gd

GAME DEVELOPER MAGAZINE

JANUARY 1999

UBM Tech

Back

Spielberg Switches to Panaflex Cameras!

Your first reaction to that headline (besides the obvious, "What the heck is this doing in *Game Developer* magazine?") should be, "Who cares?"

The answer is: nobody — at least, nobody who's not deep in the film business, selling or operating cameras. Yet, if that headline had been "XYZ Games Switches to the 123 Engine," you'd be jumping out of your seat, eager to know why XYZ dropped the famed ABC Engine for the 123 Engine. Was it money? Features? Support?

As developers, we're analogous to the people in the film business mentioned above. That is, we actually have some excuse for caring about the intricacies of game development, and it's real news to us when a developer chooses one technology over another. We want and need to know the gory details. Unfortunately, the sad state of our industry today is that we're just as likely to see an engine-oriented headline in our popular press as in our technical trade press.

I believe this press focus on game engines alienates most players. Moreover, this engine-mania is distracting us from our real job of bringing our industry out of the garage and developing it into a mature artistic medium. It's easy to see why the press focuses on game engines in interviews, articles, and reviews. Writing about technology is easy, especially when that technology is conveniently wrapped up in a spiffy engine name by the developers, sometimes even before the actual game has a name. It's much easier to write about the engine than it is to write about the actual important features of games, such as game design, playability, and experience. The vocabulary is already present for in-depth discussions of game technology with our light maps, polygons-per-second, MIP-mapping, and what have you. Contrast this with the current state of the art in vocabulary for game design and playability: "Uh, was it fun?"

We can't just blame the press, of course, because we developers aid and abet them in their laziness by talking up a storm about our engines and technology. We answer their endless questions about technology instead of getting them back on track to talking about the actual game we're developing. Worst of all, we put out self-congratulatory press releases that focus specifically on our technology and don't even mention our games. Again, I believe we do this because it's easy to talk about technology, whereas we lack the vocabulary to discuss game design effectively.

Now, I know technology is intimately related to game design — especially at this early point in our industry's maturity — and I don't think we should completely avoid "talking tech" for our hardcore fans. However, when the first question out of the interviewer's mouth is "What engine does the game use?" we have a problem.

I love game technology as much as the next developer, but I realize that the game press is going to follow our lead when it comes to discussing our games. We can steer interviewers away from the easy techno-centric questions and back to how the game will play and what experiences the player will have while playing. We *are* differentiating on non-technology features, right?

The idea that we need to develop a vocabulary for game design is not mine. Doug Church from Looking Glass Technologies has lectured on the topic at the CGDC (now the GDC), and I believe the concept is mentioned as far back as Chris Crawford's *The Art of Computer Game Design*. I'd like *Game Developer* to provide a forum for creating this vocabulary, and if you've got ideas on the topic, or if your company's game designers have their own language that they use to communicate, we're interested in hearing about it. ■

Chris Hecker, Editor-At-Large

[illegible]

ON THE FRONT LINE OF GAME INNOVATION
Game Developer

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.4962 w: www.gdmag.com

Publisher
Cynthia A. Blair *cblair@mfi.com*

EDITORIAL

Editor-in-Chief
Alex Dunne *adunne@sirius.com*
Managing Editor
Tor D. Berg *tdberg@sirius.com*
Departments Editor
Wesley Hall *whall@sirius.com*
Art Director
Laura Pool *lpool@mfi.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook id Software
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Alicia Langer *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager
Kim Love *klove@mfi.com* t: 415.905.2175
Sales Associate/Recruitment
Ayrien Houchin *ahouchin@mfi.com* t: 415.905.2788

ADVERTISING PRODUCTION

Vice President Production Andrew A. Mickus
Advertising Production Coordinator Dave Perrotti
Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann
MarComm Manager Susan McDonald
Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President Cirulation Jerry M. Okabe
Assistant Circulation Director Mike Poplardo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
President Libby Abramson
720 Post Road, Scarsdale, New York 10583
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

Miller Freeman
A United News & Media publication

Chairman-Miller Freeman Inc. | Marshall W. Freeman
President/COO | Donald A. Pazour
Senior Vice President/CFO | Warren "Andy" Ambrose
Senior Vice Presidents | H. Ted Bahr, Darrell Denny Galen A. Poss, Wini D. Ragus, Regina Starr Ridley, Andrew A. Mickus, Jerry M. Okabe
Vice President/SD Show Group | KoAnn Vikören
Senior Vice President/Systems and Software Division | Regina Ridley

BPA International Membership Applied for March 1998

www.gdmag.com

VTune 3.0 Debated

In the October 1998 issue of *Game Developer*, Dan Teven's review calls Intel's VTune 3.0 "a mandatory upgrade" from version 2.5. I say it's lame and inexcusable. Here's why:

VTune 2.5 is an excellent tool for optimizing Pentium MMX assembly code. It flags your pipeline stalls and explains the quirky details that derail instruction pairing. In blended CPU mode, it also warns you of lethal Pentium II glitches — code that runs perfectly well on a Pentium MMX, but brings a Pentium II to its knees. Other than flagging these elementary flaws, VTune 2.5 has little to say about how your code actually performs on a Pentium II. All it can show is whether or not the instruction decoder is choking on micro-ops.

With VTune 3.0, Intel had an opportunity to build a tool that would enable programmers to analyze the micro-op pipeline flow of Pentium II code. Instead, they lavished their efforts on redesigning their Gratuitous User Interface into something resembling a web browser. Question for Intel: I've already got a clumsy web browser and a monolithic IDE hogging too much screen space; am I supposed to buy a second monitor just for you?

To make matters worse, Intel chose to drop the blended CPU mode from VTune 3.0, which limits its analysis to just one CPU at a time. This forces you to switch back and forth between modes to make sure that an optimization for one CPU doesn't clobber another. VTune's worst offense, however, is the misleading way it assigns sampling percentages to assembly language instructions that have absolutely nothing to do with the event that is being monitored. What we really want to know is which micro-op, generated by what instruction, triggered the event in question. Intel, however, would prefer not to discuss micro-ops with developers — they've actually censored the original Pentium Pro micro-op documentation, and refuse to update the Pentium II docs to fill in the gaps.

The Pentium II is theoretically capable of executing three micro-ops per CPU cycle — essentially 50 percent more throughput than the Pentium MMX. This level of performance can only be achieved, however, by careful analysis of micro-op scheduling and execution port utilization in critical inner loops (not to mention cache misses and branch mispredictions). Rather than assisting developers in this effort, Intel has maintained a wall of misinformation that obscures what's really happening inside your code. VTune 3.0 does nothing to breach this wall.

Lee Powell
Machine Code Systems
via e-mail

TEVEN RESPONDS: *I should have noticed that Intel dropped the blended CPU analysis mode, and I hope they bring it back. I haven't changed my opinion, however. The improvements in VTune 3.0 are an order of magnitude more important than this small loss.*

The VTune 3.0 CD contains the Pentium Pro family documentation, including the Intel Architecture Optimizations Manual and the three-volume IA Software Developer's Manual. These references talk quite a bit about micro-ops. They also contain Intel's recommendations for writing "blended code," which I will pass along:

- *Important code entry points, such as a mispredicted label or an interrupt function, should be aligned on 16-byte boundaries.*
- *Avoid partial stalls.*
- *Schedule to remove address generation interlock and other pipeline stalls.*
- *Use simple instructions.*
- *Follow the branch prediction algorithm.*
- *Schedule floating-point code to improve throughput.*

Your desire to analyze the micro-op flow of Pentium II code may be valid, but it's unfair to blame VTune for not letting you do it. VTune relies on the event counters that are built into the processors for this level of analysis, and the Pentium II doesn't count anything but micro-ops retired. If there are undocumented counters in the Pentium II, then be upset with the Pentium II, not the profiler.

Financing Creativity

I read Alex Dunne's September editoral on the "Sundance Festival" for video games with some interest. The problem: It's not particularly difficult to get companies such as MGM Interactive, Activision, Interplay, or Sony to look at game demos, designs, and prototypes. What's difficult is getting money for anything remotely creative, and this is where we've failed miserably. Whether we ask for $500,000 or $100,000 in funding, the answer is always no. Fortunately, magazines such as *Next Generation* and game players themselves seem to disagree with their views on the subject, but that hasn't brought us any closer to discovery.

I suspect that the reason we have had such trouble is that game suits are a scared bunch. The dirtiest secret of this industry is that 34 out of 35 games lose money, and that only 1 in 140 hit it big. These odds make Las Vegas slot machines look like an attractive investment. Dodging all risks, suits follow a strategy similar to the music industry's, and chase last year's hits: year after year after year. Ask yourself which is a sounder investment: one 20 million dollar DOOM clone championed by a bunch of disgruntled id employees, or 80 $250,000 "art flick" games by 80 different unknowns and semi-unknowns?

Me? I'm a firm believer that the video game market is becoming as efficient as the stock market — it's increasingly difficult to get something for nothing. The game suits simply cannot *a priori* pick the hits, and they should give up trying to do so, diversify their development portfolios, and wait for the hits to emerge rather than try to predict them. Such a strategy couldn't work any less well than what we're currently seeing.

