render the characters. LucasArts is looking to make the performance acceptable on a 486/33MHz machine with a reported frame rate of 30 fps on a 486/66MHz. This game engine is called—appropriately enough—the Jedi, and LucasArts hopes to use it again in future games.

A shift in the market toward CD-ROM games caused LucasArts to decide—halfway into the project—to make Dark Forces CD-ROM based. This allowed the game to have more elaborate animation sequences, but the size of the remaining game code was comparable in size to a floppy-disk-based game. Indeed, the overall design of the game is more similar to the disk-based, X-Wing-style games than the CD-based Rebel Assault.

Star Wars, according to Sinnett, is ingrained into the culture of LucasArts. The basic design of all *Star Wars* games is open ended, to the point that designers can do almost anything they want. No direct link exists between the various games in the *Star Wars* license, so every game can end with Darth Vader meeting some horrible new fate.

The licensing division of Lucasfilm is involved only when a game introduces new characters to the *Star Wars* pantheon. An example is the Dark Trooper, a new class of storm trooper introduced in Dark Forces. This character had to be approved by licensing and, ultimately, by George Lucas himself.

The Future

LucasArts is looking toward the future with a variety of projects in the computer entertainment area, such as Commander, a programmable interactive simulator that can be an airplane, a submarine, or a spaceship. Another project is a multimedia network project with Japanese giant Fujitsu involving a multiplayer Doom-style game with a *Star Wars* theme. And, George Lucas is rumored to have some new *Star Wars* movies in the works, which will undoubtedly spawn new games to capitalize on the films' inevitable popularity.

Regardless of these projects, one thing is certain: whatever the outcome in the interactive entertainment boom and the Silicon-Valley-meets-Hollywood craze, one of George Lucas's companies will be right in the middle of it ■

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

Back

Artist vs. Artisan

by David Sieks

A paradigm shift has begun in the game industry—arcade action now coincides with Art itself. So what does that make us? Game developers or Arteests?

I had spent the morning climbing counter-clockwise, spiraling up through cloistered rooms and spaces lovingly lit; altars to artistic genius. Past the beautiful, the daring, the exhilarating; past Miro and Picasso, past Kandinsky and Klee. Finally, at the very top of Manhattan's Guggenheim Museum—that corkscrew through the pulpy core of Modern Art—I arrived at Kosuth.

"Clear, Square, Glass, Leaning," an installation piece by Joseph Kosuth, consists of four clear squares of glass, leaning side by side against the wall at a slight angle. As I arrived on the scene, a young art critic in pigtails and pink stockings was attempting to ferret some meaning from this piece as her mother looked on.

In the center of each glass square is lettered a word in neat, white type. Pacing studiously the length of the installation and pointing her small finger at each word in turn, she read aloud, proudly, in a clear contralto. "Clee-ur ... Squay-ur ... Glass ...um..."

Mother was there to assist. "Leaning," she offered. "Leaning," repeated the waif, regarding the Kosuth with delicately furrowed brow. "Leaning, Mommy? Is that what makes it Art?"

Art Who?

I relate that 100% Guaranteed Genuine No B.S. anecdote to you here because I think it crystallizes the confusion in our society surrounding the very notion of art. What it is and, conversely, what it isn't. What does any of this have to do with game development? Read on.

Art, as a label, intends to validate both the artist and the work. The attempt is not always successful, because broad-based agreement as to what qualifies as Art is hard-won. Even once the label has been made to stick, the question exists as to what quality content it assures. "I don't know art, but I know what I like" has long been wielded as a sharp pin to deflate such assumptions of worth.

Many people like computer games, though few, if any, would think to define them as Art. Some game developers might contend that the act of creating an entertaining game is an art form, yet even they would hesitate to acclaim the end result a work of art. Games are diversion, fun for fun's sake. In any case, the very word "game" carries with it a connotation of frivolity that effectively serves to prevent more serious consideration.

The visual artist involved in the production of a game fares little better, respectability-wise. Generally, our creations are rarely considered Art, and we are considered more artisans than true artists, probably ranking lower on the totem pole than the more traditional varieties of commercial artist. This is partly because we are associated with a frivolous product and also because of the medium itself, which has, until recently, delivered game graphics that are relatively crude.

A New Medium

Of course, the rapid pace of technology has changed all the familiar rules. The realm of possibility with computer graphics becomes ever more stunning. It is no longer uncommon to find the dazzling products of sophisticated graphics software gracing magazine covers, on television, in major films, and even featured in galleries as, you guessed it, Art.

