gd

GAME DEVELOPER MAGAZINE

JUNE 1994

UBM Tech

CD-ROM: Package or Platform?

Now that game developers are faced with the largest storage device in the history of their industry, what are they going to do with it. Just as the command line gave way to the GUI, this shiny ethereal disk is muscling its dull, flimsy, magnetic ancestors out of the way. Lower manufacturing costs were one of the first benefits, as low as a dollar a unit. They were lighter to ship and harder to damage—no more damaged PC disks. They can be repackaged without threat of corruption or infection. And, as far as piracy goes, in the words of one developer, "Nobody ever downloaded 500MB from a BBS."

But are they, as they are marketed now, a software platform separate from disk-based games, or are they merely a storage medium and should be treated as such? When I ask developers what platform they're writing for, more are saying CD-ROM. When asked to clarify, they usually say, "Well, MPC2 and Macintosh first, then Sega CD and maybe 3DO or CD-I." Clearly, they're talking about high-end authoring systems manipulating either captured video or rendered images with CD-quality sound files—a development process where the main emphasis is on raw data that can be used across a variety of platforms.

Playability

The problem is, many newer CD-ROM games have the playability equivalent of Pong games with captured video images of professional tennis players and CD quality sounds of bouncing balls. This is because high-level authoring systems available can deliver the kind of performance that is needed on every platform, so most cross-platform CD-ROMs rely on a variety of kludges, just to get a simple game manipulating video to work. Games need to be more tightly integrated into specific platforms to get good playability, and more of the CD-ROM's cross platform identity will be lost.

Stand or Fall?

The potential of the CD-ROM as a platform is tremendous because it is viewed as a superset not subset of the existing computer game industry. Everyone's hoping that nontechnical people who would never buy an Ultima, flight simulator, or Doom will be willing to buy a CD-ROM game designed to appeal to a wider audience—changing the computer into interactive VCR. If these technical neophytes' first experience is a bad one, for $60 a disk, they're not going to continue making the same mistakes.

It will be this next year as these consumers make their first CD-ROM purchases that will determine the shape of the industry. If CD-ROM games are able to vary more in subject matter than traditional computer games, retain their platform independence, and capture new demographics, they will attain the status of new platform. If not, they will just be another means to get product to market and will just be another label on the side of the box. Regardless of the outcome, it looks as if the CD-ROM is here to stay.

Alexander Antoniades
Associate Editor

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
Atten: Larry O'Brien

E-mail is even better:

GAME DEVELOPER

Editor Larry O'Brien
76702.705@compuserve.com
Associate Editor Alexander Antoniades
aantoniades@mfi.com
Production Editor Barbara Hanscome
73611.633@compuserve.com
Editorial Assistant Andrea Pucky
apucky@mfi.com

Cover Photography Carter Dow Photography

Publisher Veronica Constanza
Project Coordinator Nicole Freeman
76702.706@compuserve.com
Group Director Regina Starr Ridley

Advertising Sales Staff
New England/Midwest
Angela Barnett (415) 905-4983
abarnett@mfi.com
West/Southwest
Yvonne Labat (415) 905-2353
ylabat@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 John Rockwell
Circulation Manager Gina Oh *goh@mfi.com*
Circulation Assistant Philip Payton *ppayton@mfi.com*
Newsstand Manager Pam Santoro
Reprints Andrea Varni (415) 905-2552

mf 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

Get Your Game on a CD-ROM

Cool graphics are just one advantage to writing your game specifically for CD-ROM. But a high-end clip like this can cost a lot in terms of money and staff resources—several hundred clips can put your project out of business. Carefully planned and placed, though, you can make efficient use of your resources and still come out with a visually superior game.

The concept of developing games specifically for CD-ROM or porting existing products onto CD-ROM is appealing and a bit frightening. There are advantages to using the CD-ROM as a delivery platform, and there are technological barriers as well. Your managers see CD-ROMs as an economical alternative to floppy disks, and the lure of 550MB of storage to play around with makes programmers drool.

Unfortunately, most companies are unfamiliar with the practical step-by-step processes involved with actually producing a disc—and what the pitfalls are. I will guide you around some of the pitfalls inherent with CD-ROM development and demystify the process somewhat.

Not Just for Aerosmith Anymore

While CD-ROMs have been around since the mid-1980s, most products produced for the medium have been fairly plain, unimaginative text-based reference works. Almost 90% of the discs available are plain-vanilla DOS products with no graphics, sound, and animation. Only in the past few years have CD-ROM players penetrated the market enough to make broad-based consumer-oriented software financially viable.

Not that a single dominant platform has emerged. CDI, 3DO, SegaCD, CDTV, CD-32, Jaguar, FM-Townes, and a few other TV-top or stand-alone systems are still trying to establish a foothold in the home market (none is a clear winner yet), while Apple, IBM, Tandy, Atari, and Commodore all support CD-ROM drives as add-ons to their PC products.

There are also hybrid systems like Laser Active just introduced last fall that plays video laser discs, game cartridges, and CD-ROM-based software. All these systems have their idiosyncrasies when it comes to writing code for them, but fortunately there are some commonalties.

One common thread that runs through these systems is that the discs all adhere to the ISO-9660 standard. The ISO standard describes the physical layout of data on the surface of the disc and some of the structure the data must conform to. Beyond that, it is up to the individual system firmware or device drivers

to read that data and translate it into a form the native operating system can understand.

In other words, all CD-ROMs that adhere to the ISO-9660 standard can be read by all CD-ROM players. That's the good news. The bad news is that, like Apples to IBMs, just because you can read the disc does not mean your system can do anything meaningful with the data. Animation files saved in Commodore's ANIM.7 format still cannot be played on an Apple Macintosh; code written for the Windows environment won't run under OS9 (the basis of Philip's CDI operating system).

As the developer, you may have to write different versions of your program for each system. You can store multiple versions on the same CD-ROM, and each platform should be able to use the same disc. The other bonus to the ISO standard is that, unless you really need to write code "down to the plastic" (for a custom video playback routine or to squeeze the absolute maximum performance from a disc), you don't really have to worry about file structure. (You *do* have to worry about track layout and organization, but we'll get to that a bit later.)

The Need for Speed

Another common element to all CD-ROMs is that they are *slow*. A single speed CD-ROM drive boasts a whopping data transfer rate of just under 150K per second. Compare that to a medium speed hard-disk drive with transfer rates approximately 10 times faster. Another perhaps more crippling factor is seek times. When CD-ROM drive vendors claim a 150K transfer rate (or double, triple, or quadruple rates), they somehow conveniently neglect to mention that almost all CD-ROM drives have atrocious seek times. If you are just loading data from a text-only reference CD, it doesn't make that much difference. For game developers who want to load sound, animation, video, and data in as close to real time as possible, those seek times can be a nightmare. Also, different systems vary the default chuck sizes.

Let's look at a worst case scenario. When you send a read command to the drive, it must first move the read head to where it thinks the file probably is. It reads a 4K or 8K header chunk (depending on the firmware or hardware configuration) to see if it's got the correct file. If it's the right file, it has to wait for the disc to spin around again, then reads 80K chunks (regardless of how big the file is) until the entire file is read.

Let's say you are reading a 30K file. Seek times on some CD players, going from the outer edge of a disc to the inner, can take up to 1.2 seconds! Yes, seconds! (I have personally worked on systems like this. TV-top systems trying to appeal to the masses and are highly cost conscious tend to put the least-costly and thus least-efficient drives in their machines.)

So, 1.2 seconds plus .03 sec. for the header read, plus .5 sec. for the actual data (remember, it reads an 80K chunk no matter how big or small the actual file may be), and we are at 1.73 seconds to read a 30K file! All this is assuming it hit the correct spot the first time. If not, add a bit more time. Suddenly our 150K

by Guy Wright

CD-ROMs have blown open the game market with the potential for high-end sound, video, even multimedia. But beware the pitfalls of technology! You need to plan carefully to create a superior game.

A few well-placed 10-second clips can mean the difference between award-winning software and shovelware.

per second transfer rate has dropped to 20K per second. Imagine you are trying to double-buffer sound files while loading something else, and you begin to see where the biggest CD-ROM bottleneck occurs.

Granted, this is a worst case situation, but ignore this warning at your own peril. You may think your program is fine until you start getting irate calls from people who say that the audio is garbled or the QuickTime video sequence is annoyingly jerky on their system.

If you are producing a product for a TV-top system like 3DO or CDI, there are ways to get around some of these problems. You can force the drive to read certain size chunks (assuming you know exactly how big each file is before you issue a read command) or just tell it to start dumping raw data into RAM where your code will sort everything out on the fly. But if you are developing for a PC platform where you don't know what kind of CD-ROM drive the user is likely to own, you have to resort to other tricks.

The first, simplest, and most valuable trick for teasing performance out of a CD-ROM drive is physical layout of the data on the disc. If you are careful about the physical layout of the files on the disc (keeping sequential files close to each other), you can minimize the seek time problems. If the read head doesn't have to travel as far, it won't take as long to start transferring your data into RAM.

Of course, you must change some traditional ideas about how you organize your data during development. For instance, don't put all your sound files in one directory, your pictures in another, and your animations in a third. When you transfer those directories to a CD disc your files will end up far apart on the disc, causing longer seeks.

Instead, put sound, picture, and animation files that will be used at the same time in their own directory. That way you can be certain they will be near each other on the disc when it comes time to read them. For some systems, there are even optimization utilities that will track file use during emulation and tell you which files should be closer to each other.

Down to the Plastic

This is a good time to run through the traditional steps in making a CD-ROM disc.

Step 1: Design your product. You should already have a good idea about how to do this (if not, you probably shouldn't be in the business). What do you want? What will it look like? Create a wish list and go back and modify it based on cost, resources, development time, and so on. It's tempting to think of a CD-ROM as a panacea for all the problems you face with traditional game design and production. But before you finish your design, look at your original idea and some of the cold, hard numbers.

If you want video in your product, you must invest in a whole new world of hardware that most of the programmers on your team don't know anything about. There are VCRs, editing decks, SMPTE time code generators, editor-controllers, cameras, and, of course, actors, costumes, scripts, lighting, audio, directors, and video editing. I'm not saying you shouldn't include video, but think back to those cheesy car-dealer commercials on your local TV channel. Those obviously shoe-string budget, low-quality, 30-second spots cost tens of thousands of dollars to produce!

If you think you can do it for less, take a look at your own home videos and decide if that is the quality you want in your new game. Call a few video production companies, costume houses, and so on, and run some of the numbers yourself. You may end up spending $100,000 or more for a few minutes of video before you get it into the computer.

Once you have the video, you have to capture it and convert it into a video format for your particular platform. QuickTime and Video For Windows charge a run-time fee per product (and you can't count on your buyers already having them). In addition, there is the cost of the video capture equipment and its learning curve. You can plan on adding a few months to the development cycle if you are including video in your game.

If you can overcome all these obstacles and manage to get the video into the computer, a good rule of thumb is to figure on a megabyte of storage for each three to five seconds of video. You can have a spiffy intro video of two minutes that should only take about 60M (in a small window, that is, not full-screen and not quite VCR rental movie quality).

If you consider that a 70-minute movie can barely squeeze onto a CD-ROM using MPEG hardware compression, you begin to realize why there aren't a lot of interactive video games out there (exactly none at last count). But don't let all this discourage you. A few well-placed 10-second clips here and there can mean the difference between award-winning software and just another shovelware port. Investigate the problems, costs, and eventual benefits of video, and then make up your mind whether to include it or not.

If you want a complete CD-quality soundtrack, there is more bad news and good news. CD-quality audio (at a

22KHz sample rate in stereo) ends up gobbling CD-ROM space as hungrily as video. (That's why you can only fit 70 minutes of music on a CD-audio disc).

There is another problem with CD-audio. Because of government regulations brought about by the record industry, there is a special hardware feature built into all CD-ROM drives. When the drives switch to CD-audio playback mode, they automatically cut in digital to analog converter circuitry before the signal gets to output. This was done so no CD drive would send out pure digital signals—if they did, people could make perfect, digital byte-for-byte copies of CDs.

When you issue a CD-audio playback command to the drive, it switches into an analog output. In simple terms, while the CD-audio is playing, you cannot read any digital information off the disc at the same time. So, you must preload all the code, pictures, animations, or whatever before you start that totally hip soundtrack playing and wait until it finishes before you can load anything else.

The good news is that 95% of the people who will be running your game on a PC or Macintosh will not have their system hooked up to a $4,000 audio system. More than likely, they will just have a SoundBlaster card running through dinky little Radio-Shack speakers that, at their best, can only reproduce audio in the 8KHz to12KHz range. (For a point of reference, a telephone earpiece speaker operates at about 6KHz or voice quality, AM radio is about 8KHz to 10KHz, a very high-end stereo system can only go up to about 18KHz, and most normal human beings can only distinguish differences in sounds up to about 20KHz.)

Why is that good news? You can have your digitized audio, music, voice, and sound effects and sample it at 12KHz to 14KHz, which is more than adequate for 95% of the systems out there. You will have remarkably clean sound in digital format that you can load from disc and play back at the same time. Sound still costs about 1MB for each minute, but you don't need to overdo it on sound. (If you are just doing explosions and laser blasts, you can easily drop down to 8KHz sample rates with no loss of clarity and still sound better than disk-based games.)

You will however have to be careful during the recording and sampling stages of development. You should do a number of tests first to determine recording formats, sample rates, and so on. It's also a good idea to digitally audition your narrators because there are some human voices (particularly women's voices) that require 15KHz to 16KHz sample rates to reproduce clearly.

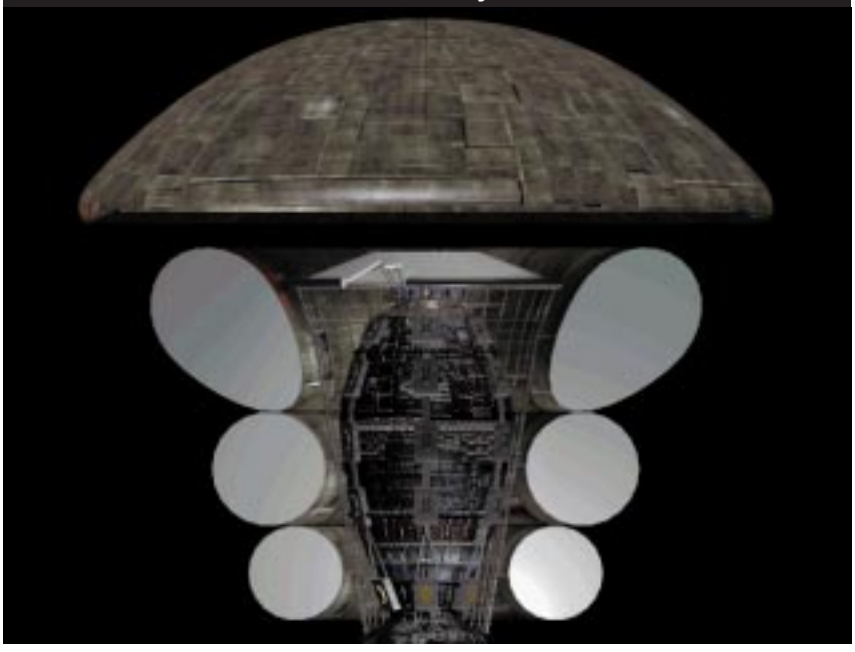
It is also a good idea to get your audio recording done in a professional sound studio. The costs are not that high ($50 to $100 per hour), and you won't end up with an hour's worth of audio tape that has the hum of an air-conditioner in the background, cars driving by the office, or 60-cycle florescent lights buzzing throughout. You need to invest in decent sound digitizing equipment and software, although sound editing is remarkably easy in digital form as opposed to traditional sound-editing techniques.

You can store hundreds or even thousands of images and animations on a CD-ROM disc, but where are you going to get them? I know of companies that have spent $1,000 for a single computer graphic screen. It was nice, but imagine buying hundreds of them. Maybe you have an in-house graphics department. How many screens does it produce in a week? One...two...ten? How long will it take to generate a hundred...three hundred...a thousand? How much would it cost to buy a hundred graphics screens? How long will it take your people to generate 20 animation sequences or more?

More and more companies are turning to stock houses for images, and some of the smart companies are looking into licensing existing artwork. Some comic book illustrators get paid very little compared to traditional computer artists, and their talents rival the best computer graphics anywhere. It might be worthwhile to investigate hiring traditional artists and scan their pencil, pen, and ink work into the computer. It would mean buying and learning the intricacies of scanner technology, but it may be worth it in the end.

Your design specifications now include a few dozen animations, some well-chosen video clips, 10 to 20 minutes of sampled sound, and something less than a hundred graphic screens. Once you are ready with a full design specification, you are ready to move on to the

Civilization of the Future or Memory Eater?



next step.

Step 2. Develop your product. Here is where the hard-core programming, graphics creation, animation, music, sound effects, digitized voices, and everything else gets done.

Step 3. Hard disk image. Create an image of the product on a separate hard disk partition, complete with boot code, installation code, and so on. This is a very important step in developing for CD-ROM because what you put in that partition is what will end up on the disc. This is a good time to clean up unused code, tools, errata, programmer's notes, internal documentation, trapdoors, and all the thousand-odd files that end up on a hard disk during the development process. Remember, if you accidentally burn a copy of your compiler, source code, font, or favorite paint program onto the CD-ROM, you might be in for some legal problems, the least of which would be paying a royalty to the originator of those products.

Just because all those files appear in nice alphabetical order when you do a directory listing does *not* mean they are sequentially laid out on the drive. Some premastering software creates a complete byte-for-byte image of the hard drive as it stands, not as it appears. Remember that file number 2.pic appears between 19.pic and 20.pic, so you may want to number files 001.pic, 002.pic, and so on. This may not be a problem for you or your system, but some people spend weeks trying to figure out why the CD-ROM image is different from what they think is on their hard disk.

Step 4. Hard-disk testing. At this point, it's a good idea to test the product from that hard disk partition. If possible, boot straight into that partition and see if everything works. You may have to create a floppy boot program that simply transfers control to that partition and see if it works. You will probably find that when booted from that partition, the program fails because you omitted a DLL, initialization process, or some code that exists on your development platform but may not exist on your target platform.

This testing will also help you build your install routines (if you have any). This is particularly important on TV-top systems, many of which transfer total control to the CD-ROM and count on it to do everything including boot the system. Once everything seems to work off the hard disk partition it is time to premaster the disc.

Step 5. The gold disc (premaster). Before you actually start producing discs, you go through a premastering phase that's divided into two steps: a build and a premaster or gold disc.

After you have your program finished and all the picture, sound, and other data files just the way you want them, you go through a build phase. A build is where a separate program reads through your hard disk and creates a build file, which is simply a text file with the proper header information and a list of all the files on the hard disk drive that you want copied in the order that they will appear on the CD.

Once the build file is created, another program creates an ISO format image of the data, ordered according to the build file. That image is then written to a write-once disc. This is called the premaster or gold disc. These discs use a very, very thin film of real gold and a laser to create the pits. The gold disc is then used for final testing in a real CD-ROM drive on a real system.

There are two ways to create a gold disc. You can either send your data to a CD-ROM mastering house or you can do it yourself. Most CD-ROM mastering facilities can accommodate data in just about any format, such as streaming tape, Syquest data cartridges, Bernoulli drives, even hard disks. They will, of course, charge you for a premaster (usually under $100), and some houses will let you apply that charge toward the actual mastering charges later.

You should already be checking into mastering and duplication charges by this time anyway, so perhaps you can work out a deal. If you are letting a CD-ROM mastering house do the premastering for you, be sure that it knows what format you are sending and what the target platform is. Most CD-ROM duplication houses are more than happy to accommodate you no matter what the technical considerations are (after all, they want your business when it comes time for duplication).

If you plan to do a lot of future projects on CD-ROM (or a lot of premasters), it's probably a good idea to look into the purchase of a WORM drive of your own. Write once drives fall into the $2,000 to $5,000 range, with various features and software. It is probably worth looking into the various configurations

carefully before you purchase one. If you go the do-it-yourself route, you must do the build phase on your own, which actually does give you more control over the finished product.

If your premastering or build software is good, it will let you edit the build file. Here is where you may find file organization and ordering problems. You may also be able to correct them here by editing the build file. Remember to keep track of your changes because, no matter how smoothly things have gone up to this point, you generally have to go through the build phase at least a few more times. When you have the build file the way that you want (organized so that frequently used files are grouped together and near the beginning of the file), you can cut a gold disc with a command or two. Depending on the WORM drive, this can take minutes or hours.

Step 6. Testing. When you get your gold disc back from the duplicator or from your WORM drive, it is time for real-world testing. Here is where you will find CD-ROM specific problems, read and seek time bottlenecks, and other things you never counted on. The gold disc is in every way exactly like the disc that you will be shipping to your customers, so if it doesn't work on your target platform system at home, it certainly won't work on your customer's machine.

Test the gold-disc version to death because this will be your last chance to solve any problems. Try it on an actual low-end system that doesn't have your developer tools loaded (or anything else for that matter). Test it at home. Test it in the field. Test it everywhere you can think of, even if you have already gone through beta testing on hard disk.

If you end up having to make another gold disc, go through the whole testing process again with the new disc. You would be amazed at how often fixing one problem causes three more to crop up. Even if you are under the gun, it's 4:45 and the UPS truck is about to leave, don't give into the temptation to send out a new gold disc that hasn't been tested! Gold discs are not perfect! A single speck of dust at the factory can create a flaw in the disc, and gold discs aren't as durable as final CDs. One bad byte in the wrong place can destroy a program, and, if it's bad on the gold disc, you'll end up with a thousand copies of a bad disc.

Step 7. Mastering. If the gold disc works the way you want, you can then send the gold disc, (or magnetic tape, or Bournelli or Syquest cartridge, or even your hard disk drive along with your build file) to a duplicator who then makes a master. You can't make your own CD-ROM master discs. CD-ROM mastering is one of those dark, arcane artforms roughly equivalent to mass-producing VLSI chips that only a few companies in the world can accomplish.

It involves millions of dollars worth of highly specialized equipment, highly trained engineers dressed in air-tight space suits and a class 2 cleanroom. Fortunately, duplication houses only charge about $1,200 for each master. Don't expect to get the master back either. Most duplication houses will store them for you if you need another run, but normally they don't give copies back to the customer.

Once a master is created, the duplication process is pretty much like stamping out a vinyl LP record. The master is mounted on a press, the CD-ROM inner layer is stamped, the plastic coating, label and whatnot applied, and the disc is popped into a jewel case (or whatever you specify). Depending on the quantity and turnaround time, CD-ROM discs can be produced for under $1.50 each. If you want the duplicator to insert your discs into jewel cases, insert instructions, and shrink-wrap them for you, each disc will cost about $2.00 each. The duplicator will even ship the product for you.

You must work out the particulars with your individual CD-ROM duplication house, and all will be very happy to meet your needs. You may be able to get a dime or two with a volume discount, but disc prices are pretty low, so don't expect 50% off if you order 10,000 copies. If you are developing for some systems you may find that the duplicator will charge you an additional platform license fee that goes back to the system manufacturer. You may also encounter a setup fee of a few hundred dollars. Some duplicators will wave this fee the first time, but charge you if you need to go back on press at a later date.