Scott LeGrand
via e-mail

CORRECTIONS FOR NOVEMBER:

The following equations appeared incorrectly in Andrew Flavell's "Run Time MIP-Map Filtering" in the November issue of *Game Developer*. Apologies for any inconvenience.

$$u_y = \frac{Z.dUOverZdY - UOverZ.dOneOverZdY}{Z^2} = \frac{e - ay}{Z^2} \qquad \text{Eq. 12}$$

$$v_y = \frac{Z.dOverZdY - VOverZ.dOneOverZdY}{Z^2} = \frac{f - by}{Z^2} \qquad \text{Eq. 13}$$

$$Compression = \frac{1}{Z^2}\max(x_\ell, y_\ell)$$

where

$$x_\ell = \left(\sqrt{(c+ay)^2 + (d+by)^2}\right)$$

$$y_\ell = \left(\sqrt{(e-ax)^2 - (f-bx)^2}\right) \qquad \text{Eq. 20}$$

BIT Blasts

News from the World of Game Development

New Products: The new Miles Sound System 5.0, Rhinoceros does NURBS, and Roger Wilco comes in loud and clear. **p. 9**

Industry Watch: Wars and profits, plus Mark Miller's report from Project Bar-B-Que **pp. 10-13**

Product Reviews: Okino's PolyTrans and MultiGen's Creator **pp. 14-18**

New Products

by Wesley Hall

The New Miles 5.0 is Out

RAD GAME TOOLS has once again done a major feature-upgrade on the venerable Miles Sound System, the all-inclusive audio tool for developers.

The 5.0 version of the Miles Sound System has a ton of new features, but the big ones are MPEG Layer-3 support, a high-level 3D audio API, and a new digital sub-system. The Miles 5.0 MP3 support includes the abilities to decompress data from your sound mixer, to stream massive MP3 sound files off of your hard drive or CD-ROM, and to play MP3 compressed instruments in DLS files. With Miles's 3D audio API, your game will support all of the commom 3D technologies, such as DirectSound3D software and hardware, Aureal's A3D, Creative's EAX, and Intel's RSX. You can switch between any of the technologies at runtime, or load multiple technologies at once. Further, future 3D providers can be dropped into your game without source code changes because the 3D technologies are abstracted from Miles's API. The API also allows you to perform 3D sample sub-block looping. The new digital subsystem allows Miles to use its own mixer even when running under DirectSound, and also includes MMX optimizations that have been added to the digital mixer subsystem.

The Miles Sound System SDK is available for DOS, Windows 3.x, Win32s, Windows 95, and Windows NT. The Miles Sound System SDK is licensed on a per-product or per-site basis with no royalties. Single product licenses start at $3,000.

■ **RAD Game Tools Inc.**
Kirkland, Wash.
(425) 893-4300
http://www.radgametools.com

Ocean Quigley modeled this rhino using Rhinoceros, and then rendered it in 3D Studio MAX.

Rhino NURBS

ROBERT MCNEEL & ASSOCIATES just unleashed Rhinoceros (Rhino), the first full-power NURBS (non-uniform rational B-spline) modeler for Windows.

Rhino is a conceptual design and modeling tool that allows you to create objects that are smooth NURBS surfaces rather than line segments or polgon meshes. In the past, this technology was only available on high-end workstations. Organic effects are now much more accessible. You can play in a free-form world as you create and edit curves, surfaces, solids, meshes, and (of course) 3D models. Rhino also integrates solids (surfaces joined together at their edges) and surface modeling so that solids can be exploded into surfaces, edited, and then joined together again. Spline-based models created in Rhino can be used in most rendering and animation products, including 3D Studio MAX, Softimage, and LightWave 3D.

Rhino runs on a system with a Pentium processor, 32MB RAM, and 15MB disk space under Windows 95/98/NT. No special graphics software is required, although Rhino is compatible with a wide variety of 2D and 3D file formats. It retails for $795.

■ **Robert McNeel & Associates**
Seattle, Wash.
(206) 545-7000
http://www.rhino3d.com

Online Radio Chatter

RESOUNDING TECHNOLOGY INC. just debuted Roger Wilco, a stand-alone application that allows online game players to talk to each other while playing their favorite multiplayer games such as QUAKE II, DIABLO, and STARCRAFT.

Roger Wilco is a companion technology that enables players to actually talk to each other instead of typing messages. In its initial release, Roger Wilco supports up to four players without requiring a dedicated server. It will support ten in the near future. Players can now communicate without interupting game play. When other players speak, their transmissions are mixed with the game audio. Unlike Internet phone programs, Roger Wilco is designed specifically for running alongside multiplayer online games without requiring an additional soundcard.

Roger Wilco works with most Windows 95/98 games that use DirectSound. It requires a microphone and a soundcard capable of full-duplex audio. It also requires a Pentium 166 or better processor, 2MB disk space and at least a 28.8K modem. Roger Wilco is currently in beta test. A free "technology preview" is immediately available for download from Resounding's web site.

■ **Resounding Technology Inc.**
Cambridge, Mass.
(512) 248-9344
http://www.resounding.com

Industry Watch

by Alex Dunne

THANKS TO SOME RECENT RULINGS in the litigation between Creative Labs and Aureal Semiconductor, this case is going to trial. First, the court rejected Aureal's motion for summary judgment, rejecting Aureal's limited interpretation of Creative's patent on a "Digital Sampling Instrument Employing Cache-Memory."

UPCOMING EVENTS CALENDAR

January 4-8, 1999

MacWorld Expo 99
Moscone Center
San Francisco, CA USA
Cost: $45 expo only, $345 and up for conference.
http://www.macworldexpo.com

January 7-10, 1999

1999 CES
Las Vegas Convention Center, Sands Expo & Convention Center, Las
Vegas Hilton, Alexis Park Hotel
Las Vegas, NV
Cost: $75
http://www.cesweb.org

February 9-12, 1999

Milia 99
Palais des Festivals
Cannes, France
Cost: $655
http://www.reedmidem.milia.com

February 17-20, 1999

TED 9
Monterey Convention Center
Monterey, CA
Cost: $2250
http://www.ted.com

In the second ruling, the court rejected Creative Labs' motion for a preliminary injunction against Aureal. Aureal argued that a preliminary injunction prohibiting the sale of their Vortex chips would essentially force the company to close, and the court ruled felt that such action would be too severe in the event that Aureal eventually emerged victorious from the trial. In other words, in this trial Aureal has much more to lose than Creative Labs has to gain.

GT INTERACTIVE ACQUIRED ONEZERO MEDIA, a two-year-old company which produces the TV show The Wild Wild Web and its accompanying web site, http://www.getwild.com. GTI was impressed with OZM's success in creating the cross-media franchise, and said that the purchase of OZM will further its growth strategy by "providing an established platform for e-commerce and for promoting new entertainment products and content." OZM will operate as a wholly owned subsidiary of GTI.

MICROSOFT MADE AN UNDISCLOSED minority investment in online game developer VR-1, the company that developed FIGHTER ACE for the MSN Gaming Zone. Not too long ago, VR-1 agreed to develop a pair of games exclusively for America Online, and while the full terms of the deal weren't disclosed, we know that the investment will not affect VR-1's ability to work with other partners, and that Microsoft won't take a seat on the VR-1 board.

ELECTRONIC ARTS EXTENDED its publishing agreement with Sid Meier's company, Firaxis. EA said that it will publish up to two more as yet unannounced Firaxis games. The original agreement, announced in 1996, allowed EA to publish SID MEIER'S GETTYSBURG! and SID MEIER'S ALPHA CENTAURI.

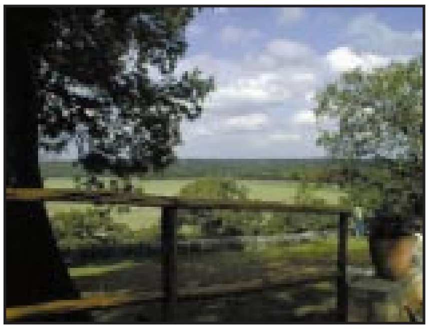
Acclaim's fourth quarter was highlighted by the success of WWF WARZONE.

OneZero Media's web site and accompanying television show attracted the interest of Ron Chaimowitz.

ACCLAIM ENTERTAINMENT REPORTED their 1998 fiscal year results, which ended August 31, 1998. The company saw net revenues of $326.6 million compared with $165.4 million in the prior year, resulting in a profit of $20.7 million. During the fiscal year, Acclaim published ten titles that sold over $10 million each, FORSAKEN, TUROK: DINAOSAUR HUNTER, RIVEN, and WWF WARZONE, among others.

MIDWAY PROFITS UP. Midway reported that for its fiscal quarter ending September 30, 1998, its revenues were more than 21 percent higher than last year, amounting to $89.3 million. Net income rose to $9.8 million, compared to last year's $7.2 million. Mirroring the pickup in console sales and slowdown in the arcade business, Midway's next generation video game revenues increased 106 percent and its coin-operated video game revenues for the quarter decreased over 40 percent over the previous year, to $19.6 million.