Art? Dogfight was created by Designers' CADD Co. of Cambridge, Mass., using sample 3DStudio geometry and plug-ins from Yost Group Inc.

With the full range of multimedia capabilities now so commonly available, the games of today—or more to the point, those of tomorrow—are not, in a manner of speaking, your father's Oldsmobile. Not only has it become possible to make games that are neater looking, better sounding, and more smoothly interactive, but suddenly (relatively so) we have at hand tools truly fit for the artist and a new medium for meaningful creative expression.

Yes, I'm talking about computer games. Why not? The art world has traditionally met encroaching technology with resistance (ask any airbrush artist). Yet, the computer is too powerful and flexible a tool to be so ostracized. Witness its near ubiquitous presence in the design field.

How's this as a vehicle for creative expression: graphics tools that allow live-action video, photo-illusion, and emulation of traditional drawing and painting media with a palette of colors vastly greater than that offered by any paint manufacturer; quality stereo sound; an ever-more-responsive array of interface options that allow the audience to become an active participant. With the list of possibilities growing almost daily, the computer game can become not only an art form, but an unprecedentedly interactive, malleable form of art, guided by the artist/game designer and uniquely shaped by the audience/user.

The time is ripe for a paradigm shift, where society acknowledges the computer game as a veritable art form fully as expressive as the novel, film, theater, ballet, opera. Capable of instilling wonder, of exciting the imagination, of provoking thought, of captivating its audience. Capable of beauty. In a word, Art.

Feet Back On the Ground

What, you may be asking yourself, does this have to do with the two-way scrolling shooter I want to make to run on the 386? Maybe not much at all. Using the available technology for meaningful creative expression is a new possibility, not a new mandate. The computer game was born out of far simpler pleasures, and there is doubtless much vigor, joy, and profit left in the familiar arcade genres.

Someone out there is probably clever and creative enough to figure out a way to make arcade action and art coincide in a 16-bit cartridge. The paradigm shift I called for earlier begins within the industry. There are indications that it already has begun.

By now you might easily have heard more than you ever wanted to hear about Broderbund Software's Myst (especially if you're the jealous type). Brodurbund's success is remarkable, though, because they dispensed with elements that seemed nearly indispensable—fluid action, readily identifiable goals, violence—and delivered a hauntingly evocative, beautifully rendered game. In a different way, Infogames' Alone in the Dark defied convention as well by incorporating a shifting "cinematic" viewpoint that introduced to computer games new possibilities for storytelling power and depth. A fresh use of viewpoint also buoys System Shock, from Looking Glass Technologies. By combining what they call a "6-D" perspective with highwire tension, they keep the player literally peering around each corner, attaining a level of audience immersion any Hollywood director would envy.

Tremendous talent exists in the field today, and gaming horizons are constantly being stretched—if in mostly familiar directions. But that's what paradigms are all about. They're tough to shake. The creative impulse to make something more than "just a game" will not likely come from a game developer looking to make a better version of something we've already seen, which is in itself a noble effort. All games don't need to aspire to the level of Art, nor should they. I can hardly imagine anything more tedious.

A New Role

At the same time, all games would profit if artists were given a greater role throughout a game's development. People in the industry tend to use "the talent" as tools—tell them only as much as they need to crank out their little segment and make sure they do as they're told—like some sort of AI-Paint program. (I can hear programmers drooling at the very idea of being able to realize their game concepts without relinquishing one iota of control, but no, such software does not yet exist. Just last issue, though, this magazine profiled software that can compose tunes given only thematic guidance by the user. I doubt it's made many friends amongst living, breathing, bill-paying musicians.)

What's lost by such an approach is potentially valuable creative input that could nudge a game in directions unthought of. What sort of game might develop if the renderers and musicians were sitting at the table from the earliest brainstorming sessions? What new ways might be conceived to interweave talent with technology, playability with beauty, strategy with color and sound? Fantastic collaborations are waiting to happen. Vast new horizons await exploration.

I've found myself dreaming lately of a day when the *New York Times* reviews interactive computer art with the same highbrow intensity it devotes to cinematic and literary efforts. Maybe, someday, I'll make my way to the very top of the Guggenheim and find there a small child staring rapt into a glowing screen that responds to each whispered command, that whispers back and hums and sings and fills that topmost corner of the museum with gentle swirls and eddies of sound. The parent, smiling quietly in the background, will be visibly pleased that the child shows such appreciation for fine art.

It may be a farfetched conceit, but that's my job—I'm an artist. ■

Send your thoughts on art and artists in the game industry to Dave Sieks at Game Developer, *or e-mail him at dsieks@arnarb.harvard.edu.*