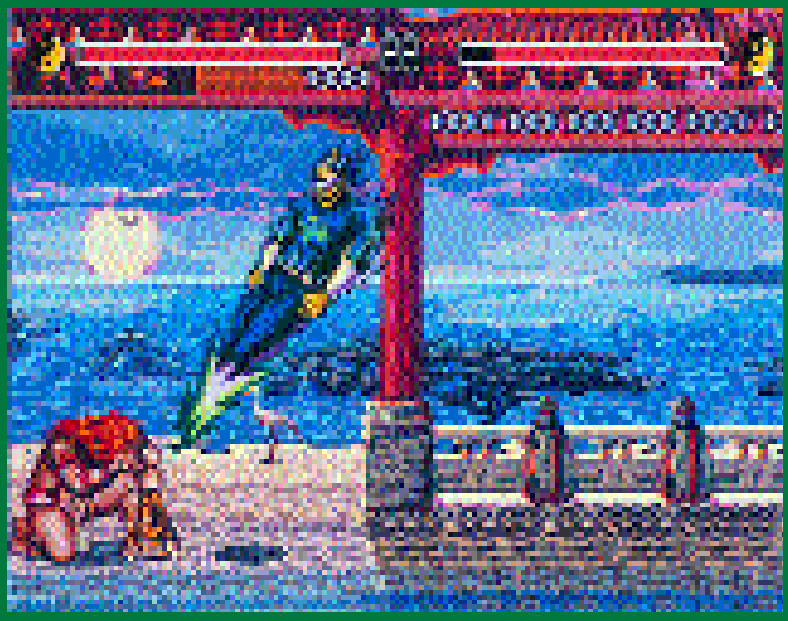
Treasures and Traps

So, while CD-ROM may seem like the end-all, be-all for software developers, it is a mixed blessing. Yes, you do get tons of storage space, but you have to count on additional load times. Yes, it is almost like shipping your product on a very large read-only hard disk and yet, because of physical layout constraints, you have to be careful how you premaster for optimum performance. Yes, you can add video, digital audio, animation, and skads of graphics, but each comes with added development costs, headaches, and design considerations.

If and when you decide to jump into the CD-ROM development venue, you are bound to find hidden treasures and traps. There is a whole new world of interactive entertainment waiting to be explored. You will find those rewards and pitfalls sooner than any other software developers because, as you and I know, it takes 10 times more sophistication and elegance to develop a good game than it does to develop just another word processor or spreadsheet. Game developers are the ones who test the limits of technology. No other software producers can do that or have to live up to the standards that game buyers and developers set for themselves. ■

Guy Wright has designed, produced and developed nearly a dozen interactive multimedia CD-ROM products for various companies. He has designed games, video production software, and multimedia authoring languages and founded two software companies. He has written two books on desktop video and is currently working on a killer CD-ROM cyborg simulation game that should put all others to shame. He can be reached via e-mail at gwright@mfi.com or through Game Developer *magazine.*

Target Your Game: Computer vs. Console



Games like Sega of America's Eternal Champions will sell to millions (zillions?) of consumers this year, but is it a video or computer game. With delivery mediums like the CD-ROM, traditional video games are crossing the line and being played not only in the arcades and on TV-top consoles, but on the PC and Macintosh as well. As the computer and video markets converge, opportunity arises for everyone!

Want to quickly compare home video and computer games? Tally up sales. Six to seven billion dollars worth of video games were sold in North America in 1993 vs. $410 million in computer games, according to the Software Publishers Assoc. That ratio, about 15 to 1, is an order of magnitude and a half.

But wait—nobody counts retail sales of IBM compatibles and Macintoshes or even joysticks and soundcards, yet Super Nintendo Entertainment Systems and Sega Genesis and Sega CD units are lumped into the home video game or console totals. And aren't the chip innards of cartridges more properly counted as hardware? Maybe we should factor out their difference in cost from all the floppy disks in an average game box or the stamping cost of a CD-ROM. More to the point, how do computer and console games differ, and how are the two suddenly growing together?

The Two Worlds

The biggest difference between computer and console games is the people who play them. Their preferences appear to dictate the kinds of games you find on either platform. Part and parcel of that content are the interfaces, which differ strikingly. The hardware available is largely responsible. It accepts player inputs, generates compelling graphic output, and processes the one while supporting the other.

Any console developer must purchase a development system to accommodate that dedicated CPU and its architecture, but every developer employs some variety of software tools and a balance of two separate languages. Finally, the process by which a video or computer game, once commissioned, reaches the hands of consumers also subtly differs.

Younger people play video games, while older people play computer games. *Nintendo Power* editor Scott Pelland calls the core video game-playing group 12- to 14-year-old males. At Sega of America, associate director of third-party licensing Steve Ackrich offers "6 to 16 vs. 18 to 35" as a ballpark age breakdown.

The game platform of choice for economically dependent consumers is clearly the video game console or set-top

box. “Right now computer games seem to be more involved, more complex, and more for older people just because of the target platform. It's an older audience. You don't have 10- to 14-year-old kids going out and spending $2,000 on a PC,” says Robb Alvey, a producer at Virgin Interactive Entertainment.

The dominant appeal of video games is the promise of instant gratification. There are indeed “two different types of gamers,” claims Maxis producer Michael Perry, who began his career at Hudson Soft USA in video games. Video gamers are “action or sports oriented. They want to put the cartridge into the machine, turn it on, and play without having to read a manual, or study it, or know anything about the game. They just go for it.”

Sega producer Tony Van has also worked in both worlds, including Activision and LucasArts. He says, “The PC and video games are different markets. On the PC side, they are more forgiving because most of them are hackers. You cannot install a PC game these days without hacking your AUTOEXEC.BAT and CONFIG.SYS files.”

One thing video gamers are not is hackers. As investors in their own entertainment, one experiential venture at a time, they are risk-averse. One unpleasant episode with a faulty product, and their pleasure capital could be wiped out, as Nintendo learned from the early 1980s Atari market and created zero-defect fun.

Video games are fad items, based on fast-moving technology, and even the best can be outdated within a year. You can hold a cartridge in your hands, but make no mistake—video games have been successfully transformed from a good into a service, like all entertainment always was and like more and more software is. And as any good service-provider knows, an annuity is better than a lump sum. The recent stir over video cartridge rentals is ironic because a video game console is really just a subscription and cartridges the renewals. With video games, we already rent our fun.

By comparison, computer games are more opportunistic; few would admit to buying a PC to play games. As a market, they are an afterthought. What can they offer an audience of greater wealth, maturity, and sophistication? Tony Van knows: “The trade-off is instant gratification vs. variation and depth of play. They all want the best game experience, on both sides. The PC side is willing to wait for it.”

Michael Perry agrees, it is “depth of gameplay. A lot of video games are very topical. You have a very simple purpose. In a fighting game, it's to beat the hell out of the other guy. On PCs, you can just dive in deeper and deeper. Users like that; they spend a lot of time with the game, seeing how deep they can go. It's not a one-shot thing like a video game.”

Successful console game categories are action and puzzle: sports, fighting, racing, side-scrolling and vertical shooters, and platform versions of the adventure genre. PC games are more typically simulation, strategy, role-playing, and first-person graphic and text adventures, as well as solitary sports like golf. After producing more than 30 cartridge games, Alliance Interactive president Gordon Walton has chosen to return to the platform with a keyboard: “PC games are more fun anyway. You see a lot more creativity, more robust games. There is a

by Jim Cooper

What's the best platform for your game? Consoles are in the limelight—but wait—isn't there a PC in half the homes in America? Market strategies play an important role in where your game ends up.

Sega's Ren and Stimpy–Interactive Hollywood

limited input bandwidth that constrains what you can do with a console game. This limits size not so much as depth."

Design

Yet the keyboard is not the key. Says Walton, "The mouse has a practically infinite input bandwidth to click on. It's the size of the screen in pixels divided by some size factor. A mouse interface is also more intuitive than 'Y-button plus Up arrow.'" The margin for error is awfully slim without that mouse; ask Alexandria designer Paul O'Connor. "You can't be sloppy when you're doing a console game. You've got three to six buttons, maximum, plus the directional pad. On a computer game you've got a keyboard with how many keys, a mouse with how many mouse buttons, and maybe a joystick on top of that. What it comes down to is the interface. It's almost the difference between a short story and a novel." The other half of an interface is the output—the graphics.

Rich O'Keefe of the Starwave Corp. explains the difference between character and bit-mapped displays, starting with video games. "Character-mapped displays break the screen down into 8-by-8-pixel characters and put those characters in tile positions on the screen. You can reuse characters in drawing the screen; that's where you save the memory. Computers, on the other hand, tend to be bit-mapped (unless you're talking about the Amiga). What that means is each pixel on the screen has a location in memory. Computers have character modes, they just don't get used because they are two-color: black and white. Video game characters tend to be multicolor."

O'Keefe continues, "Sprites are a special case of character that are independently placeable over the top of that background, to a pixel boundary. Sprites are supported in hardware; that is, they have a priority, and it's usually based on what's called their sprite number. Whether they appear on top of the background or behind the background, or one sprite appears on top of another, is basically dependent on some attributes you can set or their sprite number.

"To emulate a sprite on a computer display, say I have this character that I want to walk across this background. I have to draw the sprite and then, before I move him, erase the sprite, fill in the background, then move the sprite, saving a copy of the background where that sprite's going to be drawn. This is a very software-intensive chore. With real sprites, in a video game, the hardware does it all. It's completely unnoticeable. When the scan line is being drawn, it draws the background scan line, then the first sprite if it happens to be there and on top, and then layers it up like a sandwich. It happens automatically, and that's why you very rarely see real nasty shooting games on computers: the amount of processor required to do the erase and fill-in and move-the-object and all that is costly. It takes a lot of CPU cycles and a lot of memory."

Perry offers one design implication: "On a video game platform, you typically design your characters at least a fifth to a quarter the size of the screen. Say your screen is 224 pixels tall; you want your character to be 55 pixels in height. Just because of the low resolution of the TV, you want to be able to see that guy. But on the PC, since the resolution is much more dense and sharper as an image, you can use smaller graphics and get a lot more on the screen."

Consoles don't just come without keyboards or a mouse, they have to borrow your TV. This difference in resolution is key. One medium can be symbolic, using fine details of expression. The other has been necessarily literal, based on action; the only nuance is kinesthetic and goes by the name of "gameplay."

Perry continues, "NTSC is not as dense a resolution as the typical computer monitor. On a PC in DOS, we can have 640-by-480 pixels. On television, we are back down to 320-by-224. That makes a big difference when you design games for both platforms, especially if you are using something like a toolbar. On the DOS machine, that toolbar can be the full length of the screen. On a video game machine, you've got to really be careful and slice that toolbar up about a quarter of its size to fit. And then will the buttons be big enough?"

Text in games demonstrates this difference, in Tony Van's experience, "On the PC, text is O.K. Adventure, the original role-playing adventure game, was text-based. But it doesn't work on video, and that's one of the reasons role-playing games are nowhere near as popular on consoles as you would expect. Text is just not fun. But it is important in a game of any depth.

"When I showed Shadowrun, a role-

playing game, at the Consumer Electronics Show, what do you think were the first words out of a reviewer's mouth? 'Sure is a lot of text.' And you can't blame people. On TV, pixels blur! Computer VGA is high resolution. Somebody put it this way: you don't sit seven inches in front of the TV when you play on the Genesis." This medium, defined by action and hostile to the written word, is home to pictograms, multimedia ones.

"Consoles are built with just two things in mind: to entertain and to do it inexpensively," says Steve Ackrich. The Sega Genesis uses the old Motorola 68000 running at 12MHz with 512K of RAM. The Super Nintendo Entertainment System uses the 65816, an older, slightly inferior chip. Both are proven, available, inexpensive, and quite a bit older than the Intel 80386s, 80486s, and Pentiums that drive PCs today.

Development Systems

Just as video gamers have to own a console, developers must buy a development system, although it need not be from the actual platform provider. Systems for the 16-bit platforms from Sega and Nintendo cost around $15,000. According to Greg Tavares of Crystal Dynamics, they have the best hardware, which is required to perform tracebacks, but their software is so poor they are not the best systems to work on. Among the newer platforms, a Sega CD development systems costs $40,000; 3DO costs $15,000. Rumors of the price tag for Nintendo Project Reality and Sega Saturn are in the $100,000 range. By contrast, Atari development kits are $7,500 for the Lynx and $5,500 for the Jaguar.

In both worlds, the software engineering technology is constantly evolving. Console developers once started out equal, but every shop has built up its own arsenal of tools. Steve Ackrich is well positioned to observe the field: "You don't have everyone going down one clear path. You walk into one shop, and they have this bizarre setup. And you walk into another shop, and they have this completely different bizarre setup. In general, the pure developers have very 'clean' environments. We couldn't keep track of what everybody is doing even if we wanted to. After all, even we [Sega of America] are considered the competition. Sharing is limited; communication has a long way to go, but it is getting better. The industry is a very close-knit community."

Sculptured Software has even started selling parts of its own proprietary development technology, although director of software development John Emerson adds it sells only to approved developers of the Sega and Nintendo systems.

The first thing to realize is that video games quote their size in bits of memory, while computer games are quoted in bytes. A bit is one eighth of a byte. Video games are small; Street Fighter II, a monster cartridge at 24MB, would almost fit on two floppy disks. Gregg Tavares of Crystal Dynamics: "With video games you have to squeeze so much into a small amount of space that you take a vastly different approach. PC developers don't have to deal with it, so they don't even try."

How much can be squeezed? "Super Mario III from Nintendo was a 384K cart, and it had 88 levels. By comparison, Aliens Ate My Babysitter from Apogee, available for download from any BBS, takes up 2.3MB, and it has 13 levels. That's the difference," says Tavares.

This incredible parsimony is motivated by the memory cost. As Rich O'Keefe says, "Silicon memory is very expensive per bit." Cartridges cost between $15 and $20, compared to 75¢ to $1 for a floppy disk or CD. Using a 1.44MB floppy as the baseline, cartridges are 10 times more expensive per byte (or bit). CD-ROMs are 400 times cheaper.

According to Clyde Grossman, vice president of publishing for Spectrum HoloByte and former group director of software development at Sega, this is the difference: " 'Gee, product got a little bit bigger? Guess we have to go to another disk.' 8MB and 1 byte means the next size cart. What does that cost? Let's just say it means another $10 to the consumer at retail. It happens often enough. You want them to fill the cart, after all." The 7th Guest shipped on two CDs; the second wasn't planned, but didn't kill the project.

The two primary computer languages for games are assembly language and C. Assembler is the machine code native to a particular processor—no code is tighter. John Emerson says most PC code is in C, which is easier to work with. "Any high school kid could do it," he says.

Sculptured Software cartridges are written 75% to 100% in assembly language. In fact, all its Super Nintendo Entertainment System games are 100% assembler. Maxis uses "assembler when target platforms are 286 or a 68000-based Mac, and that's just because of speed, pure speed," says Michael Perry. "On the DOS side and Mac side and in Windows, we always program in C and use assembly language just for some of the bottlenecks to speed up some graphics process or some intense calculations. But as we get these more powerful machines, like the PowerMac, we can use straight C or C++, and it will run fast enough."

Anarchy vs. the State

The Software Publishers Assoc. does not police PC software content. Among consoles, however, Sega and Nintendo are candid about doing so. Steve Ackrich explains Sega's reasoning. "People don't realize this, but we have a very fragile marketplace. We've seen what can happen twice, once with the [Atari] 2600 and once with the 8-bit Nintendo. Nintendo let down their guard. Our customers don't know when a game is done by Sega and when it's done by a third-party publisher. We get letters all the time about John Madden Football [published by Electronic Arts]."

Ackrich continues, "Becoming a licensed publisher or developer is free, but not easy. We make it very hard to become a licensee. SOA [Sega of America] needs to see everything: the ability to develop games, the ability to distribute, market, and support games, and the ability to make quality games. We exercise two checks on individual game quality: first at the concept approval stage, and then in testing before final approval. We assign publishing slots; there is no limit to the slots, but the quality has to be there. All this is not to stifle competition because that competition gives you the best games. Most people at Sega come from third-party licensees, so they appreciate that we

are supporting them."

Atari vice president of software business development Bill Rehbock describes a laissez-faire approach to Jaguar licensing, for those who have also satisfied the stringent standards for becoming licensees. "We do not require our licensees to let us know what they're doing, but we request it. We keep a scatter chart of types—not titles—of games being worked on by all our licensees. We do not prohibit anybody from proceeding on a title, but we will be honest: 'Acme Software, we recommend you don't pursue this tennis game because it is worse than the three already in the works.' If they go ahead and proceed, God bless 'em. If they get eaten alive, that's O.K. too. Buyers for the big chains like Toys R Us know quality when they see it. The industry is a little more self-regulating than it used to be."

"We also won't reject—or slowly approve—your 3D polygon racing game because ours isn't ready yet. All third-party games get tested independently of our own projects. In fact, on the day that Tradewest announced their Jaguar Troy Aikman football game, we canceled our in-house football project."

Testing is easier for the more standardized video consoles, although not trivial. With direct responsibility for testing all—and only—Sega's third-party games and experience as a producer of PC as well as cartridge titles, Steve Ackrich says, "Testing is a nightmare on the PC, not as much so on the Mac, and almost but not quite painless on consoles. At Sega, we want to make sure that any game that comes out is compatible not only with all our existing machines, but with all future machines."

Rehbock describes a less ambitious process comprising all Jaguar title testing and approval. It begins when a publisher "submits the code to Atari, where we simply do 100 staff-hours of what we call sanity-check testing. The game just has to meet our cursory style outline: the Reset button works, Fire buttons are software-reconfigurable, Pause pauses the game, and while the game is paused you can adjust the soundtrack and sound effects volume levels. We turn over the code in two or three days."

Says Michael Perry, "We do really big simulations on the PC side. With our quality assurance process, sometimes products can stay in there for months. We've got a strict QA department.

Of course, updates to released products are unique to the computer game realm. You cannot download a patch to your cartridge from CompuServe. Gordon Walton says, "Carts have to be perfect. PC updates are cheaper, and the infrastructure is set up for it; people are trained in software updates. It's understood that everything is a work in progress."

A unique lag exists between when a cartridge game is finished and when people buy it. Clyde Grossman notices the difference: "You lose the immediacy of releases with video games. At Sega, I would honestly lose track of what was shipping because we had finished with it

three months before. With a floppy or CD game, it can be on the shelves in two weeks from QA."

The Future

Console and computer games are converging. Today, they share the CD-ROM delivery vehicle. Soon, they will share full-motion video via compression and decompression technology; witness Sigma Designs' ReelMagic PC add-on board and an MPEG adapter slated for release for the Atari Jaguar later this year.

Gregg Tavares ventures the opinion that there are now three categories of electronic games: cartridge, PC, and CD-ROM multimedia running on a PC, Sega CD, 3DO, or whatever. The challenge is to fill up a CD-ROM—with gameplay, not just memory-hungry digitized sound—and access all that data quickly.

Hugh Bowen, of the Bowen Associates multimedia video game marketing consulting firm in Half Moon Bay, Calif., would agree: "The PC CD market is exploding in sales. It used to be minuscule compared to video games. Rebel Assault from LucasArts has sold 400,000 worldwide since last fall. Activision's Return to Zork sold 300,000 in the last six months."

At this year's March Symposium of the Software Publishers Association in San Francisco, Calif., he chaired a panel on multimedia platforms that ranked their attractiveness to developers in the following order: Sega Saturn, PC and Macintosh CD, Nintendo Project Reality, Sony PS-X, 3DO, Jaguar, and Sega CD. In the PC world (including Macintoshes), five to seven million CD-ROM drives are expected to be in American homes by the end of 1994, and 10 to 12 million by the end of 1995; there are already 700,000 Sega CDs.

Nintendo has announced that its Project Reality machines will use high-capacity data cartridges instead of CD-ROMs. Gregg Tavares is skeptical: "Their carts are supposed to hold 150MB to 250MB? Crash 'N' Burn took up 300MB without the animated sequences, and Total Eclipse took 40MB." It's not that hard to fill a CD-ROM.

And Rich O'Keefe recalls his experiences while in Japan for Atari, Electronic Arts, and NEC: "The PC Engine, what we call the Duo over here that plays carts and CDs, sells almost no cartridges any more. CDs are cheaper, and you can get them back overnight—in a week in the worst case. You can gauge the market, take fewer risks, and there's no gambling on inventory."

On the other hand, in the not too distant future: "with a set-top box, there is no reason for CD-ROM: you get it through cable. Cable delivery is going to be much more important in the next 10 years than CD-ROM delivery is now. I think there's going to be this convergence of cable boxes, telephones, PCs, and video games. It's going to be an interesting little mixture. Right now, video games lack two-way connectivity. And you can get that over a cable system at a high enough bandwidth to be truly interactive. Not 9600 baud modems; that is too slow, except for very simple games and strategic games. To do the VR thing, you need high-bandwidth communication."

If you ask Gordon Walton for another reason why he got out of video games, he'll say the console market is about to crash, and the new platforms will arrive too late. "If you are currently in the 16-bit market, that's fine as it trails down. But if you're not developing for these new platforms right now, it's two years before any product comes out for them—3DO, a year and a half."

Robb Alvey would not argue: "A lot of people don't understand that it takes a while for people to develop on and learn a system. When the Sega CD first came out, everybody was saying, 'Oh, this is a dead system. Noone's going to buy it. The games are horrible,' and they were. It always takes something on a system to make me want to buy it. With the Sega CD it was Silpheed. I played Silpheed at a friend's house and said, 'I gotta have a Sega CD now.' "

And he is clearly eager to use what comes along. "For Aladdin [a Sega Genesis cartridge], we had to say, O.K., we've got a limited amount of RAM buffering, and we've got a limited

amount of cartridge space, just depending on how good compression routines are and how much stuff we can cram into a 2MB cartridge as opposed to a 600MB disc. We sat down with the Disney animators and said, 'Alright, you have to create feature film quality or at least the feel of it, given all of our limitations: characters can't have facial expression, and they can't have fingers or toes.' That was a blow to them; they're used to being free with their animation and artwork. But: if we were to do a CD-ROM Aladdin, in the same form that you would do a Dragon's Lair and have the luxury of MPEG out there now, we could have done anything we wanted with the Disney animators. We could have let them run wild and be free with it."

Into Overtime

Whatever comes along, designers and producers and technical directors will be bending it to their creative visions. Paul O'Connor is typical: "As a player, I enjoy computer games more because they provide intellectual challenge, and they have deeper levels of play. But for a racing game or a fighting game, it's hard to beat that oomph of a top-flight video game. I think computer game designers can learn a lot from the interface design of some of the best console games because they can get complex designs across. A person can pick up a controller and understand how to play first time through without reading any documentation. When's the last time you did that with a computer game, short of Wolfenstein?"

"There is a crossover between set-top boxes and PCs; they're growing together," says Michael Perry. "As set-top boxes and PCs grow more and more powerful, it's going to be hard to distinguish between the two platforms. We may end up in the future being able to run a game on a PC and on a set-top box with very few modifications at all."