WITH THE STAR WARS PREQUEL DUE OUT on the silver screen shortly, Nintendo scored a sizable victory over rivals Sony and Sega by announcing that it has exclusive console rights to LucasArts' next three games. The five-year, worldwide agreement includes STAR WARS: ROGUE SQUANDRON (slated for release by Christmas 1998) plus two upcoming games based on George Lucas' next feature film, *Star Wars: Episode I: The Phantom Menace.*



Tales from a Cook-Out

by Mark Miller

In the middle of last October, I attended the third annual Project Bar-B-Que, "an incredibly intense Texas-style think tank." The event was hosted by none other than George Sanger, a.k.a. The Fat Man. Now, despite all of his *meshugas* (a Yiddish word roughly meaning "antics," derived or at least deeply linked, I think, to *meshuganah*, another Yiddish word roughly meaning "crazy person") George is someone whom I've come to like and respect for the efforts that he and his team put into this conference to the benefit of the entire industry.

So, you may ask, what is the Bar-B-Que all about? Well, as far as I can tell, it all started about four years ago. George and I don't exactly agree on the specifics of *what* happened, but we can pinpoint a moment in time *when* it happened. George had written an article in *Music & Computers*, the main theme of which went something like this: "Things are really disorganized when it comes to audio on computers. People are trying in vain to bring order to this chaos by the grindingly slow process of getting everyone to agree on standards. I would like to propose instead that I go off with a couple of really smart people and just come up with the answer and present it to everyone as the .FAT file format."

I was the co-chairman of the Interactive Audio Special Interest Group (the IA-SIG) at that time. We were the people who were deeply enmeshed in trying to develop those standards. George was, in fact, one of our working group chairmen. I buttonholed George in the hallway at some trade show, and we had a conversation. On this much of the story, we agree.

The content of the conversation is where our recollections diverge, dramatically. I remember being fairly bent out of shape and giving George a really hard time for "dissing" the IA-SIG, of which he was a chairman, in print. George remembers it as an inspirational sharing of ideas, out of which the concept for the Bar-B-Que was born. We both also remember something about George waving flags, but the significance of this act now escapes me completely.

"Is that baby back ribs that I smell or is that George starting up the Rolls again?"

So, you may ask again, what is Bar-B-Que all about? George's idea was to create an executive-style (read "kind of expensive but, boy, they treat you nice") retreat where a small (50 or fewer) group of people with some degree of responsibility over what you hear when you turn on your computer would get together to figure out how to make what you hear when you turn on your computer sound better. This group would comprise composers, sound designers, operating-system vendors, sound driver gurus, hardware designers, marketers, and philosophers. The retreat would put all of these people in one fairly remote place for three days of provocative presentations, heated conversations, massively parallel group decision making, conclusion drawing, and of course some really hardcore two-fisted drinking. When all was said and done, out the other end would come a document called the Bar-B-Que Report, which would be required annual reading for anyone with a professional interest in computer audio.

When this event was first described to me, I was skeptical to say the least. This was crazy talk (another form of *meshugas*). What kind of results could possibly come from such an unstructured process with some guy in sequins and a cowboy hat waving a flag around?

Perhaps there is some truth in the notion that there is a unique reality for each human being based upon his or her own individual perceptions of any event. And perhaps it is precisely these differences in people's personal realities that bring forth all that is truly useful and new. But true or not, I'm really glad that George remembers things his way and not mine because somehow out of this uniquely perceived conversation, a really useful conference was born.

Later on in the year, when this conference was on its way to becoming reality, George and I did reconcile our perceptions. George would host Bar-B-Que, and wild, unorthodox ideas and processes would be employed in order to frame a picture of the next five years in computer audio. The IA-SIG would then pick through the conference output and take over the process of grinding these bits and pieces of inspiration into publishable standards and recommended practices.

The first two Bar-B-Ques generated a lot of fascinating speculation about how the future would sound. In itself, the experience was inspiring and refreshing to all involved, but somehow hard to convey to nonparticipants. Fortunately, some more tangible, near-term initiatives were also produced. Two IA-SIG working groups, the Platform Development Working Group (PDWG) and the Audio Advisory Working Group (AAWG) were born. The work of these groups influenced both the AC97 and PC98 standards. Significant progress was made on interactive composition standards some of which made its way into DirectMusic. And finally, the idea that computer audio systems should be as easy to use as a toasters (the credo of the Church of Appliantology) was explored and has taken root in the

Mark Steven Miller produced audio for way too many games for way too many years. Currently, he's in charge of geological plasma extraction ('Squeezing blood from stones' or 'U.S. business development') for Harmonix Music Systems in Cambridge, Mass. Mark serves on the Steering Committee of the IA-SIG, the Advisory Board of the Game Developer's Conference, and is a frequent writer and speaker on the topic of interactive audio. He can be reached at mark@harmonixmusic.com.

minds of many of the hardware and software designers who attended.

This year, the conference was light on composers and sound designers and heavy on hardware types. This demographic put much of the focus on the next 6 to 18 month product cycle rather than on the five-year plan targeted in previous gatherings. Attendees formed four main working groups. The first went off to define the next generation of MIDI, tentatively called M1D2. The group decided that MIDI, as we know it, works pretty darn well; but it's old and many problems have been identified and not corrected. M1D2 will correct these problems and bring MIDI into the modern world of USB and 1392 (a.k.a. FireWire), for example.

The second group attempted to make sense of multiformat audio. The group's basic conclusion took the form of a chart that listed all of the ways that audio can exist within a software program (mono, stereo, Quad, ProLogic Stereo, 5.1, and so on) on one axis and all of the possible speaker configurations (headphones, stereo, ProLogic, Quad, and so on) on the other. The group's members filled in the resulting matrix with how all those inputs should be matched to all of those outputs so that the results were predictable, sensible, and sounded good.

A third group went after the most contentious of all of the problems. It took on the task of harmonizing the next set of extensions to 3D audio beyond, "Put the sound source here and put the listener here." More specifically, these extensions would deal with the listening environment; in other words, "Put the sound source here and put the listener here, *and* put them both in a 200×300-foot stone cathedral." At the outset, there were two opposed and "dug-in" positions. One group said that the actual room geometry must be mapped and the real physics of any sound therein recreated. The other group said that generalized room descriptions (reverb presets) were simpler and good enough by far. This conversation had actually been underway for months in the 3D Audio Working Group (IA-SIG 3DWG), but the "face time" occasioned by Bar-B-Que generated greater mutual understanding and large steps forward toward a unified standard. One 3D world is enough for all of us....

The fourth group (I was a member of this group) went off into the stratosphere a bit. We decided that it wouldn't be possible to figure out what consumers might want in five years. The best that we could do, then, would be to try to create a tool that would help one understand why people bought anything in the first place. To this end, we listed all of the basic motivations that would make a consumer consume. We then divided them into categories using a cool data structure called Maslow's Hierarchy. (If you're unfamiliar with this, look it up. You'll learn something about who is buying all of those sub-$1,000 PCs.) This data ended up in a spreadsheet, which now serves as a useful tool for constructing a value proposition for something that may not yet be technologically possible to produce. Finally, we created a list of enabling characteristics that could help one understand why a particular packaging of that "something" might succeed or fail in the marketplace. The whole ordeal was kind of abstract overall, but a really good workout for the right brain.

Traditionally, Bar-B-Que participants who may be displeased with the program are encouraged to form "Rogue Groups." This year, a Rogue Group formed to deal with the possibility that certain companies (such as, say, Microsoft) might choose to make style-based automatic music scoring systems a part of their operating systems. Rogue Group members felt that experienced game composers should define a list of presets for such a system. The list would include the smallest number of styles that would cover the largest number of possible video game levels. Once the Rogue Group had framed the problem in this way, the answers were readily apparent and the document took about 30 minutes to create.

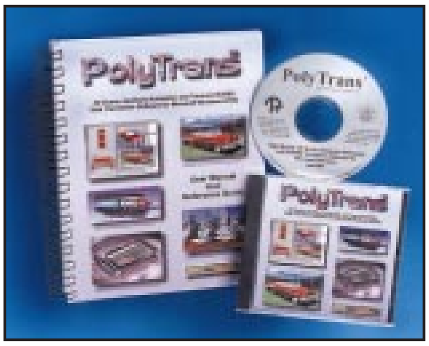
The resulting list was presented to a the video game producers in attendance. They looked it over and said, fairly consistently, that about 80 percent of their current projects needs would be met by such a set of presets. This was clearly an idea who's time had come. Recommended Style Presets included Ice World, Jungle World, Pirate World, Circus World, Underground World, Falling World (Forced scroll world), Moon World (Altered Physics world), Mechanical or Factory World, Urban World, Desert World, Cave World, Alien World, Medieval World, and Options Music

Creative Labs' Jean-Mark Jot (right) and Microsoft's Aaron Higgins (left) talk game audio at Project Bar-B-Que.