Looking ahead, David Walker, a technical director at Electronic Arts, says he used to place computer and video games along a continuum of complexity versus simplicity. That is changing with the introduction of full-motion-video and MPEG chips. "We're seeing a new genre of game, the interactive movie. Nobody knows what it means, but everybody is using the term. And it can live on both the PC and set-top boxes." ■

James Paul Cooper sold decision support software to institutional investment managers for a global consulting firm. In 1978, he joined an Arizona computer game company instead of matriculating at the University of California. Since then, he has founded or co-founded several game and software ventures and acquired an M.S. in Industrial Administration from Carnegie Mellon University, where he taught marketing. He can be reached via e-mail at 72147.2102@compuserve.com or through Game Developer *magazine.*

The Fat Man Sings

The Fat Man might not be familiar to those of you who follow the pop charts, but this Dallas-based band has been heard by millions if not billions of game players. From writing three-minute jingles for commercials to nominee for best soundtrack at the March Game Developers conference for Origin's Wing Commander, George Sanger and his team provide music for the game players of the world. The Fat Man is (from left to right) K. Weston Phelan, George Sanger, Dave Govett, and Joe McDermott.

Nestled in Texas hill country, Austin is a city jam-packed with musicians, but it's not necessarily an easy town for a musician to make a living in. In fact, for the amount of money they make, some musicians might as well be playing music as a hobby. George Sanger has gone about the whole business of making music and making money in a completely different way. Now, he's making money playing music, and he's making money at his favorite hobby: playing games. In fact, he and the three other musicians who work with him carry a lot of weight in the game business. They are the Fat Man. "We're huge," says Sanger.

Fat Man central is Sanger's small, tree-shaded red brick house in North Austin. The entire front room is taken up with computers, sound equipment, and Sanger's collection of guitars. Swiveling around in a high-backed blue-padded chair behind a giant desk, Sanger barks into the phone, fiddles with equipment, and, in general, lives large. Sanger's family reigns in the rest of the house, where Linda Sanger manages the household, the couple's two kids, and the Fat Man's books.

Family is important to Sanger. He got his start in the game business when his brother's roommate Dave Warhol, who was writing games for Intellivision, commissioned him to write a 10-second tune for $1,000. The tune was called "Carnival of the Penguins," and it was used in a game called Thin Ice. Since then, games boomed and busted, and they're booming again.

George's brother David Sanger became the drummer for Grammy Award winning Asleep at the Wheel, and George Sanger's career in the game business took a hiatus. As he puts it, "I wasn't writing 12 seconds of music for a thousand dollars anymore, I was writing something like up to three minutes for commercials for $79.95." At that time, Sanger was also running a recording studio for Austin musicians, which is how he met the musicians he now collaborates with. At his brother's wedding, he ran into Dave Warhol again, and, once again, he got a break. Warhol put him in contact with Brian Moriarty, who asked Sanger to create music based on Swan

by Kathleen Maher

Ever wonder who writes those nifty soundtracks, heart-pounding scores, and catchy tunes for your favorite game? Read how one man and his band are shaping the future of music in gaming.

Lake for the LucasArts game Loom.

The next big break and a big evolutionary step for the Fat Man came when Sanger was asked to write music for Origin's Wing Commander. He didn't have time to work on it, so he subcontracted it to Dave Govett who, conveniently, had the kind of grand World War II meets Star Wars theme Origin wanted already floating around in his head. It was something Govett had worked out in high school. Then, Origin producer Chris Roberts made several decisions that made Wing Commander a significant milestone in gaming history and also helped propel the Fat Man into the big time.

Roberts decided to make Wing Commander more interactive with the addition of characters and a storyline. Also, Origin refused to compromise game quality for the lowest common denominator market. Wing Commander required a Sound Blaster, the best commercial sound board available at the time, and it required at least a 386 computer, plenty of room on the hard disk, and a reasonable amount of RAM. It was a huge success. That year, two titles Sanger worked on, Loom and Wing Commander (nominees for best sound track at the Game Developers Conference in San Jose, Calif.; Origin won the category), were Sanger clients, and the Fat Man became "the biggest name in music for multimedia."

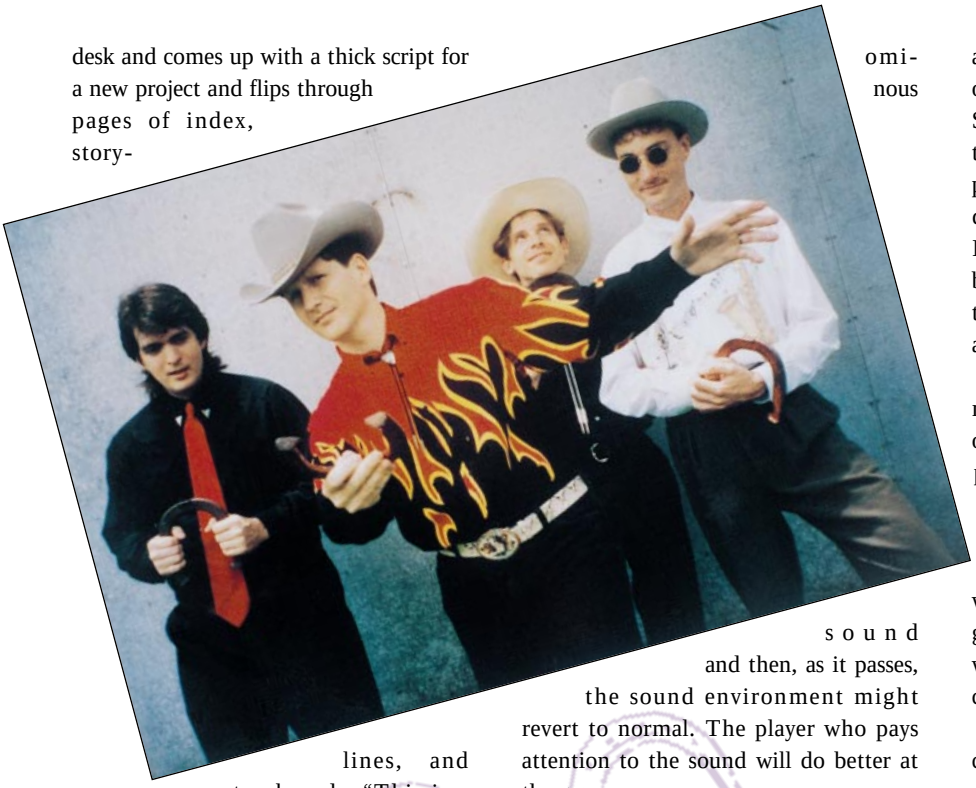
The Style is No Style

Sanger says there is no such thing as a patented Fat Man sound. For one thing, he says, he doesn't have the chops (a musical bag of tricks to fill in any blank), but also because the Fat Man is really four people: Joe McDermott, K. Weston Phelan, Dave Govett, and Sanger. Joe McDermott has produced several albums of children's music. He was also a member of Grains of Faith, one of Austin's best undiscovered bands and, says Sanger, "he's created some of the most ear-bleeding music for Nintendo games."

Kevin Phelan used to run sound for Austin musicians including Asleep at the Wheel, was a member of the punk/funk band One Bad Pig, and he still specializes in the eclectic, weaving world of music with jazz and pop. (McDermott and Phelan are now members of a new band called Caterpillar.) Dave Govett specializes in grand epic themes à la John Williams or Danny Elfman. As for George Sanger, the original Fat Man, he says his job is to bring it all together. "I specialize in music salad. I don't ever just play what's under my fingers because there's nothing there."

Sanger says he does not play keyboard well, and he deprecates his guitar playing talents. "What I have," he says, "is the ability to drink a beer and lead a jam so that everyone has fun. On the keyboard side, I have the ability to feel an emotion and slowly and painstakingly sculpt it out on the keyboard and the computer. Also, I have the ability to coach other folks through the process of performing it."

Depending on the developers, the Fat Man can enter a project at any stage. In the case of IndyCar Racing, the game was already completed, and the Fat Man was hired by the developers at Papyrus to come up with a theme. To illustrate how early he can come into a project, Sanger reaches across that broad expanse of a



desk and comes up with a thick script for a new project and flips through pages of index, storylines, and storyboards. "This is a little more than we need, but it gives us the fall and lay of the land." Obviously, Sanger and the Fat Man would rather come into the project as early as possible.

In the case of Putt Putt goes to the Moon, the Fat Man worked with the developers at Humongous Entertainment every step of the way. "They would send us a new update of the game with the music integrated. That was a wonderful way to work because I didn't have to work with any fancy technology at all. I didn't have to use any screwy tools and kill my computer by loading up prototype software. All I had to do is put the game on and see how it was playing so far and send them feedback."

The sound environment for Electronic Arts' SSN-21 Sea Wolf created in collaboration with John Ratcliff is one Sanger and his team are most proud of because it uses sound in such an innovative way. "I wish I had been even more involved in creating sound for that one," says Sanger wistfully. All the sound associated with a real submarine has been recreated and helps create a three-dimensional sound environment. Sound is also used to lead and misdirect the player. For instance, if some threat passes by the submarine, the music might change to an ominous sound and then, as it passes, the sound environment might revert to normal. The player who pays attention to the sound will do better at the game.

Sanger is also pleased with the effect of the music for 7th Guest, since the music throughout most of the game is not stereotypical "scary" music but rather cool jazz. In this way, Sanger hoped to heighten the effect of the game by working in a counterpoint to the game's ostensible horror aspects.

Methodology in the Madness

An interesting element of writing music for games is the structure of the game. In general, says Sanger, the team is asked to write a few themes: you win, you lose, you wander around. Consideration has to be given to what happens when the player changes course. In Wing Commander, the style of the music allows changes to occur without too much distraction. It draws on the conventions of early filmmaking that also had the same problem in terms of cutting film together. Therefore the music is a pastiche of bombastic warlike themes and lyrical transition passages. Likewise, 7th Guest's jazzy style lends itself well to changes in tone that, while sometimes abrupt, are not unduly distracting.

In contrast, game developers at Origin are taking a different approach. They are going for a Disney model that relies on in-house talent. At the South by Southwest music conference in Austin this year, Richard Garriot chronicled the progression of sound in his titles and described a system called NIM (Neno's Intelligent Music) developed at Origin by Nenad Vutrinic that incorporates a transition theme to move the user, the action, and the sound to a new course.

As Garriot describes it, when a user makes a decision or chooses a new course of action, the music plays to the next bar, picks up the transition theme, and moves to the theme appropriate to the user choice without any jarring shift in mood. The difficulty of working this way, according to Randy Buck, a programmer at Origin, is that instead of working in a linear fashion, music is composed in a matrix.

For his part, Sanger believes fast changes emphasize the player's control over the game, and therefore he prefers it, though he's also investigating other approaches. "Transitions are cool," he says, "if the composer writes them. But if they get into the hands of the programmers, they can lose some of their musical impact." The Fat Man can write transitions, and the members are discussing the development of tools to make it easier to write music in a matrix as Buck describes it. "We want to make sure we're not getting lazy and fat. Just fat."

Sanger believes his success in the computer game business is directly related to his liking for games. He can identify with the player of a game he's working on. "Some poor schmuck is stuck in a room with my music, and I want it to be good music so they don't hate me." Many musicians, he feels, hold games in contempt, and they're not interested in writing music for games because of the limitations of the hardware and software.

Sanger seems to welcome limitations as a challenge. Perhaps that's why he's such a game player. No doubt learning from Origin's experience with Wing Commander, Sanger encouraged the developers of 7th Guest to write to General Midi standards, giving the game an extension on life as sound boards adopted the new standard and actually

improved the sound of 7th Guest.

Nevertheless, given present levels of development, a sound board in a computer can only sound so good, and not all sound boards, even those that conform to General Midi standards, sound alike. Sanger says that in his early work with companies such as Trilobyte, LucasArts, and Origin, the developers would take his music and optimize it for all sound boards. "I thought I was really good," says Sanger. But then, on another project, Sanger was horrified to start up the brand new game and "the music literally sounded like farts, squeaks, and beeps, and there was my name—music produced by the Fat Man—and all of a sudden I realized I have got to get into the sound business."

Sanger and The Fat Man are entering the sound business in a couple of ways. Not content to simply supply musical themes for game players, the Fat Man and primarily Kevin Phelan are developing a library of tones to ensure that games players, developers, and musicians are working with the same basic elements. Yamaha has licensed the Fat Man's General Midi patch set for the opl2 and opl3 FM synthesizers, which are the heart of Sound Blasters. The Fat Man is working with Yamaha on the development of drivers to be included with boards using the opl4.

The Sound Business

Also, the Fat Man has become a consultant. Board manufacturers can hire the Fat Man and win a seal of approval that promises users the boards will produce the proper tones for specific instruments and make no demands on the CPU. For, as far as Sanger is concerned, if the sound board slows down the CPU, it can stop a game dead without one explosion, dead Nazi, or passage explored.

In addition to ensuring that computers play the game and music correctly, the Fat Man's entrance into the realm of standards and hardware has another

> "Some poor schmuck is stuck in a room with my music, and I want it to be good, so they don't hate me."

advantage. “We get our name plastered all over everything. If there's not something in it that feeds my ego, I just can't get out of bed in the morning.” In other words, says Sanger, “I'm doing the right thing for the wrong reasons, and that's what I like to do.”

Sanger now composes on the Roland Sound Canvas, and he encourages other composers to use it. Not only does he believe that it is the best wave table synthesizer (“so far”), it is now the de facto standard that gives manufacturers a common ground to work on. If manufacturers develop products with the expectation that most composers will be composing on the Sound Canvas, the sound balance will work as they intend it. “I'm trying to turn the tide on this [the proliferation of sound boards and varying standards]. Instead of having software developers burdened with having to support lots of different cards, I want the hardware companies to support the way developers are writing music.”

Turning the Tide?

Sometimes when The Fat Man players get together, they play surf music. When George Sanger was asked to perform at the Interactive Conference in San Jose, Calif., this year, he wasn't able to bring the rest of the team with him. So, he brought his guitar and got together with keyboard player Dave Javelosa of Sega, bass player Michael Land of LucasArts, drummer Neil Grandstaff of Sierra On-Line, guitar player Dave Albert of Sega, saxophonist Albert Lowe (who is not only Sierra On-Line's first composer, but also the man who created Leisure Suit Larry), and independent composers Jim Donofrio on guitar, and Don Griffin on trumpet. And yes, they played surf music.

It's appropriate. George Sanger is one of many trying to stay on top of a very quickly changing market. He's doing it by branching out rather than protecting his turf. “Every time I give something away I get more back.” Enthusiastically describing the jam in San Jose, Sanger confesses to being initially nervous about meeting and working with competitors but, as it worked out, “I made some very nice friends and cemented some very good relationships.”

Likewise, Sanger admits, “I was nervous when I started giving jobs to the guys on the team, but that's what turned it into a team instead of just one composer.” Staying on top, staying ahead of the trend, that's surfing. It's that ability, over and above songwriting, that has allowed George Sanger to make money at something he loves. ■

Kathleen Maher is a freelance writer in San Francisco, Calif., specializing in media and culture. She is also the executive editor of Cadence Magazine. *But, Kathleen's true talent may be her ability to find the best barbeque in Texas and Lucinda Williams on the jukebox. She can be reached via Internet at kmaher@mfi.com or 71154.76@compuserve.com.*

The Return of A Legend

by Andrea Pucky

New releases. New versions of old releases.

Books, gameware, tips from the gurus. All this

and more in a roundup of what's hot in the game industry

right now.

In the past, game players faced quite a dilemma. I'm sure you know what I'm talking about. You're standing there at the counter, preparing to plunk down your hard-earned money for a video cartridge. Only one thing left to decide. "Do I want a game I can play at home? Or do I want a game I carry with me and play when I'm stuck in traffic?" A minute seems like an eternity. You finally make what seems to be an intelligent choice, but suddenly everything's in black-and-white, and Rod Serling is saying something about some poor sap entering the Twilight Zone. Hey, wait! No need to sweat or hallucinate any longer.

Donkey Kong is back in a Game Boy cartridge that goes from hand-held black-and-white to a full-size, color game.

Nintendo has announced the release of Super Game Boy, which transforms Game Boy cartridges into full-size, color video games. Super Game Boy is a Super Nintendo Entertainment System (NES) cartridge that contains a Game Boy adapter. When you insert a Game Boy cartridge into Super Game Boy, it is transformed from a 2- by 2-inch, black-and-white game to a bright, multicolored image on a big television screen. It even has stereophonic sound. In other words, you can now play over 350 Game Boy cartridges on Super NES.

Super Game Boy displays all Game Boy action in varying shades of four colors. And, as if this instant colorization of black-and-white Game Boy cartridges isn't enough to make Ted Turner green (with envy, that is), players may customize several prestored color palettes to change the appearance of the screen and may place different, animated, decorative borders around the central game-play screen. Using a Super NES control or a Super NES mouse, players either select one of several prestored designs or create their own through a paint-type program, drawing not only on the borders but on the game-play screen itself.

New titles specifically designed for Super Game Boy will be able to display up to 256 colors. All of you popular-culture theorists out there know what the first new Game Boy title to capitalize on Super Game Boy's capabilities will be. Yes, it's time to relive the magic once again—Donkey Kong, bigger and badder than ever, will be released at the same time as Super Game Boy. That's right, the big, hairy ape returns to video game screens in a set of new adventures.

In the new Donkey Kong, Mario (I think we can safely say that the poor guy has some really bad karma) must once again fend off the advances of his old nemesis, Donkey Kong, in an effort to rescue damsel-in-distress Pauline. Mario returns first to the familiar surroundings of a construction site complete with ladders and speeding barrels. Then, he must chase down the gorilla through a variety of urban obstacles as Donkey Kong is let

loose in the Big City. Okay, it's not exactly the same game we remember from the early '80s, but still, it seems that Donkey Kong is well on his way to closing the generation gap. (Hey, at least I didn't say anything about the "missing link" or the *Planet of the Apes*.)

More games will soon be on the way, as Nintendo's licensees and game developers will receive development specs immediately. According to Nintendo, Super Game Boy will be available June 6, at a suggested retail price of $59.99

For more information contact:
Nintendo of America Inc.
4820 150th Ave. N.E.
Redmond, Wash. 98052-5111
Tel: (206) 882-2040

Push Your Sega to the Limit

And now, the winner of the "God, I wish I could upgrade my car that easily" category is Sega of America Inc. How would you like to upgrade your 16-bit hardware to 32-bit and get arcade-quality game play for less than $150? Sega of America can help you do just that. Sega has announced introduction plans for the Genesis Super 32X hardware upgrade, allowing video game fans to get 2 by 32-bit arcade-quality game experiences from their existing 16-bit Genesis hardware.

Genesis Super 32X is the first product from Sega that will use the Hitachi SH2 RISC chips destined for Saturn. (No, that's not some new conspiracy theory. It's the code name for Sega's future hardware platform.) The two SH2 chips in Genesis Super 32X will complement a newly-designed video digital processor (VDP) chip to bring to the Genesis the fast processing speed, high-color definition, texture-mapping, improved computer polygon graphics technology, ever-changing three-dimensional perspective, software motion video, enhanced scaling and rotation, and CD-quality audio that gamers have come to expect from arcade machines and the most advanced home systems technology on the market.

Genesis Super 32X will enhance both Sega CD discs and Sega Genesis cartridges designed and developed to incorporate this new technology. Consumers can still play the more than 500 games available for the Sega Genesis and the more than 100 games available for the Sega CD, while the Genesis Super 32X is attached to the Genesis hardware unit. In other words, it's an instant upgrade.

Sega has more than 30 titles under development and expects its software licensees to add a similar number in the first year of the new product's introduction. Titles playable on Genesis Super 32X are expected to be priced for consumer sale at levels comparable to current software prices for Sega's home systems. According to Sega, the Genesis Super 32X hardware booster will be available in the fall of 1994 and will carry a recommended retail price of $149.

For more information contact:
Sega of America Inc.
130 Shoreline Dr.
Redwood City, Calif. 94065
Tel: (415) 508-2800

The Wise Ones Tell All

Guru: a personal spiritual teacher or a recognized guide or leader. (Just a note to any hip-hoppers out there. We're not talking about Gifted Unlimited Rhymes Universal.) I think we can safely say that if you're going to call yourself a guru, you really should know what you're talking about. Although the title may simply be a marketing ploy, the authors of *Tricks of the Game Programming Gurus* really do have the experience and knowledge to fit the definition of guru.

Ken Allen is currently programming with Spectrum Holobyte and is on the development team of its latest combat flight simulator. André LaMothe, who specializes in graphics and animation, has been employed by Visions of Reality, Alligator Communications, Concurrent Logic, Versasoft, and NASA Ames (RIACS). Graeme Devine, who is writing a chapter on multimedia for the book, is the president of Trilobyte, makers of The 7th Guest and the upcoming The 11th Hour. Devine also designed and programmed the arcade version of Atari's Pole Position while in high school. (Does that make him a guru or a genius?)

The book provides the building blocks necessary to create your own interactive games. It explains the basic and advanced ideas and topics behind the development of a flight simulator, a three-dimensional walkthru game, and many utilities used to manipulate video, audio, and input devices. The book also includes a foreword by John Carmac, Technical Director of ID Software, makers of Castle Wolfenstein and Doom.

Tricks of the Programming Gurus comes with a CD ROM that includes all the source code from the book, shareware games, commercial software demos, and utilities for game design and image manipulation, including paint/texture software, sound editors, sprite editor, screen map designer, Castle Wolfenstein, Doom, and The Commander Keen series from ID software.

The book includes: building an authentic flight model, the physics of flight, graphics behind the simulator, aircraft avionics, building a user interface, assembly language basics, I/O basics, two-dimensional graphics, VGA card manipulation, three-dimensional space, bitmap graphics, sound and effects, game structures, synthetic intelligence, user-defined graphics, adding multiplayer capabilities, and testing the game. We can only hope that these gurus have a few more tricks up their sleeves for their next book.

Tricks of the Game Programming Gurus will be released in July 1994 and will be priced at $49.95.

For more information contact:
SAMS Publishing
Prentice Hall Computer Pub.
201 W. 103rd St.
Indianapolis, Ind. 46290
Tel: (317) 581-3500

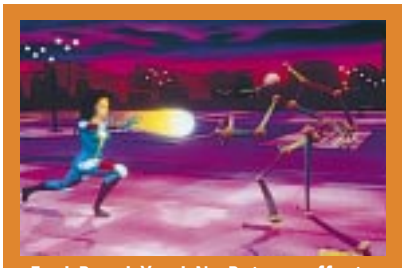
Practically Total Reality

Are you an unsatisfied game developer? Yes, even game developers get the blues. Wavefront Technologies may not be able to offer you a magical cure-all, but they have introduced GameWare, a software package designed to satisfy the demands of game developers. GameWare runs on Silicon Graphics workstations and, according to Wavefront Technologies, provides developers with

graphics tools for creating games with realistic three-dimensional objects and terrain, three-dimensional synthetic actors with realistic motion, and stunning special effects.

GameWare addresses the color bandwidth problem of game consoles by allowing the developer to render to reduced color palettes. It resolves geometry bandwidth problems by providing three-dimensional geometry reduction algorithms and a "flatten" tool that converts a three-dimensional object into an identical two-dimensional version. Open architecture allows developers to integrate their existing software tools with GameWare to an extent that is not possible with systems based on a proprietary architecture and data structure.

In addition to modeling, animation, and rendering capabilities, GameWare provides synthetic actor animation by integrating forward and inverse kinematics with skin behavior. For those of you who haven't brushed up on the jargon lately, animators can bring a static model to life by directly manipulating a user-defined skeleton like a marionette. To maintain a natural form as the skeleton moves, GameWare incorporates a behavioral model, called Smart Skin, that can be taught to behave according to skeletal position. Clothes can be taught to bunch up as a character flexes



Zap! Bang! Yow! No Batman effects here—Hyper-Plug-ins create realistic, natural images for maximum animation.

its arm, for example. Wavefront Technologies says that GameWare combines everything character animators need—three-dimensional kinetics motion, skeletal intelligence, and skin behavior—in an intuitive system.

GameWare Hyper Plug-ins are available to provide advanced levels of special effects, compositing and painting. GameWare Dynamation is a Hyper Plug-in that uses the principles of physics to ensure that special effects are highly realistic. Dynamation allows users to create realistic, natural images of dynamic events. The animator

Live video of an office interior, combined with computer-generated animation, creates a new horizon in special effects.

defines the laws of a particular dynamic world and releases objects into it. (Don't you wish you had this to play with in your high school physics class?) The software can create realistic effects so that players in a race-car game, for example, can drive through a realistic bank of fog or a burst of exhaust from the car ahead. Dynamation comes with a library of prebuilt effects, called Clip F/X, that makes it possible to create physically based phenomena, such as explosions, cloth modeling, terrains, and plants. Other Hyper Plug-ins are GameWare Composer, GameWare Paint, and GameWare Hidline/Reline.

GameWare is available for all SGI workstations, including the entry-level Indy. It is priced from $15,000.

For more information contact:
Wavefront Technologies Inc.
530 E. Montecito St.
Santa Barbara, Calif. 93103
Tel: (805) 962-8117

Director, Director!

So, have you guessed the leitmotif? If you guessed upgrade, you're right. Macromedia recently announced (what's the secret word of the day?) an upgrade to Director for Macintosh, an authoring tool for creating multimedia productions. Here's what's up with the new version.

The scripting language, Lingo, is compiled for faster script execution, and the improved memory management provides developers with control over removing cast members from memory. According to Macromedia, the new version optimizes data retrieval from storage devices and a revised file layout reduces disk access time. Plus (yes, there's more), within Lingo, new object-oriented commands provide reusable code and simplified scripting. The new "movie in a window" feature allows multiple Director movies to play at the same time while each individual movie maintains full interactivity.

Further, within the Score, the number of channels increases to 48, doubling the number of interactive elements on the screen at one time. The number of case members per movie has increased significantly from 512 to 32,000, and the file size has grown from 16MB to an unlimited size.

Director 4.0 has a few more improvements worth mentioning here. A new security feature eliminates the viewing of proprietary Lingo code. Color has been added to the Score (okay, maybe the leitmotif is Ted Turner) to help organize and track cast members within a production, and users can drag-and-drop cast members directly from the Cast into the Score for faster animation creation and production development. Version 4.0 has a binary-compatible file format with the soon-to-be-released Director for Windows, providing cross-platform authoring and playback.

But that's not all! You also get step-by-step tutorials, sample movies, on-line help, and Lingo Expos, which highlights commonly used Lingo in real-world examples.

Macromedia's suggested retail price for Macromedia 4.0 is $1,195. The really good news is that all registered users of Director (versions 2.0, 3.0, or 3.1) or even its predecessor, VideoWorks, may upgrade to Director 4.0 for $199. (But you have to do it before August 31, 1994; starting September 1, 1994, the upgrade will cost $249.)

For more information contact:
Macromedia Inc.
600 Townsend St.
San Francisco, Calif. 94103
Tel: (800) 457-1774

Sounding Out the VESA Audio Standard

by Jon Burgstrom

Worried about how your game will sound? With the advent of the VESA Audio Standard, you're game will sound the same on any system—read on to learn the ins and outs of this ground-breaking standard.

The latest development in the PC audio arena is the standardization efforts of the Video Electronics Standards Association or VESA. This standardization effort called the VESA VBE/AI (VESA BIOS Extension / Audio Interface) has recently been approved. Initially, this standardization effort was brought about by the requests of the entertainment industry that creates products for the DOS environment.

At the April 1993 Game Developer's Conference in Santa Clara, Calif., a group of prominent entertainment software developers held a meeting to discuss the problems in developing audio support for game titles. The quagmire of existing audio hardware architectures and the introduction of new ones, all of which compete for attention, was more than most developers and their companies could afford to bear. In some cases, adding support for each audio board represented about 15% to 20% of a total game development project in staff resources.

Until recently, audio standards for the PC consisted of a loose interpretation of an old industry standby: Creative Labs first Sound Blaster architecture. Based on 8-bit digital mono audio and two-operator (OPL2) FM synthesis provided by Yamaha, this pseudo-standard has been in place since the late 1980s. However, feeling the pressure of consumers, most developers couldn't afford to support just this one simple architecture.

It is not uncommon to find 16-bit CD-quality stereo digital audio and high-quality digitally sampled instruments replacing the lower quality and performance of the original Sound Blaster architecture in PCs. These higher performance and usually more advanced feature set PC audio add-in boards have met consumer expectations, but with a higher price tag. Clearly a standardized audio API for the entertainment industry is required to endure the explosive growth of the past two years.

The Standard Takes Shape

The VESA VBE/AI work group leader is Rick Allen of Media Vision Inc. The group consists of 30 representatives from PC audio hardware, PC entertainment software, and other interested companies. Although the group has been around for less than a year, its first specification passed VESA general member approval in Feb. 1994. This quick turnaround from concept to approval was a great achievement, as anyone familiar with industry committees can attest. More importantly, this speed emphasizes the PC audio industry's need to have such a standard in place—now.

The VESA VBE/AI specification provides for basic audio services, such as:

- Digital (WAVE) audio (8 or 16 bit, mono or stereo)
- MIDI (not just FM synthesis)
- Volume control
- Minimal three-dimensional sound effect positioning.

The standard is targeted at DOS platforms or DOS-box environments if the application is run from Windows 3.1, OS/2 2.x, or Windows NT through virtual services such as a VxD,

Figure 1. First-Level Services for the VBE/AI Audio Interface (Continued on p. 30)

INT 10 subfunction #0
Check if a VBE/AI compliant device driver is loaded.

Parameters:

AX = 0x4F13	The VESA assigned audio interface ID (0x13).
BX = 0x0	The VBE/AI subfunction number.
CX = 0x0	Reserved for future use.
ES:DI = 0x0	Reserved for future use.

On return:

AX != 0x004F	Used if the call was not successful.
AX = 0x004F	Used if the call completed without error.
BL =	The VESA VBE/AI driver version in packed nibble format, where the high nibble is the major version and the low nibble is the minor version.
CX =	Undefined.
ES:DI =	Unchanged.

Note: BH, DX, and SI are undefined and assumed to be trashed.

INT 10 subfunction #1
Locate a handle to the device.

Parameters:

AX = 0x4F13	The VESA assigned Audio Interface ID (0x13).
BX = 0x01	The VBE/AI subfunction number.
CX = 0xn	Initially set to 0 for the first call, subsequently set this to the handle of a device class instance that is returned for future calls.
DL = 0xn	Device class ID or 0 for all device class queries. Possible values are: 0x00 All device class types. 0x01 WAVE audio device. 0x02 MIDI device. 0x03 Volume control device.

On return:

AX != 0x004F	Used if the call was not successful.
AX = 0x004F	Used if the call completed without error.
CX =	If CX != 0, it holds the handle to a valid device class instance. This handle will be used for other INT 10 and VBE/AI function or subfunction calls to the device. If CX = 0, there are no more devices available that fit the specified device class.

Note: BX, DX, and SI are undefined and assumed to be trashed.

INT 10 subfunction #2
Query the device for information.

Parameters:

AX = 0x4F13	The VESA assigned audio interface ID (0x13).
BX = 0x02	The VBE/AI subfunction number.
CX = Device handle	As returned from subfunction #1.
DL = Query number	A total of six types of queries are possible: 0x01 Get the length in bytes of the general device class structure in SI:DI. 0x02 Return a copy of the general device class structure pointed to by SI:DI. 0x03 Get the length in bytes of the volume information structure for the device in SI:DI. 0x04 Return a copy of the volume information structure for the device in SI:DI. (DX is NULL if the device class doesn't have a volume control. 0x05 Get the length in bytes of the volume services structure for the device in SI:DI. (DX is NULL if the

which has yet to be implemented.

These audio services are provided by a VESA VBE/AI compliant device driver contained in a ROM BIOS, loaded from the user's CONFIG.SYS file, or, a DOS terminate-and-stay resident (TSR) program supplied by the audio board or other third-party manufacturer. The latter method will initially be the widest implementation of the standard.

The device driver or TSR provides a hardware abstraction layer (HAL) that keeps the messy particulars of registers, port addresses, and other hardware implementation details abstracted away from the primary issue of providing audio support. This layer has been kept as thin as possible to provide for the best performance.

The architecture that provides these services is very robust in its ability to support such diverse audio hardware implementations as standard plug-in boards, such as the Sound Blaster, Pro AudioSpectrum, and so on, and the more unusual audio hardware that is typically connected to external ports such as Media Vision's Audio Port or Disney's Sound Source. In fact, the VBE/AI audio specification will allow upcoming hardware expansion interfaces such as PCMCIA and PCI to have a supported software base without providing any fragile hardware or software alteration to achieve compatibility.

Implementation Issues

The VESA VBE/AI architecture has been implemented using two functional levels. The first level consists of the general API for device interrogation, acquisition, and termination. An application uses the VESA `INT 10` services software to gain access to this level of functionality. The second level, `FAR`, calls to the device's API services, using the Pascal calling convention.

Once a device is opened, the application is provided a list of entry points to functions contained within the VBE/AI-compliant device driver. Since it is possible that multiple instances of audio hardware may exist in one machine, the VBE/AI allows for multiple instances of each device class API and defines one unique API for each device type, where each device type is defined as a device class. Each device class API may be implemented as an individual device driver or combined into one device driver. The organization of device driver and device class API issues is still kept completely transparent to the application.

To gain access to the device driver, which is assumed to be preloaded in memory, your application will perform the following functions. This is the only level that does not use the Pascal calling convention. All parameters are passed in CPU registers, and all calls to this level of the API are executed with the software interrupt instruction, so all returns are done using an `IRET` instruction.

All devices will chain into a linked list of VBE/AI drivers found by the VESA `INT 10` services. The linked list approach allows for an almost unlimited number of VBE/AI drivers to be loaded at any one time. Figure 1 shows the parameters and return values for these subfunction calls.

Step 1. Perform an `INT 10` subfunction `#0` to see if a VBE/AI compliant device driver is loaded. This subfunction lets you initially test to see if VBE/AI services are available to your application. One of the most common technical support calls answered by sound board manufacturers is about the configuration of the user's audio hardware for each entertainment product he or she installs.

Remembering DMA, IRQ, and port address settings, much less their meanings, can be absolutely maddening for the novice or even intermediate PC user, causing them to scurry around looking for a long since misplaced manual. Many developers have realized that it's unwise to depend on environment

WE DON'T JUST DO VIDEO ANYMORE

VESA, once known only for its PC graphics standards, hence the original name of Video Electronics Standards Association (with the majority of its members making up the best-known people in video hardware manufacturing), has faced similar standardization problems before. As with the original Super VGA standardization effort, lower-level standards, such as IBM's VGA and 8514a standards, previously existed.

The standardization problem surrounded the implementations of other modes such as 800-by-600, high color, and true color by several video card manufacturers. These new extended or Super VGA modes had to be addressed for them to be useful on a broad basis. The VESA committee created what is now commonly referred to as the VESA VBE (Video BIOS Extensions) specification to standardize the programming interface to these extended modes.

Recently, the VESA membership has grown to cover all aspects concerning the PC. VESA has tackled several standardization efforts; the VESA Local Bus or VL Bus is probably the most recognized VESA standard by consumers. VESA has evolved into a capable standardization forum that provides manufacturers with an arena to compete in, while consumers can purchase compatible hardware peripherals as a result of these standards.

To acquire the written VESA BIOS Extension / Audio Interface (VBE/AI) v. 1.0 document contact:

Video Electronics Standards Assoc.
2150 N. 1st St., Ste. 440
San Jose, Calif. 95131-2029
Tel.: (408) 435-0333
Fax: (408) 435-8225

Figure 1. First-Level Services for the VBE/AI Audio Interface (Continued from p. 28)

device class doesn't have a volume control.)

0x06 Return a copy of the volume services structure for the device pointed to by SI:DI. (DX is NULL if the device class doesn't have a volume control.)

SI:DI = Points to a buffer of n bytes in length.

Note: The application is responsible for allocating the correct buffer size to hold the return values. The size of the various structures can be found by using queries 0x01, 0x03, and 0x05.

On return:

The most important query is 0x02. This query returns the general device class structure in the buffer you allocated and is pointed to by SI:DI. Following is the structure of 0x02:

```
typedef struct {
    // Housekeeping...
    char    gdname[4];                  //  Name of the structure.
    long    gdlength;                   //  Structure length.
    //  Generalities....
    int gdclassid;                      //  Type of device.
    int gdvesaver;                      //  Version of VBE/AI implementation.
    union   {
            WaveInfo            gdwi;
    }  u;
}  GeneralDeviceClass,    far    *fpDC;
```

Note: char = 8-bit byte, int = 16-bit word, long = 32-bit word. The union may also contain structures of type MIDIInfo and VolumeInfo.

INT 10 subfunction #3
Open the device.

Parameters:

AX = 0x4F13 The VESA assigned audio interface ID (0x13).

BX = 0x03 The VBE/AI subfunction number.

CX = Device handle As returned from subfunction #1.

DX = 0x0. For the device drivers 16-bit interface and the only one currently supported.

SI = Points to a segment of memory of the size required by the device driver. The memory offset is assumed to be 0.

On return:

AX != 0x004F Used if the call was not successful.

AX = 0x004F Used if the call completed without error.

SI:CX = If SI:CX != 0, it contains a far pointer to a structure of service functions. If SI:CX = 0, the requested device is unavailable.

Note: BX and DX are undefined and assumed to be trashed.

INT 10 subfunction #4
Close the device

Parameters:

AX = 0x4F13 The VESA assigned audio interface ID (0x13).

BX = 0x03 The VBE/AI subfunction number.

CX = Device handle As returned from subfunction #1.

On return:

AX != 0x004F Used if the call was not successful.

AX = 0x004F Used if the call completed without error.

variables that are easily deleted, potentially contain misinformation, or are not loaded into memory based on the user's environment space configuration.

If VBE/AI services are available, your application can take advantage of the fact that the device driver handles these issues, so there is no need to provide an audio hardware setup screen with DMA, IRQ, and port address choices to confuse users. Developers will probably want to display the manufacturer and product name as part of the system configuration or information screen to reassure users that their installed hardware has been located and is supported.

Step 2. Perform an `INT 10` subfunction `#1` to locate a handle to the device. A handle as defined in the VBE/AI specification is a number that represents each device class API instance. A typical audio board that has digital audio, MIDI, and volume and mixing control capabilities will provide three handles. Each handle represents the individual capability: handle 1 represents the first occurrence of a digital audio device class API, handle 2 represents the first occurrence of a MIDI device class API, and so on.

Step 3. Perform an `INT 10` subfunction `#2` to query the device for information. This subfunction allows you to interrogate and, in certain instances, set values for an installed device class. Your application uses this query subfunction on a device class to find out its capabilities without actually opening the device.

For example, the title you are creating may use the latest in voice recognition technology, which, of course, requires the recording capabilities of a digital audio device class API. It is possible that the user's machine is equipped with audio hardware providing only a MIDI device class API. This subfunction provides the easiest way for your application to find out and provide the user with the necessary feedback.

Step 4. Perform an `INT 10` subfunction `#3` to open the device. When your application opens a device class API, it becomes the exclusive owner of the device until the application closes it. Types of device class APIs normally opened by your application are digital audio (WAVE) and MIDI. Volume devices are not usually opened, as their information and service structures can be acquired via subfunction `#2`.

Your application is not allowed the exclusive ownership of volume devices, as the user may override your settings with a volume TSR program or other mechanism. When selecting a device, the application should base the selection process on these criteria:

- Does the device have the right feature set?
- Does this device have the highest user preference?
- Is this device currently available? (That is, does another application already have this device opened?)

Step 5. Perform an `INT 10` subfunction `#4` to close the device when your application finishes. It is very important that your application close all the device class APIs it opens. Unless your application does so, the memory the device class API allocates is never freed and is still used by the device class API.

After this subfunction is called, which depends on whether the device class API is a digital audio or MIDI device, the device class API will stop playing digital audio or turn off all voices, free the DMA, and clear any pending interrupt requests, return all queued buffers to the caller, or free any of the applications patch data memory blocks.

These steps constitute just a small sample of how to check for VESA VBE/AI compliant devices. For information on how to use the capabilities of VBE/AI, a copy of the specification and software development kit is available; these details go beyond the scope of this article.

Changes on the Horizon

The entertainment software industry is one of the fastest to evolve in technology. Developers must be competitive on all levels of an entertainment title's design, including storyline, graphics, playability, and audio. Every aspect of a potential successful title pushes the performance of PC hardware to its limits (and sometimes beyond) to vie for consumer dollars. Many developers have been forced to forsake the long-faded 8088 markets and, in some cases, 286 compatibility in their products, along with the limitations of the DOS 640K memory barrier to compete for market share.

The use of DOS memory extenders teamed with a fast 386 or better processors has pushed game development ahead into the protected mode environment. Protected mode enables developers to program in a flat 32-bit address space, allowing for multimegabyte entertainment titles that do not require users to have a proficiency memory management and pushing the PC's performance to new levels.

Clearly, protected mode is the wave of the future as 386 and 486 PC systems continue to proliferate the installed base. Some examples of protected-mode entertainment titles are Id Software's Doom and LucasArts' Rebel Assault. These cutting edge products would have suffered tremendous performance hits if they were required to run under real mode; it's even possible they may not

VESA AND PLUG-N-PLAY

VESA VBE/AI and Plug-n-Play is a combination that can't be beat. This architecture is timely, as most manufacturers will eventually implement their device driver in read-only memory (ROM). As the advent of another noncompeting standard, the Microsoft and Intel Plug-n-Play specification for automatically configuring expansion boards into a PC environment (which is bolstered by the existence of a device driver in ROM) has also recently been completed. These two factors could drastically reduce costs associated with entertainment software and audio hardware technical support, adding to the profitability of software titles and fueled hardware sales.

have been created at all.

A potential performance problem exists with the current VESA VBE/AI standard when used in conjunction with a DOS extender, which does not have native protected mode support. Hooks are already in place to provide such support, but the standard's current version relies on a device driver or TSR operating in a real-mode DOS environment. The VESA VBE/AI standard is usable under protected mode, but at the cost of some performance, a major benefit in selecting a protected-mode environment. Developers must choose carefully between performance, compatibility, and technical support requirements. Titles that demand the highest performance may require an alternative to the VESA VBE/AI standard until the VESA VBE/AI group addresses this issue.

Others have switched to the convenience found in Microsoft's Windows Multimedia APIs. This environment contains the needed hardware abstraction layers to provide compatibility for the audio portion of its products. Most of the more demanding entertainment titles require greater response from the PC and cannot afford the overhead of a graphical shell operating system to achieve the playability users demand.

Microsoft is addressing this issue and developing an environment better-suited to an entertainment developer's needs. But, as is always the case in Windows, applications will not be exclusive to the users of this operating system, with the penalty being giving up CPU cycles to other tasks running within this environment.

Third-Party Alternatives

For years, third-party vendors have offered sound libraries and tools to help entertainment title developers support the myriad array of audio choices for the PC entertainment market. Help to produce sound from everything from the PC's loathsome speaker to audio expansion peripheral boards is available with varying degrees of support and cost. Some require the purchase of a one-time licensing fee, others require a fee based on a per-title or per-piece basis or both.

What do you get for your money? A survey of the three top sound-library providers includes products such as John Miles' AIL (Audio Interface Library) and The Audio Solutions' DIGIPAK and MIDPAK (whose proprietor, John Ratcliff along with John Miles have been in the entertainment title creation business for many years). Filling out the top three is Human Machine Interfaces Inc.'s Sound Operating System (SOS).

In most cases, these products outstrip the basic sound services provided by the VESA VBE/AI standard. Some have very robust APIs, allowing developers to perform on-the-fly digital mixing of wave forms. Others have top-rated MIDI support, such as John Miles' AIL, and all have reasonable hardware support for the current installed base of audio boards.

Each company's product has its own strengths and weakness. A good examination of each product and the company that supports it will most likely yield the results required of just about any project. Most companies, along with providing native protected mode support, plan to add or have in place VESA VBE/AI support, giving developers alternatives to their products longevity path.

An example of putting the flexibility offered by third-party sound libraries that incorporate VBE/AI support to use is a recent trend in the entertainment software business to increase individual product revenues and gain as much advantage from research and development costs as possible. Many publishers are releasing slightly updated or revamped older top-selling titles bundled into a single low-cost high-profit package and provided on CD-ROM.

These products are usually geared toward consumers entering the PC multimedia marketplace for the first time or are used for seasonal promotions, such as the holiday season or summer's usual sales doldrums. These strategies provide a longer-term sales path that incorporates the current installed base of audio hardware along with VESA VBE/AI support for future audio hardware without the need to retouch the title's audio portion for updates.

Developers who want to provide the broadest level of audio support and reduce their customer's technical support issues will find the VESA VBE/AI specification a welcome addition. Third-party sound libraries that support the VBE/AI standard contain additional functionality and features above those that are provided in the basic sound services of VBE/AI, and most third-party sound libraries include a transitional set of device drivers.

These device drivers provide for compatibility with the majority of the existing installed base of audio hardware, while the VBE/AI component provides the support needed for future audio hardware. A VESA VBE/AI SDK targeted toward application developers is available electronically on CompuServe (GO GAMERS) and technical questions can be fielded on America On-Line (keyword: VESA). ■

Jon Burgstrom is the ISV developer support engineer at Media Vision Inc. The views and opinions expressed are not necessarily those of Media Vision Inc. Some are definitely not. He can be reached via e-mail at 71726.2335@compuserve.com or through Game Developer *magazine.*

VESA VIS

Human Machine Interfaces Inc.
757 W. N St.
Springfield, Ore. 97477
Tel.: (503) 747-2314
Fax: (503) 747-2627

The Audio Solution
P.O. Box 11688
Clayton, Mo. 63105
Tel.: (314) 567-0267

Miles Design Inc.
10926 Jollyville #308
Austin, Texas 78759
Tel.: (512) 345-2642

Net-Play: A High-Energy Network Solution

by André LaMothe

One of the most challenging aspects of multiplayer game programming is keeping the action on different machines in sync. You can master this challenge with some crafty programming techniques.

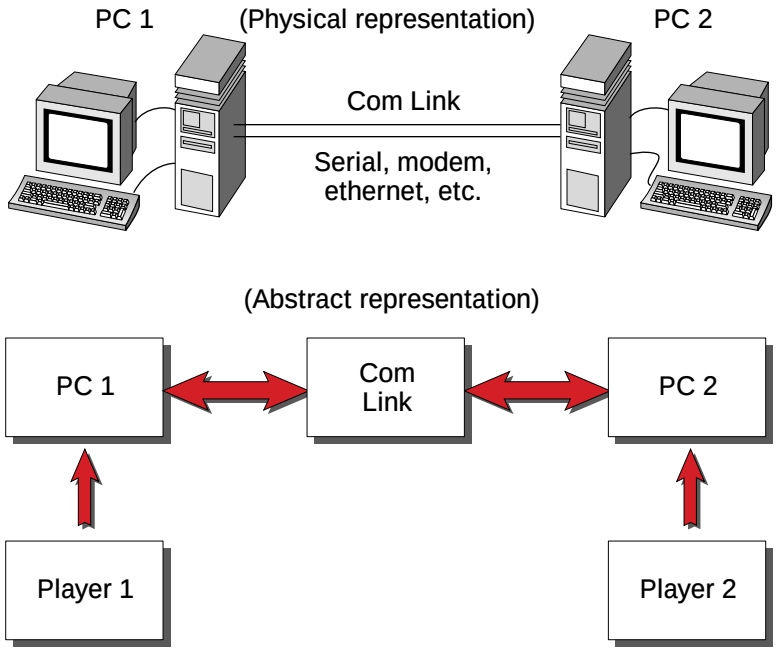
A video game is more than a collection of images moving around the screen. It is a conduit into another dimension—a dimension synthesized by the imagination and brought to life through the computer. These electrical worlds are places where the laws of physics and reality are ours to toy with. We have the power of gods, to do as we wish. However, it would be most excellent if we could place more than one player into our virtual worlds. They could be friends or foes, the choice would be theirs.