If you really want the details of what I've just careened through in these five short paragraphs, you'll simply need to pick up the report when it comes out. (You can find it online at http://www.fatman.com.) But let me conclude by saying this: Being 33 years old, I came in just too late on a number of things in this life. I missed the '60s and the early '70s. Instead, all I had for my teen years were goofy clothes, goofier music, and the empty ambition of the Reagan era. Professionally, I missed the early days of Atari and the heyday of the 8-bit Nintendo, when two guys in a garage could get rich writing games. (In retrospect though, cocaine and hookers were never my style, so I guess Atari wasn't such a big loss.) Fortunately, I haven't missed the beginnings of Bar-B-Que. It's a truly outstanding experience to have the opportunity to think hard, work hard, drink hard, and get something done quickly for a change. If there's one overarching message from The Ranch, it is this: Put the right people together, take away their company logos and cell phones, feed them well, stay out of their way, and good things will happen. ■

FOR FURTHER INFO

For information on attending Project Bar-B-Que, please contact:
Project BBQ
Executive Director: Teresa Avallone
P.O. Box 9726
Austin, Texas 78766
(512) 473-3878
e-mail: spanki@outer.net



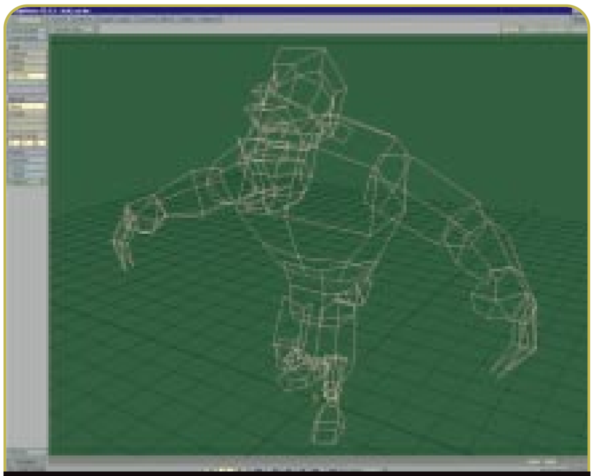
Okino's PolyTrans 2.2

by Christian Aubert

PolyTrans addresses a lot of 3D file translation problems that affect game developers. Anyone who has ever had to deal with file translation issues will benefit from this package. For this review, I tested PolyTrans release 2.2 on an HP Kayak XU-400 with a Matrox Millennium II 8MB board and an IBM Intellistation M Pro 6889 with Intergraph Intense3D Pro3400 graphics adapter. The package was very stable, and I never experienced any crashes. Clearly, Okino has spent countless hours tuning and refining this software package. A fast CPU will certainly help on large batch translations, but memory wasn't so much of a concern, as the data storage algorithms seem to be very memory-efficient. The only problem I ran into was with display flickering during animation playback. Okino customer support was quick to point out that animation playback works best in Shaded mode using hardware-accelerated OpenGL.

Real-time game developers including technical directors, technically minded 3D animators, and tools programmers will all appreciate PolyTrans's support for most standard professional 3D packages, along with MultiGen's OpenFlight, Nichimen's Game Exchange, and Cinegraphics's UView UV plug-in for LightWave. PolyTrans's OpenGL C code, DirectX, and VRML 1.0 and 2.0 export capabilities can all come in handy for quick prototyping of in-house tools or game engines. These capabilities will be of most interest to producers looking to save their teams some work by buying an off-the-shelf package instead of writing their own conversion utility.

Anyone with a minimum of exposure to 3D software will feel right at home with this package. The standard orthogonal and perspective views are available with a choice of bounding box, wireframe, or shaded geometry displayed. The interface is straightforward and fairly simple, considering the wealth of choices facing the user. A higher level of technical competence will be required to fully understand and appreciate the vast array of choices that one is presented with when dealing with 3D format conversions.



A wolf character from Behaviour Interactive's latest work in progress. This is the original model in LightWave.

PolyTrans features file import converters for the following formats: ACIS SAT, Apple 3D Metafile, 3D Studio R4, 3D Studio MAX 1.2 and 2.x, Fractal Design Detailer, .DXF, IGES 5.3, Imagine, LightWave, Nichimen Game Exchange, OpenFlight, Pro/Engineer, Softimage, Stereolithography, trueSpace 2.0/3.0, USGS DEM, VistaPro, and Alias|Wavefront (NURBS and Polygons). File export converters support import formats such as DirectX, Lightscape, Open GL C Code, POV Ray 2.0 and 3.0, Renderman RIB, RenderWare, and VRML 1.0 and 2.0. The ACIS, IGES, OpenFlight, Nichimen, and Softimage import modules are sold separately, each costing between $195 and $395.

PolyTrans does much more than simply convert polygonal geometry like older converters. Artists working on prerendered 3D art will like the easy transfer between their favorite packages. Support for triangulating NURBS and spline patch surfaces and tight surface definition transfers will definitely save lots of time. PolyTrans also does a good job of preserving surfaces attributes, lighting, and camera information. This package doesn't stop at geometry conversion either. Vertex welding, polygon normal flipping, and vertex normal computation operators are all available. You can also use PolyTrans to build physical models of in-game or prerendered geometry through a rapid prototyping technology known as stereolithography, whereby a laser draws a cross-section on the surface of a bath of photosensitive resin, partially curing the resin and thus producing a thin layer of solid material. By lowering the bath for each successive iteration of the process, you end up with a three-dimensional physical model (a nice touch when presenting your project to prospective customers, and always a nice extra in a trade show booth). All this would be enough to justify the price of admission, but PolyTrans does something that's relatively rare among data translation tools; namely, animation transfers.

You can literally take a LightWave scene and render it in 3D Studio MAX or Softimage with no tweaking. However, users will need to adapt their workflow with PolyTrans use in mind. I did a test in which I took one of our in-game characters from LightWave, converted it to 3D Studio MAX, and then back over to LightWave. PolyTrans left all the geometry, animation, surface, light-

Christian Aubert is a technical director with Behaviour Interactive, whose most recent title is Jersey Devil *for the PlayStation. He's been working with 3D animation for 11 years*

Excellent | Very Good | Average | Below Average | Poor

ing, and camera information intact. One of the features that contributes to this flexibility is PolyTrans' support for conversion of 2D bitmap formats, allowing textures to be converted to a 3D package's native format. Thus, when translating a Strata file with its .TIF textures over to Softimage 3D, for example, you'll have the option of converting the 2D textures to Softimage's native .PIC format.

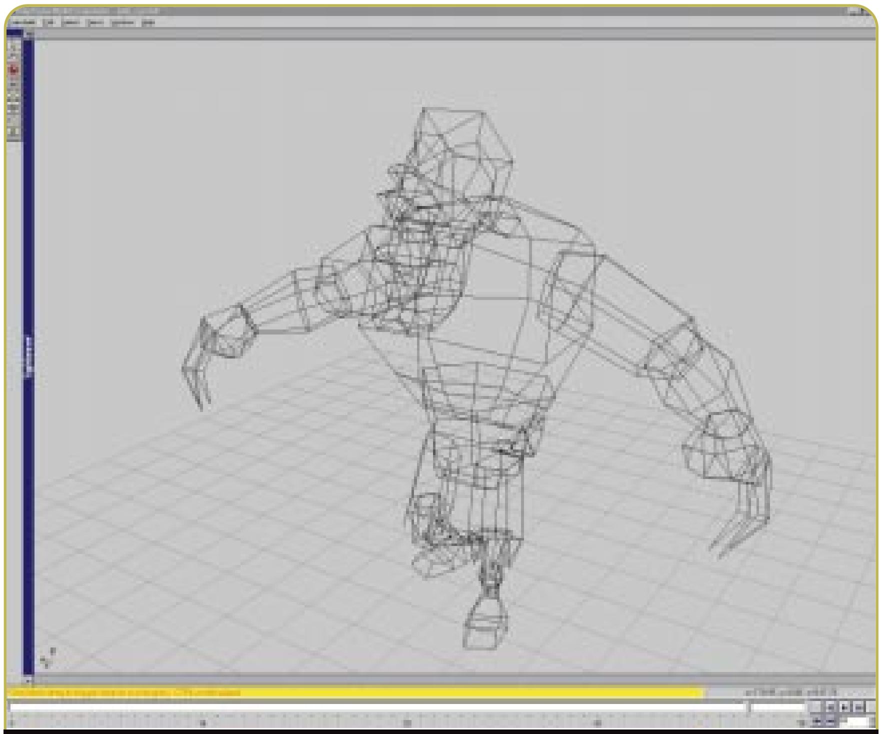
PolyTrans's main competitors are Lambsoft's Movetools and Viewpoint Interchange. Rhino from Robert McNeel & Associates does a good job of supporting polygonal and NURBS geometry, but it lacks support for batch translation and as such cannot be considered a competitor to PolyTrans.

While Movetools supports animation through per-frame keyframe sampling for most major 3D packages (with the notable exception of LightWave and Alias's Maya), it lacks PolyTrans's keyframe resampling technology and breadth of import/export formats. At a cost of $1,500 per module, a usable version of Movetools will set you back at least $3,000, or almost ten times the price of PolyTrans.

Viewpoint's Interchange 4.5 supports more file formats than PolyTrans, but it tends more towards a pure geometry translator. Support for surfaces, textures, animation, and hierarchical information is minimal or simply nonexistent.

If you ever come across file format conversion issues (and who hasn't?), PolyTrans will pay for itself in a matter of hours. If you need to do a lot of file format conversion, having multiple packages to chose from can be a good thing. As a matter of fact, our studio uses Rhino, Viewpoint Interchange, and PolyTrans/Nugraf.