This would add to the texture of the game and allow another level of interactivity to emerge. Even with the most advanced artificial and "synthetic" intelligence systems, we can't yet mimic a human or the depth of game play that a human can emote. Two-player games have been around for a long time; however, they are mostly played on a single computer. Using a single computer to cater to two players has its shortcomings. First, each player has the same view. Second, both players have to be physically near each other because they are sharing the same hardware. Advances in networks and modems have made it possible to run multiplayer games with multiple computers. Phone lines and computer networks now have the bandwidths necessary to send the large amounts of information needed to run a multiplayer game in real time.

This ability to "link-up" multiple players and coexist in the same game is truly incredible. It brings game play one step closer to reality. The whole idea of a video game is to experience something that is either difficult, dangerous, or

A Multiplayer Modem Game

Figure 1. The Multiplayer Model



impossible to experience in our own universe. Having two or more players in a world together, each with his or her own agenda (as in real life) can make a game an order of magnitude more fun and rewarding.

We're done talking about the philosophical aspects of multiplayer games, so let's get down to business and talk about how these ideas are implemented and what a game developer needs to take into consideration to create a multiplayer game. We won't get into the actual implementation of a multiplayer game or discuss any code. It is best to first become familiar with the language and design techniques of multiplayer games before you jump into coding.

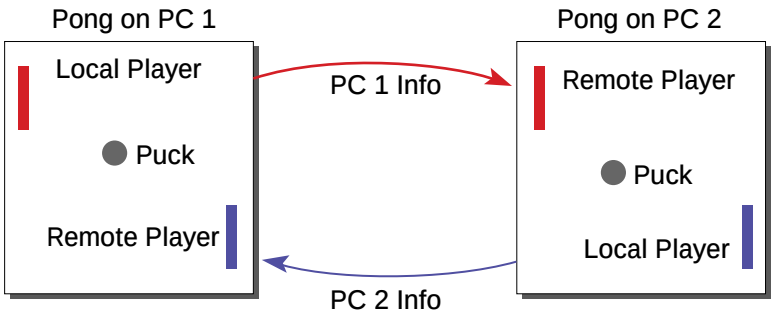
The Multiplayer Game

Creating a multiplayer game is not like creating a single-player game or even a multiplayer game on a single machine. Some constraints are imposed by the fact that more than one computer will be in the game. When designing a multiplayer game, you must consider many issues, such as communications, subsystems, and game architecture and adhere to many laws of programming. This methodology must be in flux from the beginning of the game development all the way to the end; otherwise, havoc will dance all over your code! Rule one to making multiplayer games is: design the game as a multiplayer game from the beginning. Don't try to hack it in later.

For this article, we will use a model based on two computers and two players (one on each computer), as shown in Figure 1. This model can be extended to multiple players using the same fundamental techniques to link-up two players. Moreover, we aren't going to concentrate too much on the type of communication channel we are using. It could be Ethernet, modem, or homing pigeons. The communications layer is irrelevant, at least at this point in our evolution.

Realize that there are about 1,000,000,000,000 ways to do anything on a computer. Even though many techniques and algorithms are mappable to another (an isomorphism exists that's one to one and onto), many are not. I say this because the techniques and methodologies we will discuss are only a couple of ways of looking at the problem. They are not the only viewpoints and solutions. The problems we will cover aren't an exhaustive list, either. We just want to get the basics down, then you can learn and extend them from there.

Figure 2. Information Transfer in a Simple Pong Game



Video Game Communication Strategies

To link up two players, we need to somehow keep both computers in synchronization. We need to somehow turn both computers into one computer so that each player is unaware of the physical discontinuity between the machines. We need to keep both game worlds in synchronization by using the communications channel to send information about each game to the other machine. As an example, let's talk about a Pong game. To keep a two-player Pong game in synchronization we would somehow need to make certain

pieces of information common to both machines. This would include the positions of each player's stick, the position of the puck, and finally the scores, illustrated in Figure 2.

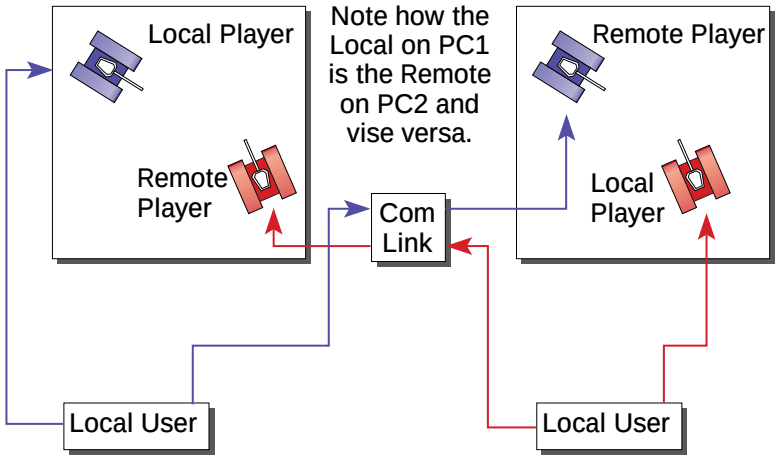
All right, that doesn't seem too bad. But what if we wanted to have a game that had thousands of objects and actions going on, such as a space battle with zillions of ships and asteroids. Then, we would quickly get to a point of complexity that wasn't so easily fathomable. Moreover, all the complex interactions of the game system would have to be synchronized. As you can see, we need a plan, or at least some ideas of how we are going to relay the state of each game to the other computer and vise versa. Whatever we do, it must be done on a cycle-by-cycle basis and must never let the two games get out of synchronization. Two methods can accomplish this goal. I call them I/O space synchronization and vector-positional space synchronization.

I/O Space Synchronization

As its name implies, I/O space synchronization uses the I/O state stream of the other computer to accomplish synchronization. Imagine, if you will, two computers, each running almost the exact same game program. I'll get to the "almost" part in a second. Each game is a tank game. You have two little tanks, one red and one blue. Each player controls one tank. Each tank can move or fire a missile, and that's it. Now, let's come up with some naming conventions. We will call the local player the game object that is controlled by the local computer. We will call the remote player the game object that is supposed to be controlled by the other remote computer, as shown in Figure 3.

For a moment, let's forget about the other player and concentrate on Player 1 and Computer 1. Also, let's assume that the software is designed in such a way that the two game objects in the game (one represents the local player, and the other represents the remote player) are controlled by the output of two functions. These functions are `Get_Input_Local()` and `Get_Input_Remote()`, respectively, and

Figure 3. A Setup for a Tank Game



are shown in Listings 1 and 2. As you can see, we have the keyboard tied to the local player; however, the remote player has nothing tied to it (yet), and so it does nothing. If we run the game, the player will be able to move around the local tank, but the remote tank will just sit there. Now here's the catch. What if we were to feed the `Get_Input_Remote()` function with the keyboard of Computer 2? Or in other words, use Computer 2's keyboard as another input device.

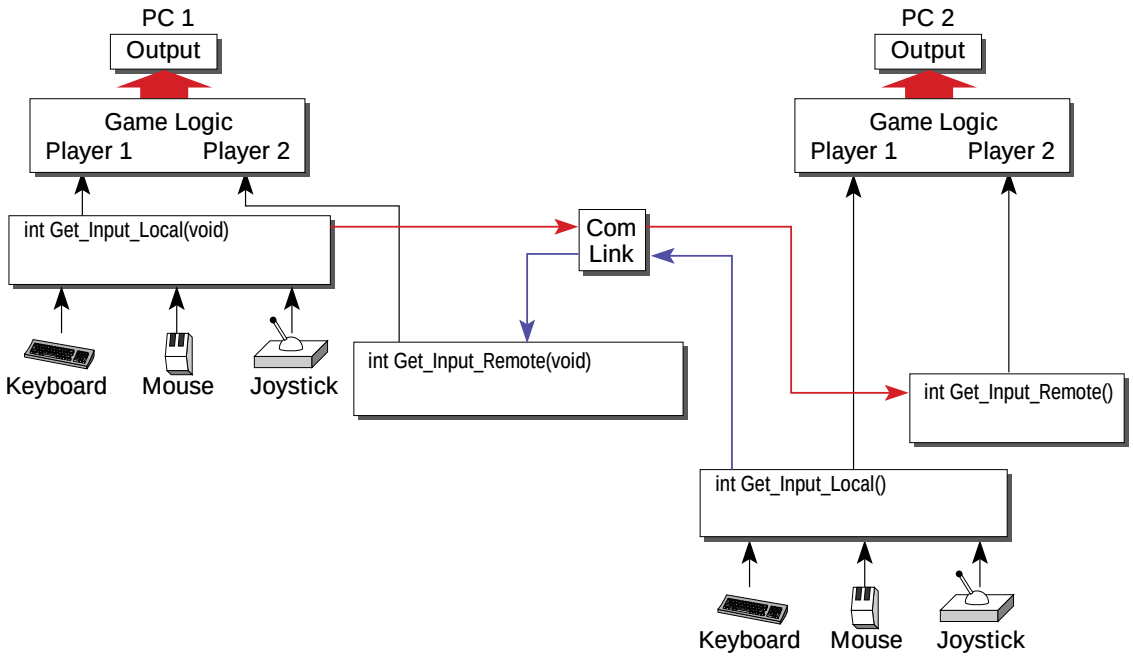
We would, in essence, be taking the input of another computer as the control for Player 2. This is the basic idea behind I/O space synchronization. We send the I/O state of each machine (their keyboards in this case) to the other machine. Therefore, each machine thinks it is running a two-player game on a single computer, but with one input coming from the keyboard and the other input coming from another "virtual" keyboard that is, in fact, the other computer!

This kind of synchronization will work fine as long as the exact same game runs on each machine, save one little caveat. The local player on one machine is the remote player on the other. If you run the exact same piece of software on two machines, the following problem will occur: the local player and remote player will always be in the same starting position. That may seem confusing, let me clarify.

When the game runs on Computer 1, it places the red player (local) at (10,10) and the blue player (remote) at (100,100). Then, the game running on Computer 2 does the exact same thing; however, from Player 1's point of view the remote blue player is at (100,100). But the remote blue player on the other machine should be the local player from that machine's point of view, and, therefore, be at (100,100) and blue! Alas, this won't happen because the exact same software is running! To remedy this, each game must be slightly different because the remote player on one machine is the local player on the other. Each game program must have been compiled slightly differently and have a file that is read or something to ensure that this is taken into consideration, as shown in Figure 5.

Now that we have discussed I/O space synchronization, let's briefly go over it one more time. Each computer runs a two-player game. Each computer obtains the input control for one player from a physical input device such as a keyboard or mouse. The other input that controls the other player is obtained from a virtual input that is retrieved via the communications port. So, all that each local player must do is send the state of its input device over the communications channel with each game cycle, and the other computer will interpret this as the remote player's

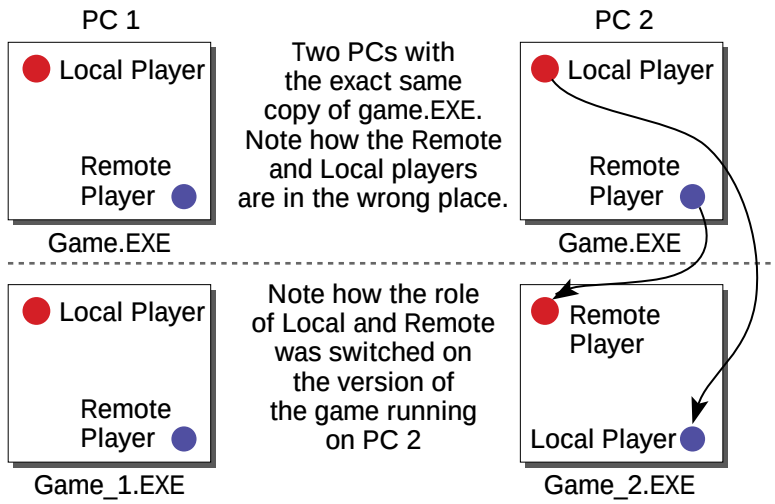
Figure 4. The Software Architecture of an I/O Space Linked Game



input. Because each game is exactly the same (except for the start-up positions), each game should stay in synchronization. For example, if the local player on Computer 1 rotates a tank to the right, this input will be sent to the other computer as the remote input, and the tank on that computer will be rotated also. Don't worry if all this local and remote stuff confuses you. It's like time travel; it's kind of weird!

I/O space synchronization will usually work fine; however, it can go amiss. Obviously, a problem can occur if information is lost during transmission, but another more subtle problem can occur if one machine is much faster than the other. Just food for thought, we'll cover problems like these later. I didn't want you to think that this was a perfect world...even for a minute!

Figure 5. The Proper Setup of Two Machines



Vector-Positional Space Synchronization

The second tactic we can use to link up two computers is through vector-positional space synchronization. Instead of sending the state of the input devices as we do in I/O space synchronization, we send the state of the entire universe, or at least enough of a subset so that the games have a current copy of all the positions and states of the objects in the game universe. This method is similar to keeping a globally shared game space in two computers, like shared memory. This game space and all the objects in it are continually updated via the communications link. For example, let's use a three-dimensional, asteroids-type game. Each game has a collection of asteroids; however, there must be a copy of the remote asteroids on the local machine and vise versa. So, whenever an asteroid is created or destroyed, its new state is sent to the remote machine. Again, this is done on a cycle-by-cycle basis. Vector-positional space synchronization is

Listing 1. Get_Input_Local

```
Int Get_Input_Local(void)
{
// query the input device and return status to caller

switch(input_device)
   {

   case MOUSE:
      {
      // get mouse direction and status of buttons
      if (mouse has changed)
      {
      return packed information;
      } //end if mouse moved
      else
      return NO_MOTION;
      } break;

   case JOYSTICK:
      {
      // get joystick direction and status of buttons
      if (joystick has changed)
      {
      returned packed information
      } // end if joystick moved
      else
      return NO_MOTION;
      } break;

   case KEYBOARD:
      {
      // get keyboard state
      if (a key is down)
      {
      return packed information;
      } // end if a key idd pressed
      else
      return NO_MOTION;
      } break;

   default:break;

   } // end switch
   } // end Get_Input_Local
```

Listing 2. Get_Input_Remote

```
int Get_Input_Remote(void)
{
// look at the data port and see if
there is a packet there

if (data ready at COMM port)
{
// grab data from queue and pack it

return packed information

} // end if data ready

} // end Get_Input_Remote
```

more powerful than I/O state synchronization. We know that both universes are always in perfect unison because we always send the state of almost everything.

This technique has one drawback: the amount of information we must continually blast over the communications channel. You will be making games that use a modem to communicate because this is the device that our commercial audience owns. The modem has an average bitwidth of 2400 baud. Therefore, sending tons of state information can be a problem.

The best method of synchronizing two games is to use a mixture of each technique. For instance, I/O space synchronization doesn't allow new objects to be created in the universe because the other machine wouldn't know about it. However, if we sent this new state information to the remote machine when a creation event occurred, the game play could continue without losing synchronization. I suggest you figure out what you want to happen in each game and create a communication strategy that takes the best of both worlds. However, I tend to rely mostly on I/O space synchronization for my games.

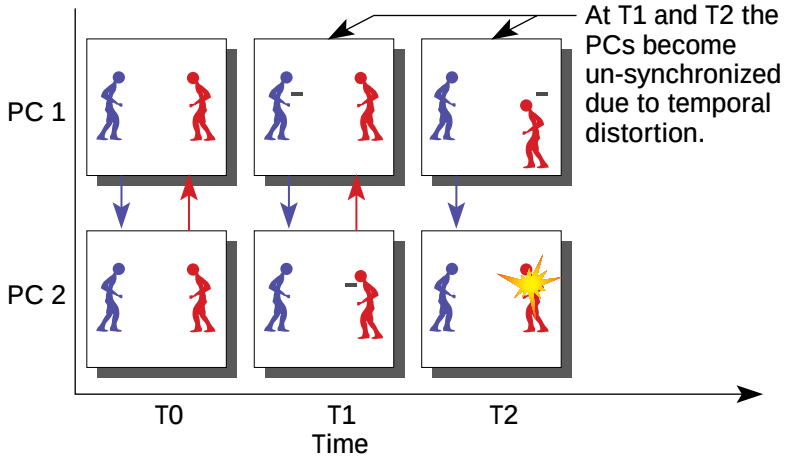
We have a couple of tricks under our belts, but it's time to let the harsh walls of reality come crashing down and cover some of the problems that will occur and how to remedy them.

Types of Distortion

Two types of distortion (or missynchronization) can occur in a linked game: temporal aliasing and nondeterministic aliasing. Temporal aliasing occurs when one computer is faster than the other and is further in the game than the other computer. As an example, say we have two players pointing guns at each other, as shown in Figure 6. Player 1 fires and destroys Player 2 in Player 2's world; however, on the other machine, Player 2 moves fast enough to avoid the missile and lives.

Now, both machines are out of sync. To rectify this problem, we must ensure that both games run at the same speed regardless of the speed of the machines. The two machines must synchronize to the lowest common denominator.

Figure 6. The Effects of Time Lag



Two methods will help us accomplish this: the first method locks the game to a specific frame rate, such as 15 or 20. Each machine is guaranteed to operate at this frame rate and no faster. The game logic will then operate at the same speed. The second method sends a token back and forth between each machine. Only when the computer sends the token and gets it back can it proceed to the next frame or cycle. This will synchronize the game to the slower machine.

The other type of distortion or nondeterministic aliasing occurs when something happens in one game that isn't reflected in the other. This is more of a problem when using I/O space synchronization. Imagine if Players 1 and 2 were both in a room with each other. All of a sudden a monster pops out and dodges right; however, on the other machine it dodges left! This would occur if a random number generator were used to drive the monster's logic. The moral to the story is, if you use I/O space synchronization, everything in the game must be completely deterministic. At least as far as each instance of the game is concerned on each computer. As a developer, you must use patterns or look-up tables with precomputed random numbers. However, free-running random number generators cannot be used on each computer.

Conclusion and Reflections

That wasn't so bad. Creating linked-up games is not magic, but it is a formidable task. You must take many factors into consideration and make a lot of design decisions from the beginning of your game design. Otherwise, your code will look like a bundle of fiber optic cable! ■

André LaMothe, pictured here after a long day of multiplayer game programming, holds degrees in math, computer science, and electrical engineering. He has worked in neural networks, three-dimensional graphics, virtual reality, and robotics, but now he's doing serious research—he's writing computer games. His latest book, Tricks of the PC Game Programming Gurus, *will be published by SAMS Publishing. He has no e-mail address (he's worried the FBI will bust him for pirating video games at age 12), but you can contact him through* Game Developer.

The Doom of Doom

by Alexander Antoniades

Last month, we brought you the story of Doom. This month, we follow up with a look at how Doom is faring in the market—and its unusual band of followers who are creating their own descent into Hell.

In an industry where the life of a product can often be measured in months instead of years, the idea that a particular program would stay in the limelight long enough for people to spend months writing extensive utilities for it seems unlikely to say the least. But to one company, Id Software, it happened twice.

The first time was with its smash hit game Wolfenstein 3-D. After the release of Wolfenstein, level editors sprang up, and the flood gates burst open. Utilities were made for changing the bitmaps of the characters, the walls, and pretty much anything else as long as the basic level and episode structure (six episodes with 10 levels per episode) remained in tact. Hacked levels were turning up all over the place. When you saw somebody starting a game of Wolfenstein, you never knew what to expect.

Fame or Fortune

This unexpected fame represented a problem to Id and its distributor; at the time, it was Apogee. The game and the first episode were in the public domain, where money was made from people ordering the remaining five episodes from Apogee. If people could create their own levels, who would order the final game? Apogee went on the warpath threatening to sue the authors of the various customization utilities, and Id designers stated that nobody would be able to hack the maps in their next game. In the midst of this concern, sales of Wolfenstein kept going strong. The game was on its way to becoming the best-selling shareware game of all time.

By the time I visited the Id designers during the creation of Doom, I asked them if they were going to allow the kind of modification to Doom that had happened to Wolfenstein. They said not only were they going to allow it, they were going to encourage it. "If people get satisfaction out of modifying our code, it's just another form of entertainment that they derive from buying our games," said John Romero. "We fully expect people to be blowing away Barneys soon after the game is released," echoed John Carmack.

They had been surprised when the first slew of editors came out for Wolfenstein because they thought the compression algorithms that compressed the levels would be some of the hardest code to crack. So, after Doom was finalized, they planned to release some level specifications and the Binary Space Partitioner they used, so that wannabe Doom hackers wouldn't have to write their own.

But Carmack's vision of a Barney holocaust was realized soon after the shareware version of Doom v. 0.99 was released on Dec. 10, 1993. Within weeks, there were saved game editors, map viewers, and a slew of other hacks, including BarneyDoom, which turned the main villains into giant Barneys complete with sound effects.

Barney in Hell

There's something about Doom that inspires people like Bill Neisius, a professional aerospace programmer who works mainly on manufacturing and inventory control on Apollos and VAXs, to hack BarneyDoom in his spare time. You know its serious when two other people (David Lobser and Aaron Blackwell) rerender the Barney images in 3D Studio

A Journey to Hell with BarneyDoom

and touch them up with paint programs so that Barney looks even better.

Bill Kirby, another professional business programmer, was bored playing Wolfenstein, but he was still fascinated by the technology. He wrote two of the map editors for Wolfenstein, which earned him a nasty letter from Apogee. When Doom came out, although he doesn't usually modify games, he figured he'd check it out. He got the program by calling the folks at Id Software. They sent him some source code, and he made one of the first map utilities for Wolfenstein. His experience with Wolfenstein was helpful because he found similarities in the file formats between the two games.

Moving up the ladder of Doom is game hacker whose goal is to make every game he gets modifiable so his two-year old daughter can play and win. With his arsenal of disassemblers and knowledge of programming, he has made utilities for many commercial games. One of his utilities, RA Easy (a utility used to modify LucasArts game Rebel Assault), worked so well he got a call from LucasArts, asking if he'd gotten hold of a beta copy. His opinion was that Id designers had gone out of their way during the development process to make Doom easily modifiable because there were a lot of hooks, and the main data was easily accessible.

As it stands now, one of the key Doom utilities is Deu, a level editor, written by a Raphael Quinet, a Belgian electrical engineering student. This utility has been passed from author to author because the source code has been included with every release. As one person's interest wanes, another picks it up. Using this editor in conjunction with one of the binary space partitioners (preferably Id's own) makes the creation of levels very easy, leading to a proliferation of Doom levels over on-line networks and Internet.

Why Is This So Easy?