Still, some of the liberties PolyTrans must take to allow conversion of very different data formats might cause problems for real-time game developers. Exporting a hierarchy to Softimage will look great, but the pivot points will all be in the world center. Users have brought this deficiency to the attention of Okino's technical support, and the company is trying to find a workable solution to the problem. Okino has a very good reputation for working closely with customers and quickly implementing needed features, often within hours or days if the problem is critical.


The same wolf character model, in PolyTrans.

Depending on workflow, this package might be more appropriate to rendering applications than to real-time, where data integrity has priority over visual quality. Considering the fact that it costs one quarter to one twentieth the price of the 3D packages it helps interface with, the question of whether to buy PolyTrans is moot. The only reason I wouldn't recommend buying this piece of software is because you can get its big brother Nugraf for a mere $100 more. Nugraf offers a high-speed ray-tracing/Z-scanline photorealistic renderer that can handle huge datasets, very complete interactive material, and 2D/3D texture editing support, all with a much more extensive graphical user interface. ■

PolyTrans 2.2 ★★★★½

Company: Okino Computer Graphics Inc.
Mississauga, Ontario, Canada
(888) 336-5466
http://www.okino.com

Price: $395 Windows 95/NT, $495 Irix 5.2

System Requirements: Windows 95/NT – Minimum 386, 6MB RAM, 30MB hard-disk space. Irix 5.2 – Minimum R4x00, 32MB RAM, 30MB hard-disk space.

Competitors: Viewpoint Interchange and Lambsoft Movetools

Pros:

1. Very powerful and robust animation resampling and keyframe reduction algorithms.
2. Very attractive base price.
3. Solid geometry conversion, including support for NURBS and spline patch surfaces.

Cons:

1. ACIS, IGES 5.3, Softimage and Openflight add-on modules can end up costing as much as the base package.
2. Some features might not work as expected, and thus may require Okino's collaboration for customization.
3. High level of technical knowledge required.

MultiGen's Creator 2.1

by Jeff Abouaf

Creating 3D game objects traditionally involves a lot of back-and-forth between the modeler and the programmer. Usually, an artist takes a stab at creating a low-polygon model, then hands it off to a programmer. The programmer evaluates the model's behavior in the game engine, and then sends the model back to the artist with comments. Creator, from MultiGen-Paradigm, is a design tool that helps artists create better models from the start.

Creator is a highly specialized tool that helps 3D modelers create efficient polygonal objects for real-time 3D applications. It complements traditional animation development environments such as 3D Studio MAX. Creator lets you build objects a few faces at a time; as you build faces and objects, Creator constructs an underlying database for the model. This database is then sent through your game's engine at run time. Because Creator lets the artist construct an efficient hierarchical database from the beginning, it can reduce the number of hand-offs between artists and programmers.

CREATOR'S ROOTS. When MultiGen-Paradigm Inc. released Creator 2.0 for Windows NT in December 1997, it brought to Windows NT a specialized set of real-time authoring tools formerly available only on Irix (as MultiGen II Pro). These capabilities had been used to create military and flight simulators, and more recently, real-time 3D urban simulations. In effect, Creator 2.0 was MultiGen II Pro (their flagship product) redesigned, less the road and advanced terrain tools, and developed for Windows NT (later ported to the SGI).

This past October, MultiGen released Creator 2.1, which incorporated all of the advanced features of MultiGen II Pro. This high-end tool set occupies a unique niche in the market: most game developers use Creator for its hierarchical database editor, terrain tools, LOD tools, and Road Pro options, while continuing to use their other 3D content creation tools for character animation, organic modeling (Creator doesn't support patches or NURBs), and non–real-time animation segments.

Creator uses a broad and extensible 3D file format called OpenFlight, which was invented by MultiGen. Whereas VRML is a minimal specification optimized for real-time 3D (RT3D) over a network, OpenFlight was designed to meet the demanding requirements of dedicated, stand-alone, high-end flight simulators. The format accommodates the development of new database nodes, or "beads." The Road Pro options represent such a bead, in which all road attributes can be written to a single node. Creator 2.1 ships with its SDK and a new Plug-in Wizard to simplify plug-in development. Third parties are extending Creator's capabilities by developing plug-ins, ranging from new file translators to real-time viewers within the product.

THE MODELING ENVIRONMENT. Creator offers a split-screen layout that displays both the scene and database hierarchy (Figure 1). The database can be reordered simply by dragging boxes around. The boxes are represented top to bottom by group(s), object(s), faces; and left to right according to the order in which they proceed through the run-time engine. You can specify the drawing order for faces, choosing from fixed list (which determines the order in which faces are painted), BSP, or Z-buffer. The last option doesn't require

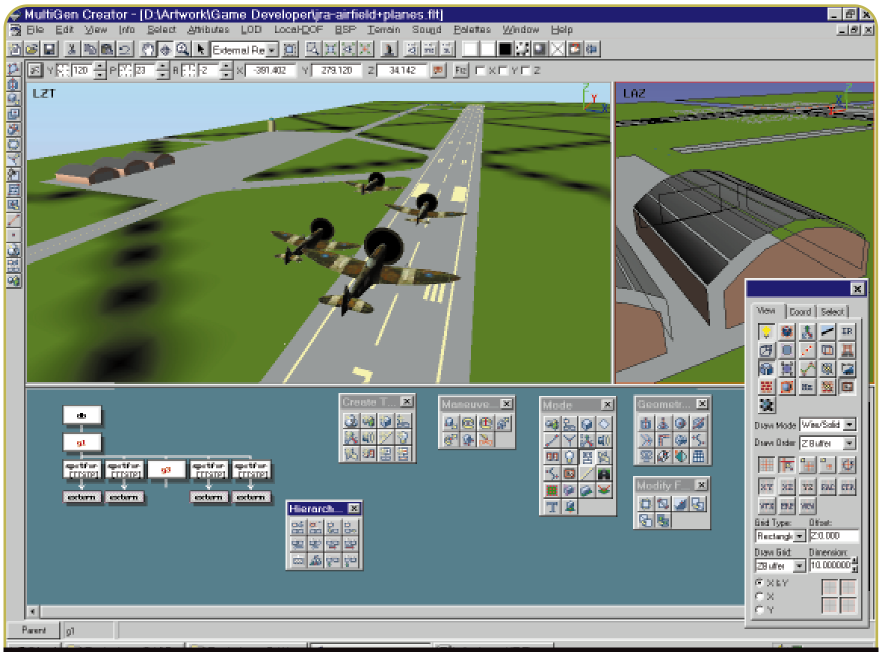


FIGURE 1. *Creator's layout enables working interactively on the model, while adjusting the underlying hierarchical database. Double-clicking on either the object in the scene or the box in the database brings up a detailed attributes dialog box (not shown).*

Jeff Abouaf is the principle at Ogle cg/fa, a multimedia and fine art design studio in Mill Valley, Calif. He can be reached at jabouaf@ogle.com.

faces to be processed in a particular order, but does require more powerful 3D graphics acceleration.

Creator doesn't contain a run-time engine. If you don't have one, the Performer-based "Per-Fly" is available on Irix. On Windows NT, you can use Open GVS, RealiView (Datapath's Realimation viewer program), or SimStudio (NDimension Simulations Inc.'s viewer), all of which are included on the Creator Partner's CD that's included with the product. Most Creator users have a proprietary run time in mind: for example, Atari's Creator-built environments in San Francisco Rush ran on Atari's own engine. Creator also has special run-time support for the Nintendo and PlayStation. Specifically, the Creator NIFF option allows N64 game developers to take full advantage of Creator's modeling and database features by providing an interface to the capabilities provided in the Nintendo development kit. Also, an optional .HMD conversion tool provides support for Sony's Hierarchical Modeling Data (HMD), their high-level graphics processing framework available under the PlayStation development environment.

Creator's roots in visual simulation are reflected in a very specialized tool set. The only modeling techniques available involve extrusion, lathing, and lofting. Creator doesn't support spline modeling, patch modeling, and NURBS modeling (as in, curve-based modeling technologies). Creating a simple rectangle requires you to position a grid (the Tracking Plane), then first and second diagonal points. Building off of that surface requires repositioning the Tracking Plane. At first, this feels constraining, especially compared to free-form drawing products such as 3D Studio MAX, Softimage, and others. But in practice, it forces you to consider the need and purpose of each face created, whether the model is better optimized by slicing, combining, or reshaping faces, and how that face fits within the database.