What makes the data easy to change is the way Doom is written. The main executable, DOOM.EXE, loads a data file called DOOM.WAD or DOOM1.WAD. This file contains all the level, image, and sound data for the game. By modifying the .WAD file, most of these independent utilities work. What Id added to Doom that wasn't in Wolfenstein is a disclaimer that would pop up on the screen if the .WAD file isn't the original file that came with the program.

The best resource of information about .WAD file structure is a text file called "Unofficial DOOM Specs" by Matt Fell, who, together with other Internet denizens, has compiled most of what's inside a .WAD file.

Another good source of information

WHERE DO I GET THIS STUFF?

The best place to start is with the official Doom FAQ file, which contains more than you ever wanted to know about Doom and includes the e-mail addresses of most Doom spelunkers.

The file is available on the following Internet news groups:
- **comp.sys.ibm.pc.games.action**
- **comp.sys.ibm.pc.games.announce**
- **comp.sys.ibm.pc.games.misc**

Additionally, the FAQ is available at the following Internet sites:
- **ftp.uwp.edu in either /pub/incoming/id or /pub/msdos/games/id/home-brew/doom**
- **ocf.unt.edu in either /pub/incoming or /pub/doom/text**
- **wuarchive.wustl.edu in /pub/msdos_uploads/games/doomstuff**

Alternately, the FAQ is available from the Software Creations BBS: (508) 365-2359 at 2400 baud, (508) 368-7036 at 9600 to 14.4k v. 32bis, or (508) 368-4137 at 14.4 to 16.8k HST/DS.

Outside these sources, the FAQ is available on most commercial on-line services.

If you're interested in contacting any of the Doom tool authors mentioned in this article, you can reach them at the following e-mail addresses:

Bill Neisius	**bill%solaria@hac2arpa.hac.com**
David Lobser	**lobser@csn.org**
Aaron Blackwell	**aaron525@denver.relay.ucm.org**
Bill Kirby	**bkirby@netcom.com**
Raphael Quinet	**quinet@montefiore.ulg.ac.be**
Matt Fell	**matt.burnett@acebbs.com**
Hank Leukart	**ap641@cleveland.freenet.edu**

about Doom is a text file called the "Official DOOM FAQ." It's written by Hank Leukart, a beta tester for Apogee, who noticed that there were questions about Doom that would end up getting asked over and over again, so he compiled a FAQ (frequently asked questions) file. He works with Id in compiling this document that's swelled to over 200K and lists over 80 after-market Doom utilities.

What about Id

If you're wondering where Id is in all of this, it's moving forward like nothing's happening. The Id team is working on various Doom ports, including Windows, Atari Jaguar, and Linux/X; preparing new levels for the commercial version of Doom called Hell on Earth; and starting up its next project called Quake.

The Doom after-market has made the product more marketable now than when it was first released. With all the new levels and graphics available, players who found the original game not enough can get their fill with all the add-on utilities. The only stipulation that Id has made so far is that created levels only work on the registered version. This is done by placing an item that only exists in the registered version on that level. Most people have abided by this rule, so there haven't been any problems.

It will be interesting if this move makes other developers think more about potential after-markets of computer games. As Id CEO Jay Wilbur put it: "Our fans used to be our salespeople, but now they're our developers as well." ■

Alexander Antoniades is associate editor of Game Developer *and assistant editor on* OS/2 Magazine.

AI to the Rescue

by Steve Estvanik

Programming an intricate game like bridge has thwarted enthusiasts for years. Using a sophisticated AI engine and some ingenuity, you can add complex rule- and logic-based features to your card games.

The three
Laws of Thermodynamics
have been summed up as:
You can't win.
You can't break even.
You can't get out of the game.

The first two
Laws of Artificial Intelligence
should be:
The hard stuff is easy.
The easy stuff is hard.

Case after case can be cited to show how tasks people perform without thinking, like recognizing a face or carrying a package while running, present almost impossible challenges for present AI techniques. Yet, human difficulties, such as remembering long strings of numbers or complex connections among events or objects, have simple computer solutions. Nowhere is this more obvious than in the game of bridge.

Bridge presents some formidable tasks for humans who wish to excel. You must learn to count cards and determine where each of the 52 cards is. You need to be aware of probabilities of distribution and card placement and how these change with each play.

When I started playing duplicate bridge, I marveled at the players who could recall hands they played last month or last year and describe not only what each person held but how the hand played. It took practice, but my partner and I, over the course of a year, became proficient enough that we could sit down after a session of 24 hands and reproduce every hand we played—recapitulating what card every player held and how the cards were played. That mental effort is trivial for a computer.

Bridge differs from other card games in that it consists of two distinct phases: bidding and play. During the bidding, you describe your hand to your partner in a coded language. Then, the hand is played out, with one hand exposed for all to see.

These two phases present vastly different mental and computer challenges. It's relatively easy to teach a beginner the essentials of a bidding system. It turns out to be one of the more difficult tasks to do the same for a computer. Before you can define a rule, you need to be sure the computer understands such basic concepts as suit, distribution, and point count. Part of the intelligence required for a bridge AI relies on a shared background. Most new bridge players are familiar with the now-standard 52 card deck, the four suits, and the names and values of the various cards. Many players are familiar with the concept of the trump from other games like pinochle or Rook.

False Starts and Blind Alleys

Bridge AI tends to be less theoretical than much of chess AI has been. It's messy and heuristic. But the problems go deeper. Chess is a simple zero sum game of complete information. Not only are the rules explicit and invariable, but, at all stages, both sides know exactly what the entire universe looks like.

This is not the case in bridge. The scoring is not precisely zero sum—my making a contract is not the same as an opponent going down in defense. Even

in the best bid hands, the opponents can find a sacrifice. And the value of that sacrifice depends on the randomly dealt hands that follow. The same hand dealt at different times can have drastically different effects on the final score.

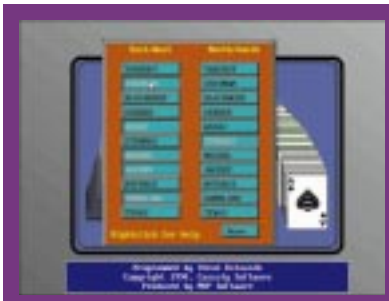
There are the other large elements of probability that must be accounted for. Each player bids while knowing only 25% of the cards. Through the use of a coded system, opponents and partners exchange information. But even if all hands were exposed, there would not be consensus on the best bidding and play. What works on one hand might not be suitable for all others. Players must constantly calculate not only this hand, but the style and current abilities of their partners and opponents and the current state of the game. What works with one partner or against some opponents might not be proper against or with other players.

Previous Attempts at Bridge AI

Early commercial bridge games were terrible. They made mistakes no beginner would even think of. Over the last several years, stronger bridge programs have emerged. Since this is a discussion of the AI techniques I used in my game, I'll limit my general comments on other approaches.

One of the first approaches to show strong results was the use of flow-charts. This requires all possible types of hands be considered. The engine first does an analysis to decides which chart is applicable, then the engine processes the selected flowchart.

The beauty of this system is that it's very easy to show the player how the program arrived at its conclusion. It's less useful as a teaching tool if the flowchart branching is not self explanatory. The down side is that alternate systems are more difficult to add. Conventions pose a particular difficulty, since so many conventions would require alternate flowcharts. As the number of conventions increases, and as each side can choose which of the many conventions they want



Using an AI engine allows you to program complex rules structures and bidding strategies into your game.

BRIDGE BASICS

Bridge is played by four players. All cards are dealt so each player has 13 cards that are kept hidden from other players. Players are designated as the four directions: north and south, east and west, with players sitting across from each other being partners. A hand of bridge is divided into two parts: bidding and play. During bidding, you try to describe your hand to your partner and make a claim to the number of tricks you can take. During play, you try to make the contract that you bid.

To decide how strong your hand is, we assign points to certain cards. These are called high card points (HCPs). We rate an ace (A) at 4 points, a king (K) at 3, a queen (Q) at 2, and a jack (J) at 1. To count, though, a king must be accompanied by at least one other card in the same suit. We show this as Kx and describe it as a "king doubleton." Similarly, to count for HCPs, the queen must have two other cards, and the jack needs three other cards of the same suit.

Bidding starts with the dealer and proceeds clockwise around the table. The suits are arranged in the following order: clubs (C), diamonds (D), hearts (H), and spades (S). Thus 1S can be bid over 1H, but over 1S, you must bid 2H. Bidding a suit usually means you wish to use that suit as trump. Bidding normally starts at the lowest level possible, but there are also times when the bidder jumps the bidding. (For example, an opening bid of 2H or 3H shows a very different hand from 1H.)

Playing in a suit contract makes that suit the trump suit. During play, you must follow the suit played. But, if you lack any cards in the suit played (this is called a void), you may play a trump and win the trick. You want to play in trump if you and your partner have at least eight of the cards in a suit. You may also play without trump (NOTRUMP or NT).

Bidding continues until everyone has had a chance to bid, and there are three passes in a row. The person who first bid the suit of the final bid becomes the declarer. The person to

(Continued on p. 44)

Table 1. The Jacoby Convention

After the transfer, subsequent bids by the responder describe the player's hand:

- Pass shows a bust.
- 2S shows 5-5 in majors, invitation to game.
- 3H after spade transfer shows 5-5 in majors, forcing to game.
- 2NT is invitation: with three of the transfer suit, opener bids game with big hand, three of suit with minimum. With only two of transfer suit, bid 3NT with maximum.
- Raise to three of transfer suit is invitation.
- 3NT offers choice of game contracts.
- Four of the transfer suit promise six-card suit.
- 4NT is quantitative. Partner can bid slam or five of the transfer suit.

to use, the set of possible flowcharts becomes an unwieldy matrix.

Another more recent approach allows you to teach the program bidding systems. You bid a series of deals, giving the bids for each hand, and the game learns the bids by observation. There are several problems with this neural network approach. First, if you make a mistake, due to faulty memory, boredom, or perhaps even your own mistaken ideas, the computer learns your system incorrectly. This problem is compounded if you never discover you made the error.

A larger problem is that you have to spend dozens if not hundreds of hours bidding deals so the game can learn your system. Even after that time, there's no simple way to know if it has really interpreted the deals or situations correctly. Beyond these problems, it's easy to create hands that experts would disagree about.

Mere copying of what was done on a particular hand doesn't guarantee there's any learning of the principles behind why a particular bid was made. Sometimes it may be a simple rule: "Bid your longest and strongest." Other times, a response may be based on everything that's happened thus far: "I opened, and the opponents bid a little. My partner didn't bid, but must have some points or the opponents bid should be much higher. Therefore, I can make a stronger bid than my cards justify by factoring in partner's presumed strength."

Many rules are analogous to irregular verbs in a natural language translator. You can't deduce a particular rule, you just have to learn it. A regression analysis or neural network approach would still need to have all the special cases designed.

Beyond these concerns, there's a more fundamental problem. Much of bridge is codified, not natural, so there's no way to figure out even the simplest conventions without playing many many hands. Just to teach it Jacoby, you'd have to give it dozens of hands, as shown in Table 1. Then, there's the additional problem that there's no consensus on many tougher hands. *Bridge World*'s monthly Master Solver hands often have at most 30% to 40% of the experts in agreement.

The Solution

When I first started this project, I hoped to let users configure the system by defining their own conventions and even telling it certain preferred ways of bidding. Instead, MVP Bridge lets you choose from several basic bidding systems, and you have options to use any of

BRIDGE BASICS

(Continued from p. 43)

the left of declarer is the leader for the first trick. The declarer's partner is dummy—those cards are turned over for everyone to see, after the opening lead.

Your goal in rubber bridge is to win two games to make a rubber. A game is scored 100 points made by bidding. This can be the result of one or more hands. Scores vary by suit. Clubs and diamonds are the minor suits and worth only 20 points each. Hearts and spades are the major suits and worth 30 points each. Thus, a contract of 4H would give you 120 points, which is a game, but a contract of 4D gives you only 80 points. No trump is a special case. The first NT level is worth 40 points, and successive levels are worth 30 each. Thus, 2NT is worth 70 points, and 3NT is 100, a game. If your contract does not give you a game, it remains as a partial score until one side makes a game. If a contract is set, the defenders gain points.

A convention is a bid that carries additional information and does not promise anything in the suit actually bid. For example, in the Stayman convention, a 2C bid over 1NT says nothing about the responder's club suit, but does promise at least four cards in at least one of the major suits. Conventions are used to describe your hand more accurately. Any conventions in use must be known by all players—you may not have secret arrangements with your partner.

The person who first bid the suit that becomes the final bid is the next declarer. To make a contract, you must take six tricks plus the number you bid. Thus, a 3H contract requires you to take nine of the 13 tricks. The person to the declarer's left is the leader.

The leader selects a card and places it so everyone can see it. At this point, the partner of the declarer lays down his or her cards for everyone to see. The partner is now the dummy and his or her cards are played by the declarer. In MVP Bridge, when north is the declarer, you get to play as declarer, and your original hand becomes the dummy.

Play proceeds in a clockwise fashion. You must follow suit. If you are void (have none of that suit), you can play any card. In a trump contract, the highest trump played on the trick wins. The winner of the trick leads the following trick.

the dozen conventions available. You cannot add conventions (yet) or modify existing rules.

The system that evolved for bidding is a combination of rules (17 aspects defined, but on average only to six used in any particular rule) and hard-coded heuristics coded. Aspects include factors such as high card points; total points estimated between partners; distribution; last actions by opponents, me, and partner; vulnerability; level of bidding; trump support; and quantity and quality of defensive stoppers in each suit.

If you were teaching someone elementary bridge, a basic description of a rule for opening at the second level might look like this:

"With a six-card suit and six to 12 high card points (but not with a six-card club suit), when no one has bid yet, give strong consideration to an opening bid at the second level in your longest suit if you're using the Weak 2 style of play."

Some of the difficulties are immediately clear. Each rule may depend on a variety of conditions. Some are constant (you'll always have six cards during this hand; you either are or are not playing the Weak 2 system); others are more variable (someone may or may not have bid; your partner's bid may change the value of your cards). The difficulty comes in defining the priorities and resolving clashes among several rules. In a computer system, the AI must be flexible enough to deal with any of the billions of hands possible, yet strong enough to be able to make a reasonable bid.

In practice, while it might be simple to add each rule to the list, a fair amount of testing is required to see that the rule actually works as planned and, more importantly, that by adding this rule, other rules are not adversely affected. The resulting testing and debugging of new rules is iterative and recursive and not the sort of system that is easily given to users. It's too easy to enter a wrong rule, and, without a thorough understanding of the rule and heuristics systems, debugging would be difficult.

When I turned to conventions, I first used the same rule-based generalized system I had used for standard bids, but

Listing 1. Formal Descriptors for a Hand (Continued on p. 46)

```
#define HCP   1           /* ASPECT 1 -- High Card Points (HCP) */
   #define G0x7     1
   #define G0x10    2
   #define G4x8     3
   #define G6x10    4
   #define G6x12    5
   #define G8x10    6
   #define G8x12    7
   #define G10x12   8
   #define G13x15   9
   #define G15x17   10
   #define G16x18   11
   #define G20x21   12
   #define G21x22   13
   #define G24x25   14
   #define G25x26   15
   #define G8P      16    /* >= 8   */
   #define G10P     17    /* >= 10  */
   #define G13P     18
   #define G16P     19
   #define G18P     20
   #define G21P     21

#define TCP   2           /* ASPECT 2 -- Total Card Points (TCP) values are      */
                          /* same as for HCP                                     */

#define DIST  3           /* ASPECT 3 -- Distribution          */
   #define FLAT     1     /* 4-4-3-2 or 4-3-3-3                */
   #define MAJOR4   2     /* 4 card major                      */
   #define MINOR4   3     /* 4 card minor                      */
   #define MAJOR5   4     /* 5 card major                      */
   #define MINOR5   5     /* 5 card minor, balanced (at least  */
                          /* two cards, each suit)             */
   #define UNBAL    6     /* for example, 4-4-4-1, 4-4-5-0     */
   #define SUIT5    7
   #define SUIT6    8
   #define SUIT7    9
   #define SUIT8    10    /* >= 8 card suit                    */

#define OPP         4     /* ASPECT 4 -- Opponent Status       */
   #define NB       1     /* no bid yet                        */
   #define BOP      2     /* both opponent passed              */
   #define LHOP     3     /* LHO passed                        */
   #define RHOP     4     /* RHO passed                        */
   #define OO       5     /* opponent opened                   */
   #define LHOO     6     /* LHO opened                        */
   #define RHOO     7     /* RHO opened                        */
   #define OOV      8
   #define LHOV     9     /* LHO overcalled                    */
   #define RHOV     10    /* RHO overcalled                    */
   #define ONT      11    /* opponent opened NT                */
   #define LHONT    12    /* LHO opened NT                     */
   #define RHONT    13    /* RHO opened NT                     */
   #define OS       14    /* opponent showed strong hand       */
   #define LHOS     15
   #define RHOS     16
   #define OD       17
   #define LHOD     18    /* LHO doubled                       */
   #define RHOD     19    /* RHO doubled                       */
   #define OPR      20    /* opponent preempted                */
   #define LHOPR    21    /* LHO preempted                     */
   #define RHOPR    22    /* RHO preempted                     */
```

Listing 1. Formal Descriptors for a Hand (Continued on p. 47)

```
    #define ORV      3     /* opponent reversed              */
    #define LHORV    24    /* LHO reversed                   */
    #define RHORV    25    /* RHO reversed                   */
    #define ORD      26    /* opponent redoubled             */
    #define OBG      27    /* opponent bid game              */
    #define ONS      28    /* opponent showed new suit       */
    #define OST      29    /* opponent supported trump       */
    #define LHOB     30    /* lhop bust                      */
    #define RHOB     31    /* rhop bust                      */
    #define LASTO    31

#define PARTNER      5     /* ASPECT 5 -- Partner Status  statPart[]   */
    #define NB       1     /* no bid yet                               */
    #define PP       2     /* partner passed                           */
    #define PO       3     /* partner opened                           */
    #define POV      4     /* partner overcalled                       */
    #define PN       5     /* partner showed new suit over opening     */
    #define PS       6     /* partner showed strong hand               */
    #define PRV      7     /* partner reversed                         */
    #define PNT      8     /* partner opened NT                        */
    #define PPR      9     /* partner preempted                        */
    #define PBG      10    /* partner bid game                         */
    #define PST      11    /* partner supported my trump               */
    #define PD       12    /* partner doubled                          */
    #define PRD      13    /* partner redoubled                        */
    #define PB       14    /* partner bust                             */
    #define LASTP    14

#define ME           6     /* ASPECT 6 -- my status       */
    ... similar to the above

#define VUL          7     /* ASPECT 7 -- vulnerability   */
    #define WE       1
    #define THEY     2
    #define ALL      3
    #define NOTUS    4     /* that is, either neither or just them     */

#define EXCEPT       8     /* ASPECT 7 -- exceptions                   */
    #define SCLUBS   1     /* clubs -- longest suit is clubs           */
    #define SVCLUBS  2     /* longest suit is clubs and no void        */
    #define SMINORS  3     /* minors -- longest suit is a minor        */
    #define SMAJORS  4     /* majors -- longest suit is a major        */
    #define NCLUBS   5     /* longest is not clubs                     */
    #define NVCLUBS  6     /* longest is not clubs, and no void        */
    #define NMINORS  7     /* not minors longest is not minor          */
    #define NMAJORS  8     /* not majors longest is not major          */
    #define NOVOID   9     /* does not have a void                     */
    #define NSINGLE  10    /* does not have a singleton                */
    #define NSECMAJ  11    /* second suit is not a major               */
    #define T1MAJOR  12    /* agreed trump is major                    */
    #define T2MAJOR  13    /* agreed 2nd trump is major                */
    #define NSUIT5   14    /* no suit of 5 or more                     */
    #define NJACOBY  15    /* not playing JACOBY                       */

    ....

#define TRUMPS          13    /* ASPECT 13 -- trump support        */
    #define POOR        1
    #define FAIR        2     /* 2+ trump, or < 4 HCP              */
    #define GOOD        3     /* 3+trump, or 4-5 HCP               */
    #define EXCEL       4     /* 4+ trump, or 6HCP in trump        */
    #define GOODOPP     5     /* 4+ of opp´s trump                 */
```

realized that heuristics or, in some cases, specific code would be better, since conventions are so constrained. I use rules to start all conventions, but then switch to coded modules for each convention, since the possibilities can be rigidly defined. Obviously, this is just one solution, and other bridge programs will surely arrive that use vastly different techniques.

MVP Bridge

The result, MVP Bridge, was released earlier this year. I started with a single approach for bidding and play: the use of a rule-based bidding engine. The bidding engine required some tweaking, and adding conventions eventually required more algorithmic approaches. It also became clear that play was more heuristic than rule-based, especially since it is so much more constrained. This eventually resulted in two completely distinct approaches to the AI.

This article focuses on the bidding algorithms. I originally planned to let people add their own conventions and even bridge bidding style, but they would have to learn how to teach new rules to the engine. At present for MVP Bridge, this consists of working with the debug logs, which can run to several hundred thousand K per hand, detailing the decisions made, and rules selected.

Each rule has up to 17 conditions that must be met to be triggered. When it came to conventions, I first started to use this same system, but it soon became clear that I would just be coding some very explicit situations, so I switched to a more conventional coding approach. The result is a hybrid system that can be adapted to very specialized bidding, while still retaining a lot of flexibility to handle hands never considered before.

Design of the AI

Unlike more abstract topics, such as natural language or vision, in bridge and chess, the identification of relevant features is straightforward. Possible solutions are reasonably clearcut. Thus, it's potentially possible to create a program that outplays humans, even the creator. This is true even if the program does not play perfect bridge at all times. I used a fairly

standard approach to the design of the AI:

1. Characterize the task, analyze task and actions of the program, and construct the system domain as a collection of objects, properties, and operations.
2. Design a formal representation so bridge knowledge is stated explicitly.
3. Embody the representation in the computer system. It's important that the operations of the computer are consistent with the rules of the formal system.
4. Create a search procedure and operate on the computerized data structures to carry out the task.

Let's look at each of these in detail for the MVP Bridge AI.

Bridge Bidding AI

The nature of the game immediately suggests that the AI should be divided into two separate sections. The play of the hand is to some extent dependent on the bidding, but, by the time the play starts, the bidding information has become history and is easily worked into the AI as starting conditions. It makes sense to consider the bidding system first.

1. Characterize the Task. Since we're working with a constrained universe in which explicit laws and rules apply, much of the task is already done. The goal is to find ways to analyze each hand that's dealt and propose legal and appropriate bids.

2. Design a Formal Representation. Now the work begins. Obviously, some representation for the cards and hands is required. I chose to structure each player's hand as an array, turning cards on or off within that array:

```
Card[NHANDS][NSUITS][NCARDS]
```

where:

`NHANDS` equals 4 (the number of players)
`NSUITS` equals 4 (the number of suits)
`NCARDS` equals 13 (the number of cards in each suit)

This representation has the additional advantage of being flexible enough to work for alternate games in which there are different numbers of players, suits, or cards. In bridge, this is not a consideration, but in a game like hearts it would be, since the game is often played by three, four, or five people.

In addition to cards, facts about each hand need to be codified. Listing 1 shows some of the aspects identified. Eventually, I came up with 17 aspects that described the hand. Some were obvious, like high card points or distribution. Others evolved over time, such as exceptions or level of bidding.

Defining Aspects

High card points (HCPs) describe the value placed on a particular hand. The first series of values such as `G6x10` mean a range of 6 to 10 inclusive, while `G13P` means 13 or more points. These could

Listing 1. Formal Descriptors for a Hand (Continued from p. 46)

```
    #define EXCELOPP     6      /* 5+ trump, or 4 with more than Q    */

#define PRIORITY         14     /* ASPECT 14  an actual value         */

#define CONVENTION       15     /* ASPECT 15                          */
    enum { TAKEOUT=1, STAYMAN, BLACKWOOD, GERBER, WEAK2, STRONG2, NEGDBL, JACOBY,
           NTFORCE, GAMBLING, TEXAS, UNUSUAL, WKJUMP, MICHAELS, MINORTFR};
```

Listing 2. Decoding Rules

```
ABCD

1xCx  C=1, PASS,  2=DOUBLE, 3=REDOUBLE

2xxx   new suit
2Bxx   B = 0 relative level for D
       1 = absolute level (eg,
           preempts)
2xCx   C = 1 longest suit
   2 No trump
   3 4 card major
   4 second suit
   5 transfer (bid 1 lower than actual
               suit)

4xxx   raise of suit previously bid
4Bxx   B = 0 relative level for D
       1 = absolute level (eg,
           preempts)
4xCx   C = 1 partner´s first
         = 2 NT
         = 3 my first
         = 4 partner´s second suit
         = 5 my second
4xxD   D = # of levels (next eligible
           bid)
```

AI Bridge Player Updates

Listing 3. Examples of Bidding Rules

```
A. REGULAR BIDS

/* -------- OPENING BIDS */
/* standard opening */
0,G13x15, SUIT5, NB, NB,NB,    0,0, 0,   1,0,0,0, 4,0,0,0,    2011,  "OPEN-NB-1"
0,G13P,   SUIT6, NB, NB,NB,    0,SMAJORS,0,1,0,0,0, 6,0,0,0,  2011,  "OPEN-MAJ6"
0,G13P,   MAJOR5, NB, NB,NB,   0,0,0,    1,0,0,0, 4,0,0,0,    2011,  "OPEN-NB-MAJ5"
0,G13P,   0, NB, NB,NB,        0,0, 0,   1,0,0,0, 2,0,0,0,    2011,  "OPEN-NB-MAJ4"

/* -------- RESPONSES*/
/* raise of partner´s suit */
G6x10,0,0,    0,PO,NB, 0,T1MAJOR,0,1,0,0, GOOD, 4,0,0,0,   4011,  "RA-MAJ-1LVL",
G10x12,0,0,   0,PO,NB, 0,T1MAJOR,0,1,0,0, GOOD, 3,0,0,0,   4012,  "RA-MAJ-2LVL",
G6x10,0,0,    0,PO,NB, 0,0,0,1, 0, 0,     GOOD, 3,0,0,0,   4011,  "RA-1LVL",
G10x12,0,0,   0,PO,NB, 0,0,0,1, 0, 0,     GOOD, 2,0,0,0,   4012,  "RA-2LVL",
0,0,0,        0,PO,0,  0,0,0,2, 0, 0,     EXCEL,3,0,0,0,   4011,  "RA-1LVL-WK1",
0,0,0,        OOV,PO,0,0,0,0,2, 0, 0,     EXCEL,3,0,0,0,   4011,  "RA-1LVL-WK2",
/* rebids */
0,0,0,    0,PO, IST, 0,0,0,0,LT25,0, 0,  4,0,0,0,         4113,  "IFIT-GAMETRY",
0,0,0,    0,PO, IST, 0,0,0,0,GE25,0,     FAIR, 4,0,UNBID,0,NT3,  "IFIT-NT",

B: CONVENTIONS

Conventions are handled differently from other bids.  Only the initial conventional
bid is generated from these rules.  Once the convention is triggered, the responses
and rebids are handled by special convention routines.

Lowest priority for a convention is 7.
HCP TCP DIST | OP PT ME |VL EXC QT LVL TP LAST | TR PR CONV Stp 2nd VObj | name

/* ============= STAYMAN */
/* start Stayman sequence? */
G8P,0,MAJOR4, NB,PNT,0, 0,0,0,2,0,LPART,     0,8,STAYMAN,0,0, C2,    "STAY-2C-1",
0,G8P,MAJOR4, NB,PNT,0, 0,0,0,2,0,LPART,     0,8,STAYMAN,0,0, C2,    "STAY-2C-2",
G4x8,0,MAJOR4,NB,PNT,0, 0,0,0,2,0,LPART,     0,8,STAYMAN,0,0, C2,    "STAY-2C-3",
G8P,0,MAJOR4, NB,PNT,0, 0,0,0,3,0,LPART,     0,7,STAYMAN,0,0, C3,    "STAY-3C-2",
G8x10, 0,FLAT, 0,PNT,0, 0,0,0,2,0, 0,        0,6,STAYMAN,0,0, NT2,   "STAY-NT-RSP",

/* ============= JACOBY    5    */
0,0,SUIT6,  NB,PNT,0, 0,NMINORS,0,2,0,LPART, 0,9,JACOBY,0,0, 2051, "JACOBY-MAJOR6",
```

have been actual numbers, but that would have enormously increased the number of possible rules to be considered.

If a rule was valid for either 6, 7, 8, 9, or 10 points, five separate rules would need to be written. For the limits such as "greater than 10," there would be dozens of possibilities. Instead, I created relevant ranges. The actual values are a function of the bidding system in use and would vary slightly with other systems.

For example, either 15 to 17 or 16 to 18 can be used in rules for deciding when to open "1 no trump," depending on the system in use. A hand can easily have more than one value for a particular aspect. This is a vital element of the design and a reason for its robustness. Thus, a hand that had precisely 17 HCPs is rated as G15x17; G16x18; and G8P, G10P, G13P, and G16P.

In addition to HCPs, another factor, total card points (TCPs) includes an assessment of how the distribution of the hands adds to its value. The values for this aspect are identical to those for the HCPs.

Next to consider is distribution, which is straightforward. Hands are described as flat, unbalanced, long suited, or having long majors or minors. Again, hands can have multiple values; a long suited hand, by definition, is unbalanced.

The next series of aspects describes the status of the various players. It's often important to know who opened the bidding (me? my partner? which opponent?). Sometimes, you need to know which opponent did something; other times, just the fact that something happened.

Aspect 4 covers all possibilities for the opponents. Aspects 5 and 6 cover similar ratings for my partner and ME, that is, the computer player who's doing the evaluation. Aspect 7 is a simple one. Vulnerability is dependent on previous hands. Some bids are only made if the vulnerability is favorable.

Exceptions need to be coded, also. Some bids are made whenever X is true or if Y is not the case. Most rules are general and apply to any suit. But there are some instances where a suit is used for special bids. Clubs are a particular problem. Thus, SMAJORS says the hand does not have a long major suit. The usefulness of this feature becomes clear later.

Several of the less important aspects aren't shown. The last few in the listing show trump support (aspect 13), and whether a rule is part of a convention. Thus, if a rule contained the coding BLACKWOOD, it would only be considered if the partners are playing that convention.

Aspect 14 is unique. Priority is used to resolve clashes among rules. It's an entirely subjective value, determined by the rule designer. It's not used when selecting rules to be evaluated. Rather it's used in the decoding phase.

Defining Suggested Bids

Rather than have the AI spit out a specific bid, I designed the system so it would give a more general result. Thus, the AI might suggest: "Bid the stronger of your two unbid suits," or "Bid your longer major." This allows the same suggestion to be made on many different hands.

Listing 2 shows some of the rules. Each result is a four-digit number. The 1000 series contains special cases. 1010 is a PASS, 1020 is a DOUBLE, and 1030 is a REDOUBLE. When these rules are found, no decoding is needed.

The 2000 series is used to indicate a new suit should be bid. This suit can be specified as either a relative or absolute level. Thus, some rules require that the bid be at the second level. Other rules require that it be two levels above the previous bid. Thus, a result of 2110 would

require a bid of your longest suit at the first level, while a result of 2010 would require a bid of your longest suit at the one level higher than the current bid. The 4000 series shows similar coding for a raise of previously bid suits.

Specify a specific bid, for example, 2 CLUBS when a convention calls for it. With these essential tools and descriptions defined, we can move to the design of the actual rules.

3. Embody the Representation in the Computer System. Transferring these concepts to a computer representation consisted of two steps: codification of rules and analysis of a hand. The designer specifies the rules external to the program, so the compiled program can be updated by adding a new rule set.

Listing 3 shows excerpts from the bidding rules (the actual game uses more than 250 rules for each bidding system). Each rule is given an internal mnemonic for easier debugging. OPEN-NB-1 is a rule for opening when no one has bid yet. It requires total card points of 13 to 15 (G13x15), at least a five-card suit, no bids by opponents or partner, and is given a priority of 4 (about average).

If this rule is chosen, it returns a result of 2011, which means to bid your longest suit at the first level. Thus, the rule will also work for a six-, seven-, or eight-card or longer suit if no more specific rule is found. HCPs are irrelevant (0) in this rule. This lets us use as many or as few of the aspects as we need and is an important feature of the AI's rule determination system.

Under the opener's rebids section, we'll look at two more examples. IFIT-GAMETRY says that when my partner opens (PO) and I supported the suit (IST), but we have less than 25 points total between us (LT25) and the bidding is below the fourth level, return 4113.

This decodes as an instruction to rebid the agreed suit at the third level. If the opponents have interfered and the bidding is higher, this rule won't apply. The next rule, IFIT-NT is similar. Again, my partner opens and I support the suit, but this time my trump support is fair, we have more than 25 points, and I have stoppers in the unbid suits. In this case,

Listing 4. AIBUILD (Continued on p. 50)

```
int AI_build_master_tran(int who)
{
int jcards[] = { 84, 74, 81, 75, 65 };       /* asc equiv of honors */
int side, partner, lopp, ropp, tp, lbid;
int x, z, i, j, k, nr, zlen;
char bidder[6];
int dcard, oppSuit;
   side      = getSide(who);
   partner   = getPartner(who);
   lopp      = getLHO(who);
   ropp      = getRHO(who);

   /* set aspect 1 (HCP) */
   if(hcp[who] >= 0 && hcp[who] <= 7)    setAsp(side,HCP,G0x7);
   if(hcp[who] >= 0 && hcp[who] <= 10)   setAsp(side,HCP,G0x10);
   if(hcp[who] >= 4 && hcp[who] <= 8)    setAsp(side,HCP,G4x8);
   if(hcp[who] >= 6 && hcp[who] <= 10)   setAsp(side,HCP,G6x10);
   if(hcp[who] >= 6 && hcp[who] <= 12)   setAsp(side,HCP,G6x12);
   ....
   if(hcp[who] >= 8)    setAsp(side,HCP,G8P);
   if(hcp[who] >= 10)   setAsp(side,HCP,G10P);
   if(hcp[who] >= 13)   setAsp(side,HCP,G13P);
   if(hcp[who] >= 16)   setAsp(side,HCP,G16P);
   if(hcp[who] >= 18)   setAsp(side,HCP,G18P);
   if(hcp[who] >= 21)   setAsp(side,HCP,G21P);

   /* set aspect 2 (TCP)   */
   .....
   /* same as for HCP      */

/* set aspect 3 (DIST)     */
AI_build_dist_pattern(who);

/* all hands with singleton or void are unbalanced */
if(pattern[3] < 2) setAsp(side,DIST,UNBAL);
if(pattern[3] >= 2 && pattern[0] < 5) setAsp(side,DIST,FLAT);
for (i=CLUBS; i<=DIAMONDS;i++) {
   if (dist[who][i] >= 4) setAsp(side,DIST,MINOR4);
   if (dist[who][i] >= 5) {
      setAsp(side,DIST, MINOR5);
      setAsp(side,DIST, SUIT5);
   }
}
for (i=HEARTS; i<=SPADES;i++) {
   if (dist[who][i] >= 4) setAsp(side,DIST,MAJOR4);
   if (dist[who][i] >= 5) {
      setAsp(side,DIST, MAJOR5);
      setAsp(side,DIST, SUIT5);
   }
}
if (pattern[0] >= 6) setAsp(side,DIST,SUIT6);
if (pattern[0] >= 7) setAsp(side,DIST,SUIT7);
if (pattern[0] >= 8) setAsp(side,DIST,SUIT8);

   /* set attribute for aspect 4 (OPP) */
if (statLHO[who] >= LHOP)   setAsp(side,OPP, statLHO[who]);
if (statRHO[who] >= RHOP)   setAsp(side,OPP, statRHO[who]);
if ((statLHO[who] <= LHOP && statRHO[who] <= RHOP) ) {
   setAsp(side,OPP, NB);
   statOpp[who] = BOP;
}
if (statRHO[who]     == RHOO   || statLHO[who]  == LHOO)  statOpp[who] = OO;
```

Listing 4. AIBUILD (Continued from p. 49)

```
if (statRHO[who]    == RHOV   || statLHO[who]  == LHOV)  statOpp[who] = OOV;
if (statRHO[who]    == RHOPR  || statLHO[who]  == LHOPR) statOpp[who] = OPR;
if (statRHO[who]    == RHONT  || statLHO[who]  == LHONT) statOpp[who] = ONT;
setAsp(side,OPP, statOpp[who]);

/* set aspect 5 (PARTNER) */
if(statPart[who] == PP) setAsp(side,PARTNER,NB);
setAsp(side,PARTNER,statPart[who] );

/* set aspect 6 (ME) */
setAsp(side,ME,statMe[who]);
if(statMe[who] == IP) setAsp(side,ME,NB);

/* set aspect 7 (VUL) */
if(vul[who] == 1)                       setAsp(side,VUL,WE);
if(vul[getRHO(who)] == 1)               setAsp(side,VUL,THEY);
if(vul[who]==1 && vul[getRHO(who)]==1)  setAsp(side,VUL,ALL);
if(vul[who] != 1)                       setAsp(side,VUL,NOTUS);

/* set aspect 13 (TRUMPS) */
setAsp(side,TRUMPS,0); /* reset */
z = getSuitBid(partner, 1);
if (z >= CLUBS && z <= SPADES) {
   switch(support[who][z])  {
      case POOR: setAsp(side,TRUMPS,POOR); break;
      case FAIR: setAsp(side,TRUMPS,FAIR); break;
      case GOOD: setAsp(side,TRUMPS,GOOD); break;
      case EXCEL:
         setAsp(side,TRUMPS,EXCEL);
         setAsp(side,TRUMPS,GOOD);
         break;
   }
}
if (getSide(lastBidder) == getSide( getLHO(who) )) {
   oppSuit = getSuit(lastCall);
   if (oppSuit >= CLUBS && oppSuit <= SPADES) {
      if (dist[who][oppSuit] > 4) {
         if (holding[who][oppSuit] > Q) setAsp(side,TRUMPS,EXCELOPP);
         else setAsp(side,TRUMPS,GOODOPP);
      }
      if (dist[who][oppSuit] > 5) setAsp(side,TRUMPS,EXCELOPP);
   }
}
   setAsp(side,PRIORITY,2);    /* set aspect 14 (PRIORITY) to 2 */
```

the program bids 3 NT. My partner then has the option of pulling to four of the suit or playing the 3 NT game.

Conventions are started in a similar fashion. Thus, STAY-2C-1 says that with eight or more HCPs and a four-card major, if the opponents haven't bid and my partner has opened with 1 NT, the result is a bid of 2 CLUBS with priority 8, and that this will be noted as the start of a STAYMAN sequence. Subsequent bids are extremely controlled and handled by hard-coded routines for each convention.

Similarly, JACOBY-MAJOR6 is triggered when I have a six-card major (SUIT6 + NMINORS) and any number of points. If the opponents haven't bid and my partner has opened with 1 NT, the result is 2051, which is the code for a transfer bid. This instructs the player to bid one less than the longest suit. If my suit is hearts, I bid diamonds; if it's spades, I bid hearts.

My partner must bid the next suit. My partner, with, presumably, the stronger hand bids the chosen suit first, and the hand remains concealed. Subsequent Jacoby conventional bids are handled by programmed functions. The logic for the response to a Jacoby bid is shown in Table 1.

All rules must be prepared in advance and live in a reference file called during the game. Whenever a bid is called for, the program builds a new transaction record, since information and status change during each round of bidding. One important element that changes is the estimate of the points held by others.

If my partner opens, I know he or she has at least 13 HCPs. But I also know the opponents have at most 27 points minus whatever I hold. Thus, each player has a different estimate at any time. These estimates by each player of minimum and maximum holdings for each other player are constantly updated. They're also available for the human player to reference.

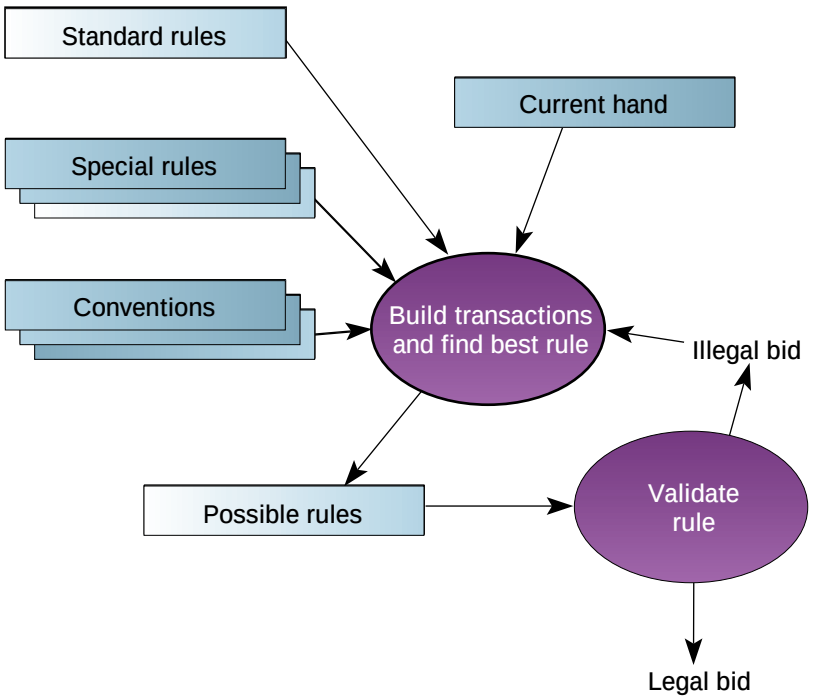
The construction of a master transaction is shown in Listing 4. Many terms are relative, so we start by finding the current values for our side, my partner, left hand opponent (lopp), and right hand opponent (ropp). Then, current values for each aspect are determined. For hcp, this is a series of compares. It's possible for a given aspect to take several values. Aspects are stored as bits. Each value of the aspect turns its bit on or off.

For distribution, we first check the pattern of the hand, sorting the cards by length of suit. pattern[0] holds the longest suit. We can set the aspects for minors, majors, five-card suits, and longer suits. In similar fashion, we work through the various status aspects. In some of these cases, multiple aspects are triggered.

For example, if the status of LHOP and RHOP is PASS, we also set the opponent status to BOP (both opponents passed). Alternatively, if either opponent has overcalled or opened, we can set that aspect. This increases the flexibility of rule writing and allows more concise rules, while retaining the ability to specify an opponent for those rules that require it. Usually, we don't care which opponent opened when we're deciding on a bid. But, if we're considering a double, it makes a difference whether it was the left or right opponent.

4. Create a search procedure. By now, most of the work is behind us. What remains is to find a rule that matches the

Figure 1. The AI Engine



current evaluation of the hand as closely as possible. This is done in the main routine of the AI engine. Since each aspect is a series of bits, we can test it against all the rules by performing a logical `AND`. The engine works by matching the master transaction that represents the individual hand against all possible rules, sifting out the ones that apply. The resulting list is then sorted by priority, and the best rule is applied.

A decode function then takes the result of the matching. If it's a direct bid (`PASS`, `2 CLUBS`, and so on), it just returns the bid. Otherwise, the bid is deciphered, and the result passed back. In either case, the bid is first checked to be sure that it is legal. Thus, if the current bid is `3 CLUBS`, and the suggested bid is `2 HEARTS`, the bid will be rejected, and the next potential rule will be considered. In practice, this is a rare occurrence. A summary of the AI engine is shown in Figure 1.

Completing the Bridge

Since the bridge world is tightly controlled by a limited number of cards and a restricted set of rules for bidding, it's possible to build a program that plays a credible version of the game. Whether the program is truly intelligent is less likely. Its very strength is a reason why broader AI fails.

The circumscribed description of the world causes the program to be blind to outside effects. The essence of intelligence is to act appropriately when there is no simple predefined problem or task domain in which to search for solutions. For these reasons, it's a giant, perhaps impossible step from a bridge game player to a common-sense program. This is almost a definition of common sense—we accuse people of lacking common sense when some situation has blinded them to a task space of potentially relevant actions. ■

Steve Estvanik is a professional game designer in Seattle, Wash. He can be reached via e-mail at 76703.3046@compuserve.com. General questions on this article can be left on CompuServe in section 11 (Games Design) of the GAMERS forum for open discussion.

MVP BRIDGE

For further gameplay, MVP Bridge is available in the file MVPBR.ZIP in LIB13 of the GAMERS forum on CompuServe.

What is 3D Studio?

by Larry O'Brien

The sheer number of skills necessary to exploit 3D Studio makes for a highly lucrative skill set—you'll find tools and techniques in this complex package to launch your game into another dimension.

There are two ways to get three-dimensional effects in your game. The first is by programming with high-performance three-dimensional algorithms. This is the only way that allows players to interact with environments in real time for simulators or arcade games. While fast bitmap manipulation à la Doom and fast texture mapping à la IndyCar Racing are vast improvements over the original Flight Simulator, photorealistic three dimensions updated in real time are years away.

How many years? About 20, if a trend that's held true for the past 20 years continues. Moore's Law states that every 18 months you get twice the computing power per buck. Today, it takes the typical home PC about 16 minutes to create a view into a moderately complicated scene. In 18 months that will be 8 minutes, in three years only 4 minutes, and so on. Now, keep in mind that this is for full-screen video at SuperVGA resolutions and 24-bit color—in five years or less, we should be able to achieve real-time three dimensional rendering in viewports that take up, say, a quarter of the screen.

What you can do today is render three-dimensional movements at development time, creating predefined paths through the game's environment. These animations can then be linked at run time to create turn-based games.

What Is It?

Autodesk's 3D Studio is a tool for creating and animating three-dimensional objects. Although the majority of broadcast and film computer animation is still done on UNIX workstations, such as those produced by Silicon Graphics, 3D Studio Release 3, which shipped in the second half of 1993, has a competitive feature list, runs on (relatively) cheap Intel-based hardware, allows for C-based extensions, and, with its network rendering features, blows away UNIX systems on a price-per-frame basis.

Who Uses It?

Typically, 3D Studio is used throughout the creative process. On the set of the upcoming Mechadeus production The Daedelus Encounter, we saw 3D Studio used in a number of ways:

- As a production design tool to demonstrate the dark look of the virtual sets the actors were moving through.
- As a three-dimensional storyboard to rough out interiors and character placement on the set floor itself. Instead of hitting marks taped to the floor of a set, actors Tia Carrere and Christian Bouchet moved against a bluescreen, were Chromakeyed against a 3D Studio created interior, and made sure their motions intersected two 3D Studio characters rendered in gold metal that looked mysteriously like Academy Award statues.
- As the only medium in which some creatures existed (Mechadeus modeled the creatures in clay, made computer wireframes of them in 3D Studio, and animated them there).