For example, I recently generated a low-polygon aircraft model with a "traditional" 3D modeling package. With smoothing, the model required just over 600 faces; a similar model built in Creator resulted in about half the faces, due to the manner in which it

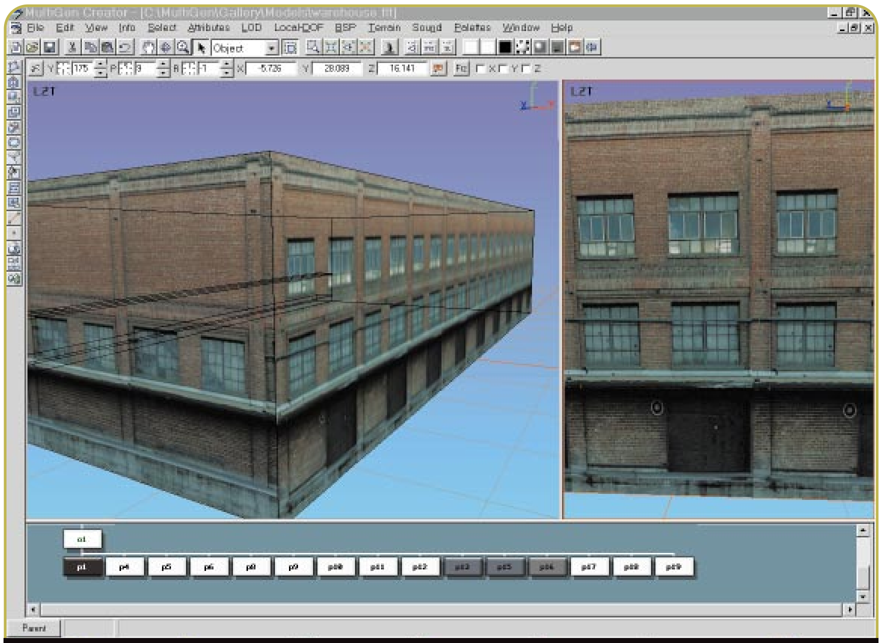

MultiGen Creator - [C:\MultiGen\Gallery\Models\warehouse.flt]
File Edit View Info Select Attributes LOD LocalDOF BSP Terrain Sound Palettes Window Help
Object
L2T
L2T
o1
p1 p4 p5 p6 p8 p9 p10 p11 p12 p13 p15 p16 p17 p18 p19
Parent

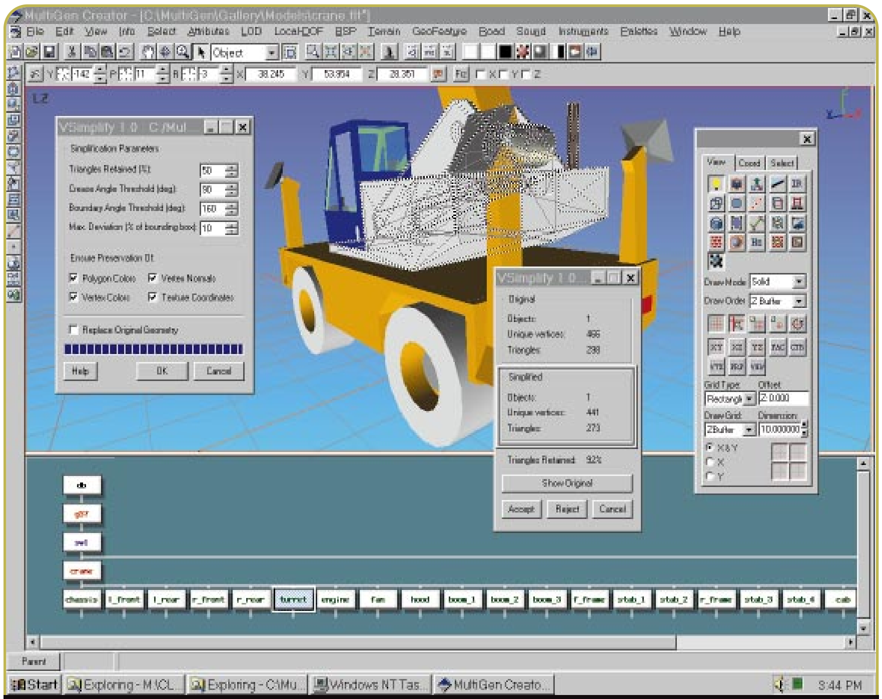
FIGURE 2. *Creator's materials and texturing capabilities support realistic results. This warehouse model contains 15 faces, 60 vertices, and uses a 768K texture map.*

was constructed and the ease of editing faces. Creator has an extensive tool set for working at face, edge, and vertex levels, and additional tools for checking for coplanar polygons or concave polygons, which are illegal in most run-time engines.

Creator's Materials Palette gives control over the material's shininess, opacity (alpha), diffuse, ambient, specular, and emissive attributes. The Texture Palette lets you store groups of bitmap textures for mapping onto objects, but the materials and textures don't support bump maps. Still, with vertex coloring, the materials capabilities, and good texture maps, Creator can achieve remarkable results. Figure 2 is a screen shot of a warehouse model containing 15 faces, 30 triangles, and a 768K RGB texture map.

Unlike 3D Studio MAX, Softimage, and their competitors, Creator is not an animation environment; the reason being is that most RT3D animation occurs at run time. However, you can build minimal animation into a model using the local degrees of freedom (DOF) tools, or by creating a series of position poses that can be cycled in a flip-book. The former is useful for rotating the wheels of a vehicle; the latter works well for making a character's arm move or setting up a walk cycle. In these examples, Creator could take care of secondary movement, while your run-time engine controls translations in the scene.

Creator includes a powerful level of detail (LOD) tool set for swapping lower-polygon mesh copies, which is useful for populating scenes with distant or background objects where using a higher-detail model is a waste of resources. You create a duplicate model (or series of models) of lesser detail, and Creator handles the task of switching LODs based on the object's distance from the viewpoint (which can be set manually or automatically). More important, Creator can smooth transitions by morphing vertices between LODs, producing the effect of a progressive mesh or intelligent polygon decimation. But it does so without introducing incompatible geometries or file formats. In the aircraft model in Figure 1, the highest LOD was about 300 faces, while the lowest was just 3. Virtue 3D has developed VSimplify, an intelligent polygon reduction plug-in for Creator. VSimplify works by triangulating all selected polygons on a model, and reducing that number. In the polygon-reduction process, you can control the percentage of triangles


MultiGen Creator - [C:\MultiGen\Gallery\Models\crane.flt*]
File Edit View Info Select Attributes LOD LocalDOF BSP Terrain GeoFeature Road Sound Instruments Palettes Window Help
Object
View Coord Select
Help OK Cancel
Show Original
Accept Reject Cancel
Start
3:44 PM

VSimplify 1.0 C:/Mul

Simplification Parameters

Triangles Retained (%):	50
Crease Angle Threshold (deg):	90
Boundary Angle Threshold (deg):	160
Max. Deviation (% of bounding box):	10

Ensure Preservation Of:

- ✓ Polygon Colors
- ✓ Vertex Normals
- ✓ Vertex Colors
- ✓ Texture Coordinates
- ☐ Replace Original Geometry

VSimplify 1.0

Original	
Objects:	1
Unique vertices:	466
Triangles:	298
Simplified	
Objects:	1
Unique vertices:	441
Triangles:	273
Triangles Retained:	92%

FIGURE 3. *Applying Creator's VSimplify plug-in to a low-polygon mesh reduced the triangle count an additional amount. The plug-in works automatically, allows you to preserve the original state, and toggle between the before and after versions.*

to retain, the crease-angle threshold, the object-boundary threshold, and the deviation from the original bounding-box size. Figure 3 shows VSimplify applied to a crane model. From my point of view, this plug-in fills a noticeable gap in the LOD tool set.

PRO OPTIONS TOOLS. The two most important additions to Creator in the 2.1 release are Pro Options Tools: the advanced terrain-following tools and the Road Pro package. These are advanced tools designed for simulations based on actual data. Creator can import USGS or DTED files and convert them to its own .DED file format, generating scene terrain to double-precision accuracy. The new advanced terrain capabilities enable importing satellite image data, which can be mapped onto the terrain. From this image data, Creator can populate the scene with trees, vegetation, and other features, which, depending on hardware, can be generated on-the-fly. If the geographical database is particularly large (for instance, the U.S. Eastern Seaboard), Creator can slice the terrain into tiled sections for loading on-the-fly during the simulation.

The Road Pro options consist of three basic operations: road construction, road tessellation, and scenario data. Road construction interactively defines the type of road, beginning and end points, curve, banking, and slope. Road tessellation defines and applies the road attributes, such as light poles and signs, as well as road LODs. The scenario data, such as lane and centerline data, are additional attributes that can be accumulated for driving simulations. The detail and accuracy of the roads generated and the LOD capabilities reflect MultiGen's interests in urban simulation. For game developers, you can drive along the road and generate instrumentation for the driving simulation.

Creator fits into the game development workflow in two respects: it works as a content/hierarchical database creation environment in the first place, or as an intermediary tool to be used between a cinematic 3D authoring tool (such as 3D Studio MAX, Softimage, and so on) and the run-time engine. While very powerful in its terrain- and environment-building capability, Creator is not strong in character modeling/animation, especially when compared to Maya, Kinetix's Character Studio 2.0, Softimage, and Nichimen. In my tests, Creator imported 3D Studio MAX models well, preserving the database hierarchy. However, I found the time it took to edit the polygon count from the 3D Studio model was better spent rebuilding the model from scratch in Creator using the 3D Studio model as a template. My hope is that VSimplify and other polygon reduction tools on other packages will alleviate this problem. ■

Creator 2.1: ★★★★½

Company: MultiGen-Paradigm Inc.
San Jose, Calif.
(408) 261-4100
http://www.multigen.com

Price: $9,500. The Road Pro option is an additional $12,000.

System Requirements: Windows NT 4.0, 133 Mhz Pentium P5, 32MB RAM, 1GB hard drive space, and a 3D graphics accelerator.

Pros:

1. Creator is very useful for building CAD-accurate RT3D environments, from scratch or using USGS data.
2. Its extensive tool set for generating and editing low-polygon models and LODs is impressive.
3. It allows complete visual editing of a hierarchical database (no code), bringing the artist close to the programmer in the production pipeline.

Cons:

1. While a visual authoring environment, Creator is not as interactive as it may become. Fundamentally, this is a tool for creating and editing a database one piece at a time.
2. The interface is an improvement over MultiGen II Pro on Irix, but for the untrained, the relationship between tool palettes, menu items, and step sequences result in a significant learning curve.
3. The price point may limit size of the user base.

Crashing into the New Year

It's hard to believe that it's already 1999. The last year moved at an incredible pace. It was a year with amazing advances in the visual quality of games. The predictions that 3D hardware would become a major force in the industry have come true.

Consumers can now buy cards for under $100 that deliver 3D graphics performance that would have cost thousands only a few years ago. This added processing power leaves game developers more and more time to dedicate to exploring other areas in computer simulation.

I'm continually amazed that learning a simple trick or technique can open the door to so many different effects and applications. In past columns, I've discussed how techniques such as the dot product and cross product can be used in applications such as animation and inverse kinematics. This month, I'm going to apply these same, well-used methods to the problem of collision detection. Collision detection is a huge issue in graphics simulation. In fact, it's an active area of research, so SIGGRAPH and professional journals are a great source of information.

Let me start off by looking at some common problems that can be important to a variety of game applications. These routines, though fairly simple, are very handy to have in your library. The first issue is how to determine whether a point is inside an arbitrary area. Detecting whether a point is inside a convex polygon can be determined very easily. Figure 1 shows a point inside a simple four-sided polygon. Our first step is to create vectors for each of the polygon edges and a vector from the test point to the first vertex of each edge. As you may recall from previous columns, the dot product of two vectors defines the cosine of the angle between those vectors. If the dot product for each of the edges is positive, all the angles are less than 90 degrees and the point is inside the polygon.

That rule is pretty useful for some things. However, it only works when the boundary that you're checking is convex. Many spaces that we're interested in are actually concave (Figure 2). This polygon looks like a character in a DUKE NUKEM level. And in fact, DUKE is a pretty good application for this kind of test. Each "sector" of a DUKE level is a polygonal boundary defining a region with a specific floor and ceiling height. Knowing whether I'm inside or outside of a particular sector is important information. Unfortunately, the aforementioned dot product test won't work on these concave polygons. I could divide this region into smaller convex polygons, but that wouldn't be very efficient. Luckily, this problem is the classic "point in polygon" test that's commonly described in computational geometry books. There are many approaches to solving this problem, but I want to look at just two of them.

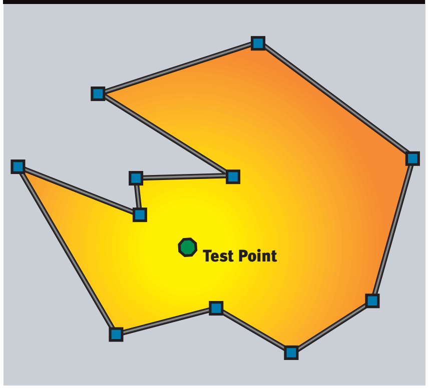


FIGURE 2. *Inside a concave polygon.*

Jeff made a resolution this year to shrink his own bounding box and to spend more time away from the computer. Show him how futile this is by mailing him at jeffl@darwin3d.com.

Here We Go Round the Vertex List

One method for determining if the test point is inside the concave polygon comes from the idea that a circle is 360 degrees. Calculate the angle between each vertex and the test point (at the test point itself) and then add up all the angles. If the total is equal to 360, then you are inside. You can see

FIGURE 1. *Inside a convex polygon.*

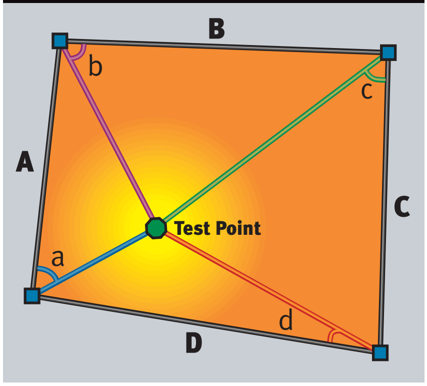
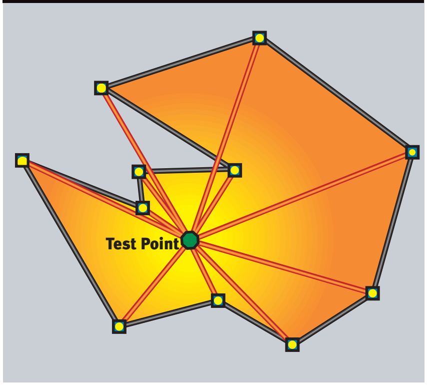


FIGURE 3. *Angles around the test point.*



what this looks like in Figure 3.

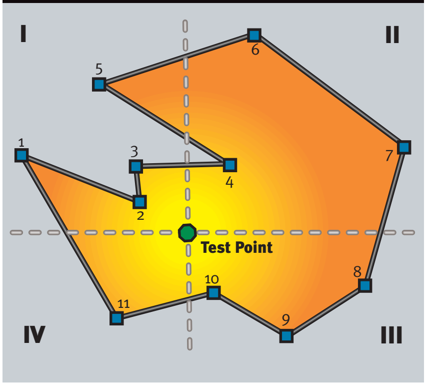
This actually works very well, however, it is not very efficient. Calculating each angle requires a dot product and an arccosine operation. Those will add up quickly.

A better strategy is to divide the polygon into quadrants centered on the test point, as in Figure 4. Start at the first vertex in the polygon and set a counter to 0. Anytime an edge crosses from one quadrant to the next, add one to the counter if it crosses clockwise around the test point and subtract one if it crosses counter-clockwise. If the edge crosses diagonally across two quadrants, you need to determine which side of the test point it crossed, and then either add or subtract 2. Try it yourself on Figure 4. Start at vertex 1. Add 1 when edge 3-4 crosses from quadrant I to II, and subtract it again with edge 4-5. When you reach the last edge (11-1), you should have 4. When using the routine, if the counter is equal to 4 or –4, the test point is inside the polygon. You can see the code for this routine in Listing 1.

Don't Cross that Line

The quadrant method is pretty efficient. However, there's a completely different approach. An interesting feature of this problem can be found if you draw a line from the test point to a point definitely outside the polygon. Now count how many polygon edges crossed that line. If that number is odd, the point is inside the polygon. If the number of edge crossings is even, the point is on the outside. Try it out.

FIGURE 4. *Dividing the polygon into quadrants.*



I saw a pretty fast way to implement this in *Graphic Gems IV*. This method projects a line from the hit position along the x axis. Only testing line segments that are on either side of this position lets you avoid some calculations. Segments that could cross this line require an x intercept calculation to be sure. However, this can be simplified to eliminate a divide because of the unique needs of the test. The code for this routine is in Listing 2. Whether or not this method is faster than the quadrant method depends

LISTING 1. *The quadrant approach to the bounding box test.*

```
// FIGURE OUT WHICH QUADRANT THE VERTEX IS RELATIVE TO THE HIT POINT
#define WHICH_QUAD(vertex, hitPos) \
  ( (vertex.x > hitPos->x) ? ((vertex.y > hitPos->y) ? 1 : 4) : ( (vertex.y > hitPos->y) ? 2
: 3) )
// GET THE X INTERCEPT OF THE LINE FROM THE CURRENT VERTEX TO THE NEXT
#define X_INTERCEPT(point1, point2,  hitY) \
  (point2.x - ( ((point2.y - hitY) * (point1.x - point2.x)) / (point1.y - point2.y) ) )

///////////////////////////////////////////////////////////////////////////////
// Procedure: PointInPoly (SUM OF ANGLES CROSSING VERSION)
// Purpose:   Check if a point is inside a polygon
// Returns:   TRUE if Point is inside polygon, else FALSE
///////////////////////////////////////////////////////////////////////////////
BOOL CFateView::PointInPoly(tSector *sector, tPoint2D *hitPos)
{
/// Local Variables ///////////////////////////////////////////////////////////
    short  edge, first, next;
    short  quad, next_quad, delta, total;
///////////////////////////////////////////////////////////////////////////////
    edge = first = sector->edge;
    quad = WHICH_QUAD(m_edgelist[edge].pos, hitPos);
    total = 0;    // COUNT OF ABSOLUTE SECTORS CROSSED
    /* LOOP THROUGH THE VERTICES IN A SECTOR */
    do {
        next = m_edgelist[edge].nextedge;
        next_quad = WHICH_QUAD(m_edgelist[next].pos, hitPos);
        delta = next_quad - quad;      // HOW MANY QUADS HAVE I MOVED
        // SPECIAL CASES TO HANDLE CROSSINGS OF MORE THEN ONE QUAD
        switch (delta) {
        case  2: // IF WE CROSSED THE MIDDLE, FIGURE OUT IF IT WAS CLOCKWISE OR COUNTER
        case -2: // US THE X POSITION AT THE HIT POINT TO DETERMINE WHICH WAY AROUND
        if (X_INTERCEPT(m_edgelist[edge].pos, m_edgelist[next].pos, hitPos->y) > hitPos->x)
            delta =  - (delta);
                break;
        case  3:         // MOVING 3 QUADS IS LIKE MOVING BACK 1
            delta = -1;
                break;
        case -3:         // MOVING BACK 3 IS LIKE MOVING FORWARD 1
            delta =    1;
                break;
        }
    /* ADD IN THE DELTA */
    total += delta;
    quad = next_quad;    // RESET FOR NEXT STEP
    edge = next;
    } while (edge != first);

    /* AFTER ALL IS DONE IF THE TOTAL IS 4 THEN WE ARE INSIDE */
    if ((total == +4) || (total == -4)) return TRUE; else return FALSE;
}
```

greatly on the polygon being testing. The routines are so easy to implement that you should try both in your application if speed is a real issue.