The most impressive thing was that everything we were seeing was flexible and contingent—the final virtual sets would be redone with higher resolutions, moodier lighting, and adjusted for the

motion subtleties of the human actors.

The ubiquitous nature of 3D Studio on the set points out a double-edged aspect of the product. 3D Studio is no magical shortcut to astonishing effects. Many roles of a traditional production lot have been captured in a single product and, while this makes it an amazing tool, just think of the roles a 3D Studio technician has to play.

Production Designer. You've got to design not just the virtual sets but every aspect of the virtual world, and, unless you're doing something like the Mechadeus project, you may be designing every single thing in that world, from the main characters' look to the molding on a picture frame hanging in the background of a single shot.

Props Master. Once you know all those things, you have to create them. For some things, you'll be able to use commercially available (but not cheap) meshes intended primarily for architects. You want a chair? You got it. You want a cyborg? You'll have to sculpt it in 3D Studio.

Director. You'll have to figure out the movement of cameras, objects, and actors within the virtual set.

Actor. There's no yelling "Action" and watching what happens in 3D Studio. You have to specify every discontinuous motion. However, once you specify a path, 3D Studio can automatically calculate the intermediate stages (tweens), so, to some extent, you only have to specify master scenes. Unfortunately, 3D Studio doesn't have the knowledge of the real world of a professional fill-in animator, so you can't say "he sneezes" and let someone else do the work—a sneeze may require 10 master scenes, manipulating dozens of control points on the eyes, cheeks, nose, and mouth.

Cinematographer and Lighting Technician. You place cameras, specify their lenses, and move them in pans and swoops (and some things, like complex rotations and extended zooms, that are impossible with real cameras). Lights work the same way and can be controlled in terms of color, drop off, and hotspots.

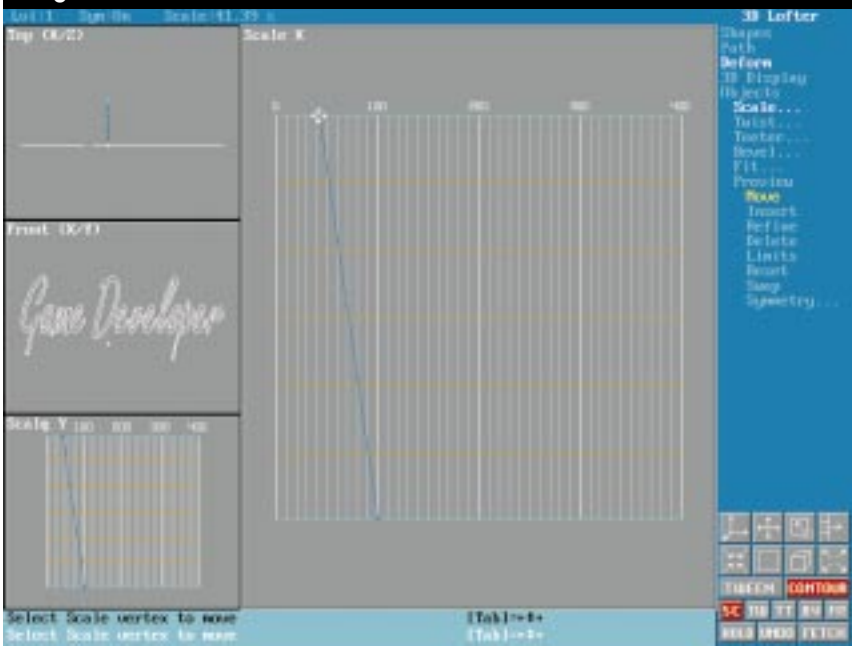
Editor. Cameras are invisible in 3D Studio, so you can place a dozen in a scene and create something with enough jump cuts to outdo Oliver Stone.

Printer. Finally, getting the final animation into a delivery medium (generally, a series of FLC or FLI animations that will be tied together with authoring

The latest Trilobyte release relied heavily on 3D Studio for its exciting visual elements.

Figure 1. The Shaper

Figure 2. The Lofter



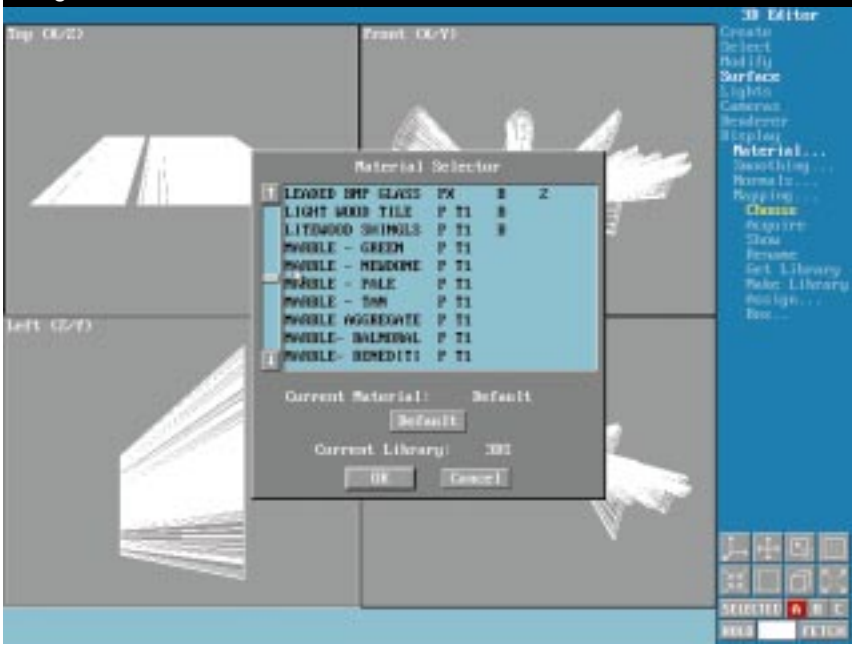
software such as Director) may not be a trivial matter. You may generate hundreds of large files that need to be tracked and organized.

One way or another, the sheer number of skills necessary to exploit 3D Studio fully represents a formidable barrier to entry but makes for a highly lucrative skill set. If you build up a portfolio of impressive 3D Studio animations, you'll have game companies knocking down your door trying to hire you.

Working with 3D Studio

3D Studio is a dongle-protected extended DOS program built with the Phar Lap DOS extender. Installation will take up the better part of 40MB of hard disk space, with a CD-ROM full of additional material thrown in for good measure.

Figure 3. 3D Editor



The Phar Lap DOS Extender is more stable than most and well supported, so, unless you're running really bizarre hardware, you should be alright.

3D Studio requires VESA graphics extensions, but most high-end graphics cards go beyond this to provide special drivers (it uses the same drivers as AutoCAD, so check your card's installation disk for those). One of the smartest things you can do first, though, is call Vibrant Technologies (US: 800-937-1711, Europe: +44-0-91478-1016, Taiwan: +886-2-593-5201) and buy the appropriate Liquid Speed driver for your video card. This will speed up rendering time significantly and, if you value your time at all, will quickly pay for itself.

Speaking of things that will pay for themselves, many 3D Studio users favor OS/2 for running the program. It's easy to configure (follow the instructions in 3D Studio's READ.ME file), there's no conflict with the DOS extender, and, most importantly, you can switch out of 3D Studio and use other programs while rendering in the background, which you cannot do under Windows. RAM requirements vary. The more complicated a scene and the more bitmaps used for texture mapping, the more RAM is needed. Some of the high-resolution examples on the example CD couldn't be rendered even by a system with 24MB of RAM. For high-end production, plan on 32MB or even 64MB of RAM.

I should emphasize here that all these requirements only apply to the developer. The animations produced by 3D Studio are just data files and can be rendered at a resolution appropriate for whatever target machine you choose.

3D Studio comes with a tutorial that, even though it takes several days to get through, gives only a cursory overview of the product. If you're a free-spirited soul with no time deadlines, you can get away with just jumping in and asking a lot of questions on on-line hotbeds of 3D Studio users such as CompuServe's ASOFT forum. But if you're under deadline, you'll certainly want to check out training resources such as AutoDesk University (Call 415-905-2354 for more information. In the "Don't

call me a whore" department, this is jointly produced by AutoDesk and Miller Freeman Inc., the company that publishes *Game Developer*) and the highly regarded book *Inside 3D Studio Release 3* by Steven Elliott, Phillip Miller, and Greg Pyros, by New Riders Publishing (I'd tell you more, but they haven't sent us a review copy—hint, hint!)

To create an animation in 3D Studio, you work through a series of subprograms. These programs are the Shaper, the Lofter, the Material Editor, the 3D Editor, and the Keyframer. The general interface for all these programs is similar, with a number of viewports showing different perspectives on the object and a telescoping menu system on the right-hand side.

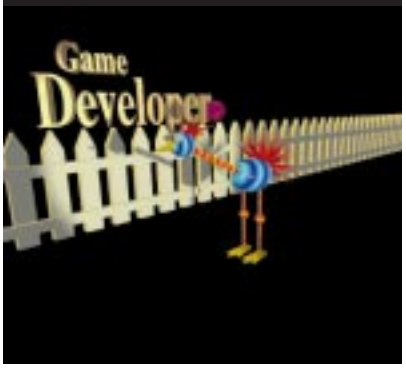
The interface wouldn't get any awards from a computer-human interface expert, but it does minimize the interference of the menu with the screen elements. The basic metaphor is to work down through the menu structure: create, select, modify, configure, and render. It takes a while to get used to that basic structure and familiarize yourself with the use of the various subprograms, but eventually the whole thing starts to seem almost coherent.

Shaper

The Shaper is used to create two-dimensional objects that serve as starting points for more complex structures. In Figure 1, you can see the single large viewport of the Shaper as well as the menu telescoping down the right-hand side of the screen. In this view, I'm in the `Create` mode, and, as you can see, I just finished with the `Create/Text/Place` option.

The text I've created ("Game Developer" in a calligraphic font) shows in the Shape viewport, with control points determining the vertices of the polygon. The numbers in the upper left-hand corner reflect the position of the mouse (represented by the crosshairs) and hide the various system menus that choose between sub-programs, load and unload projects, and configure the overall system. The control buttons in the lower right basically control the various zooming options as well as some selection and undo options. Finally, there is a message bar along the bottom of the screen.

Figure 4. Rendering an Image



The capabilities of the Shaper should be familiar to anyone who has used a structured, vector-based illustration tool. Basically, any object is made of straight lines and curves. If you want a really organic shape, you must spend a lot of time working on the details.

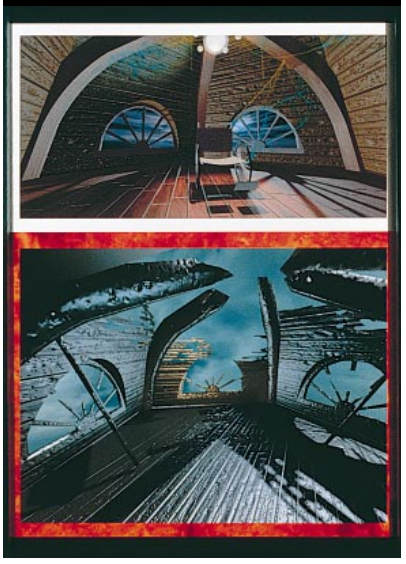
The Lofter

Once you've created a base shape in the Shaper, you turn it into a three-dimensional object in the Lofter. I think of the two basic options in the Lofter as extruding an object and lathing it. Figure 2 shows the Lofter as I specify how to extrude the logo I created in the Shaper. On the right, you can see I'm in the middle of a `Deform/Scale/Move` action. In the upper left corner, you can see I'm at Level 1 of the path along which I'll be extruding the logo, symmetry is on, and the smaller end of the logo will be 41.39% of the long end.

Along the left of the screen, you see a view looking straight down on the logo showing the path the object will be extruded from front to back. (If I'd chosen to lathe the logo instead of having it extruded from front to back, it would be extruded in a circle.) Below that, you can see the head-on view of the logo, and below that is the Y-scale that, because symmetry is on, will be deformed in the same way as the X-scale, which is shown in the large viewport.

I could continue to work on the path in either of the viewports, twisting the logo, making it expand or contract, and so on, along as many control points as desired. For me, this and the Material Editor are the hardest of the subprograms of 3D Studio to master. However, once you get what you desire, you hit a button and wait a few seconds while the object is created. Even a moderately complicated shape may have thousands of vertices, but 3D Studio has the ability to optimize the shape to simplify it.

Figure 5. A Complex Image



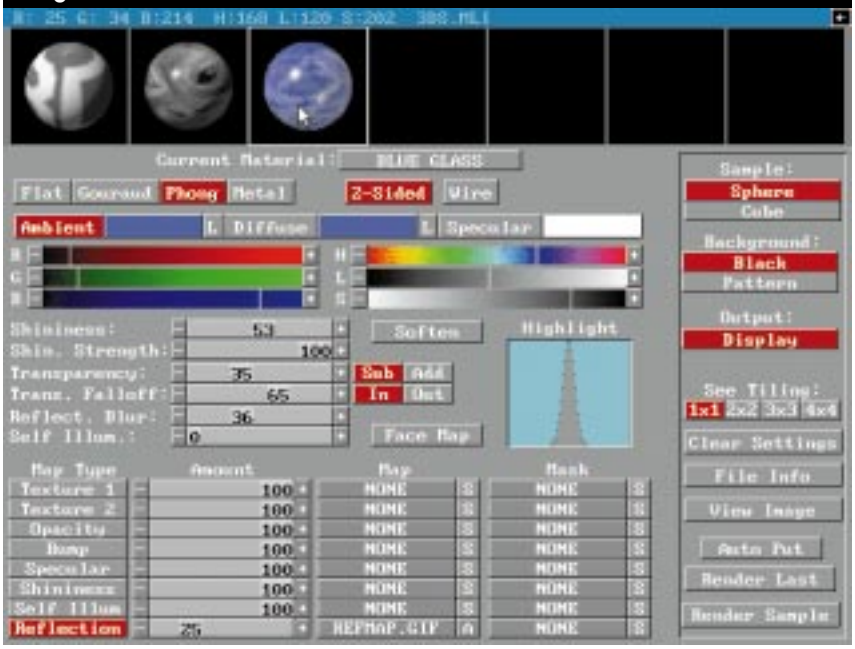
3D Editor

Once you're satisfied with the product of the Lofter, it's time to use the 3D Editor to set the virtual stage. Simple geometric figures such as boxes and tubes do not need to be created in the Shaper and Lofter, but can be created directly in the 3D Editor. 3D Editor is where much of the work in 3D Studio gets done. The default viewports shown in Figure 3 show a Top view, a Front view, a Left view, and a Camera view.

To do any three-dimensional manipulation, you generally have to manipulate the object in two viewports. Remember those tests you took in eighth grade that tried to determine whether you'd be an engineer, a doctor, or a homicidal maniac? It's sort of like one of those. (When I took those tests, I had significantly below-average tendencies toward everything. This brought me great acclaim in my peer group.)

After you've placed your objects, you assign materials to them. Materials can be uniform (plastic or metal), while others are textured and need mapping coor-

Figure 6. The Materials Editor



dinates to be applied. For instance, Figure 3 shows me selecting a Marble texture map for the logo. With mapping coordinates, you can rotate the map, project it out onto a sphere, or project it onto a cylinder.

This is also the point where you play cinematographer and lighting technician. There are a variety of lights that can be modified either by RGB or HLS values. The cameras come with stock lenses, but you can modify these to your heart's content.

Finally, you can render a still from this editor. You can configure the renderer to whatever output resolution you want (obviously, the lower the resolution the faster the rendering). On a 486/33 with 16MB of RAM running OS/2, the renderer took about two minutes to produce Figure 4 at 800-by-600 resolution. On the other hand, complex illustrations with bitmaps, such as that in Figure 5 from Trilobyte Production's upcoming 11th Hour, probably would take the same machine an hour or more to render.

Materials Editor

Although 3D Studio comes with a number of material libraries, eventually you'll want to move beyond them (especially when it comes to texture maps). For this, you use the Materials Editor. The Materials Editor sports an entirely different interface from the other subprograms of 3D Studio, as you can see in Figure 6. This interface is much more centered on dialogues and sliders.

In Figure 6, I have three materials that I am working with and have currently selected Blue Glass, the material in the third frame. To judge materials, they are rendered onto a sphere or a cube (I've selected the sphere) with a number of shading options. You adjust the ambient, diffuse, and specular colors (mid-left in the figure) to set the basic characteristic of the material and use the various modifiers in the center and bottom of the screen to create effects and specify the bitmap you want to be the basis of whatever texture map the material uses. When you're done with a material, you place it in a library, from which you can select it in the 3D Editor.

I mentioned previously that for me this is a difficult program to master, but I do not think that the different interface is entirely to blame for that (although it doesn't help). It's just that, of all the skills needed in 3D Studio, this is the one I have the least experience with.

Keyframer

Once you've created the set, it's time to play director. This is where the Keyframer comes in. Figure 7 shows the Keyframer working on an animation with the Game Developer text. (To speed up the screen refreshes, you can use the `Display/Geometry/Box` option to display all objects in their bounding box, as can be seen in the Camera viewport on the lower right of Figure 7.) In the lower right, there are now some VCR-style controls.

In Figure 7, we're at the 15th of 30 frames, and I've dollied the camera forward (you can see its original location in blue in the Left viewport). As you can see in the menu, I could have moved the camera anywhere in three-dimensional space, rolled it, or adjusted its field of view or perspective. Similar things can be done with objects and lights. 3D Studio takes care of tweening intermediate frames, and you can make sure that nothing is jerky by using spline-based paths for everything.

The first item in the menu is Hierarchy. This is a great aid in animation. You use the Hierarchy Editor to link together the various objects created in the 3D Editor. Once objects are linked, you can constrain their movement in a variety of ways. For instance, if you had a person, the thigh bone's connected to the shin bone, the shin bone's connected to the (you got it!) ankle bone, and so forth. Once you've linked them, you basically want to constrain those things so they only move through one plane (unless you're animating Joe Theissman getting tackled). This is easily done with the Hierarchy Editor. More complex hierarchies require the use of dummy objects, and, if you're interested in creating animations of complex objects, you'll be spending a good deal of your time mastering this feature.

Hierarchies are good for mechanical objects, but three-dimensional morphing gives better results for living creatures (or anything that you want to have a flexible skin). The morphing in 3D Studio is not to be confused with that used in those $100 morphing toys. Those are strictly two-dimensional (and some cheat and use simple bitmap fades instead of morphing, but that's another story), while 3D Studio will transform one three-dimen-

sional object into another as long as the two have exactly the same number of vertices (a nontrivial restriction). You can move the camera wherever you want during this transformation, so 3D Studio can handle *Terminator*-style shots where the camera follows the object as it moves by, transforming as it goes.

The Keyframer has a Preview option for creating low-resolution, non-color versions of the sequence, and you'll be using this a lot, because rendering a full scene is going to tie up your workstation (or workstations) for a long time. 3D Studio has become a hit with game developers because of its network rendering capabilities. For no extra money, you can install 3D Studio on a number of machines and, when you render a scene, your master workstation will send out the job to the slaves.

Every time one finishes a frame, it is given a new one. So you still can't generate a frame at the rate of a UNIX workstation, but if you're generating 10 frames in parallel, you can achieve much higher overall throughput. The slave machines can be anything—a dozen rack-mounted 486DX4s would make a nice system and could be put together for less than the cost of a single loaded UNIX workstation (and, if you placed it near an air duct, could serve as central heating for your offices). Such a system is called a "rendering farm," (doesn't that phrase make you think of *Silence of the Lambs*?) and is becoming a standard way of generating animations.

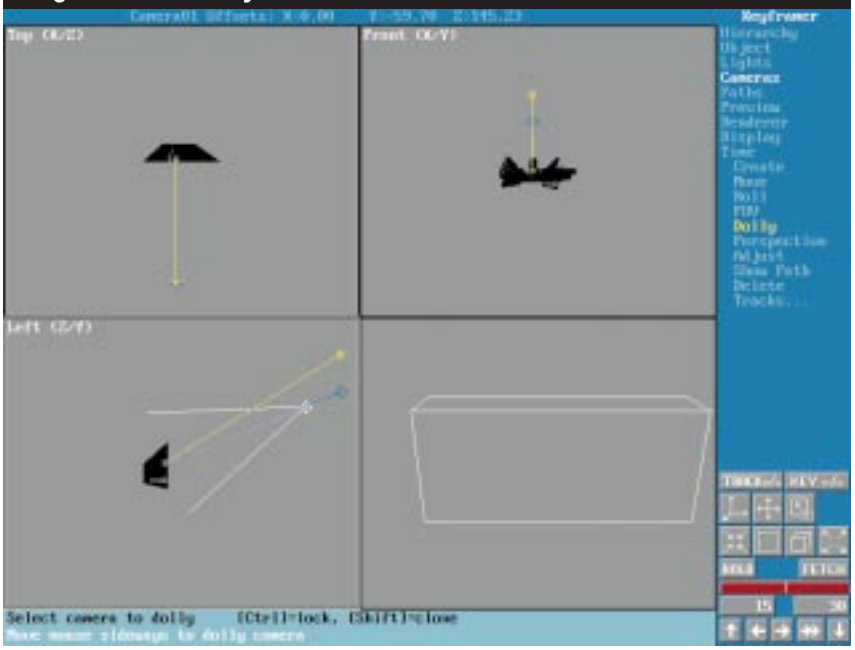
Extending 3D Studio

No product can do everything, especially in a field as rapidly evolving as computer animation. Luckily, 3D Studio provides for this with external C processes called IPAS routines. Many such routines are available from such companies as the Yost Group Inc. (you can ask for a catalogue by writing to them at 3739 Balboa St. #230, San Francisco, Calif. 94121).

Some of the routines they provide include special effects, such as flip, clamp, crumple, smoke, water, and spatter; procedural modeling, such as twist, stretch, reshape, and melt; image processing, such as lens flare, blur, glow, and highlight filter; and—the hot topic—particle system procedures for snow, rain, explode, disintegrate, fireworks, and spurt.

We'll be reviewing existing IPAS libraries as well as telling you how to build your own in future issues of *Game Developer*, but that all lies outside a general review of 3D Studio. So, if you'll excuse me, it's 6:00 p.m. on a Friday night, and I don't have a network of Pentiums slaved to my system, so I have to fire up the renderer and get started on the Klingon warbird flyby. Maybe it'll be done by Monday. ■

Figure 7. The Keyframer



Larry O'Brien is the editor of Game Developer. *He's also been known to program for Windows in C++, write the occasional screenplay, and play Ultimate Frisbee for 40 hours straight, none of which are options on eighth-grade career aptitude tests. He can be reached via e-mail at 76702.705@compuserve.com or through* Game Developer *magazine.*

3D STUDIO RELEASE 3

Autodesk Inc.
2320 Marinship Wy.
Sausalito, CA 94965
(800) 525-2763

Price:	**$2,995**
Run-time fee:	**N/A**
Support:	**Only available through dealer channel (approximately 500) who provide support**
Money-back policy:	**90-day limited warranty**
Operating system support:	**MS DOS 3.3 or later**
Hardware requirements:	**8MB of RAM, Intel math coprocessor, SuperVGA display device (at least 640 by 480 by 256), and pointing device (mouse, SummaSketch-compatible table, or ADI pointing device)**
Minimum hard disk space:	**20MB**