Standing at Arm's Length

The above routines are enough to let your player navigate around in a DOOM-style level. You would just need to make sure that the player is always inside a sector. If the player leaves one sector and does not enter any other, a "collision" has happened. This works very well. However, using the inside polygon test for collision by itself has a drawback. The player can get very close to the wall of a sector and still be considered "inside." Logically, this works fine. However, in a 3D rendered game engine, being too close to a wall is a bad thing. Textures will look blocky, they can distort badly, and walls may clip out.

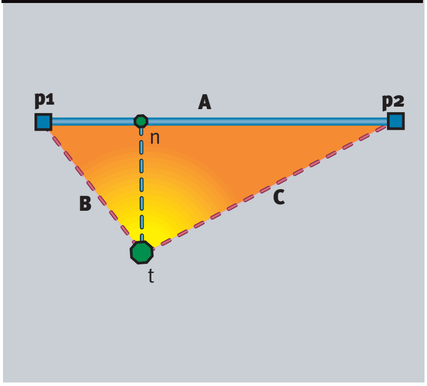
What you really want to do is keep the walls at "arm's length" from the player. You can simply make the logical collision walls closer in than the visual walls; however, this can lead to other problems. So how do I keep the character away from the wall? Turn once again to our dear old friend, the dot product. Take a look at Figure 5.

What I want to know is, how far away is the test point, t, from line segment A. An easy solution would be to find the nearest point, n, to the test point on the line segment and measure the distance to it. First, I create a vector, B, from the test point, t, to vertex p1. I can dot this vector with the line segment A. This will give me the cosine of the interior angle. If this angle is 90 degrees or greater, the nearest point is the vertex itself and I'm done. But let's say that the dot product gives me 0.7, or the cosine of about 45 degrees. I will then do the same thing on the other side. I create a vector C and dot it with the segment A. If it had returned an angle greater than or equal to 90 degrees, point p2 would be the closest and I would be done again. In this case, the dot product returns 0.75, or the cosine of about 40 degrees. Now that I have the two dot products, a linear ratio will solve the problem.

$$n = p_1 + \frac{(p_2 - p_1) * (B \bullet A)}{(B \bullet A) + (C \bullet A)}$$

You can see the code that determines the nearest point on a line segment to an input point in Listing 3. The squared distance from t to n can be used to make sure the player cannot get too close to the wall. When I combine this with the inside-polygon tests, I have the pieces I need to create a DOOM-style collision model. In these days of QUAKE II and UNREAL, it may seem a bit retro to talk about DOOM-style collision detection. However, the ability to build simple collision boundaries that you can use and modify in real-time is a very attractive feature. Rules in game development are meant to be broken. Just because these days you are displaying a world made of 3D polygons, your collision boundaries don't have to be 3D polygons. Many of the environments we wish to interact with have boundaries that can easily be defined as 2D concave-polygonal-line-segments. Sometimes the best results can be achieved with simple solutions. If you didn't have these routines already in your math library, add them and you will be surprised by how often you use them.

FIGURE 5. *Checking the distance.*



LISTING 2. *The x intercept calculation.*

```
///////////////////////////////////////////////////////////////////////////////
// Procedure: PointInPoly (EDGE CROSSING VERSION)
// Purpose:      Check if a point is inside a polygon
// Returns:      TRUE if Point is inside polygon, else FALSE
///////////////////////////////////////////////////////////////////////////////
BOOL CFateView::PointInPoly(tSector *sector, tPoint2D *hitPos)
{
/// Local Variables ///////////////////////////////////////////////////////////
   short  edge, first, next;
   tPoint2D *pnt1,*pnt2;
   BOOL   inside = FALSE;      // INITIAL TEST CONDITION
   BOOL  flag1,flag2;
///////////////////////////////////////////////////////////////////////////////
   edge = first = sector->edge;      // SET UP INITIAL CONDITIONS
   pnt1 = &m_edgelist[edge].pos;
   flag1 = ( hitPos->y >= pnt1->y ) ; // IS THE FIRST VERTEX OVER OR UNDER THE LINE
   /* LOOP THROUGH THE VERTICES IN A SECTOR */
   do {
      next = m_edgelist[edge].nextedge; // CHECK THE NEXT VERTEX
      pnt2 = &m_edgelist[next].pos;
         flag2 = ( hitPos->y >= pnt2->y );  // IS IT OVER OR UNDER

      if (flag1 != flag2)     // MAKE SURE THE EDGE ACTUALLY CROSSES THE TEST X AXIS
      {
   // CALCULATE WHETHER THE SEGMENT ACTUALLY CROSSES THE X TEST AXIS
   // A TRICK FROM GRAPHIC GEMS IV TO GET RID OF THE X INTERCEPT DIVIDE
   if (((pnt2->y - hitPos->y) * (pnt1->x - pnt2->x) >=
      (pnt2->x - hitPos->x) * (pnt1->y - pnt2->y)) == flag2 )
      inside = !inside;// IF IT CROSSES TOGGLE THE INSIDE FLAG (ODD IS IN, EVEN OUT)
      }
   pnt1 = pnt2;  // RESET FOR NEXT STEP
   edge = next;
   flag1 = flag2;
   } while (edge != first);
   return inside;
}
```

Colliding in the Third Dimension

Running around a maze is one thing, but that's just the beginning to the collision detection story. Consider the problem of determining if two objects have collided. Two of the most common approaches to this problem are bounding boxes and bounding spheres. For a bounding sphere, find the point furthest away from the center of the object. The distance of the point from the center defines the radius of the bounding sphere. You can see a bounding sphere around an object in Figure 6. Now imagine that the object and circle around it are actually 3D. As you can see, although the entire object is inside the sphere, it isn't a snug fit. You can see how easy it would be to have an object that would hit the bounding sphere yet miss the object completely.

So why use a bounding sphere? Well, testing to see if a collision has occurred is really fast. If you measure the distance between two objects, and the distance is less than the radius of either bounding sphere, then there is a collision. This is a very quick test, so it's easy to see why many games use spheres as at least a first line of defense. But if you need to determine exactly where two objects made contact, this won't be enough.

Axis-aligned bounding boxes, or bounding boxes for short, are another simple method for collision detection. The box is called axis-aligned because the sides of the box are parallel to the principle world x, y, and z axes. This reduces the check for collision to a simple minimum-maximum test. You create the box by determining the minimum and maximum extents in each dimension. The collision test then consists of:

```
IF (  (point.x >= box.minX
and point.x <= box.maxX)
and (point.y >= box.minY
and point.y <= box.maxY)
and (point.z >= box.minZ
and point.z <= box.maxZ)
 ) then a collision occurred.
```

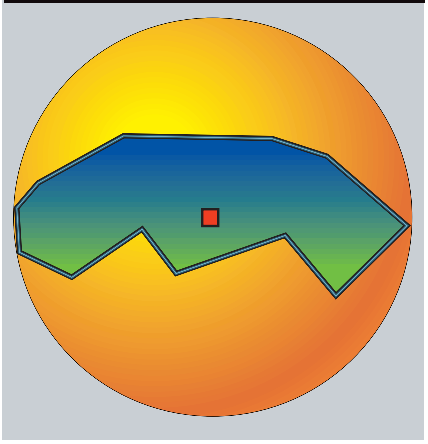
Bounding boxes are a simple, fast way to check for rough collisions. However, like the bounding spheres, the fit is not necessarily very accurate. They're generally used as a first test to check if further investigation is needed. You can improve the fit by maintaining a hierarchy of smaller bounding boxes or spheres that are tested after the initial collision is determined. For many games, such as 3D fighting games which require fairly detailed collision, this is enough for realism. If you need more detailed collision information, you need to look elsewhere.

Other methods such as oriented bounding boxes (OBB), where the bounding box is allowed to rotate to an arbitrary orientation, will allow for a tighter fit than either above method. However, even OBBs do not provide information on the exact point of collision on an arbitrary mesh unless the object happens to be a box.

Getting to the Point

If you really need to know which point of an object has collided with another — say for your realistic physics simulation — you have some work in front of you. All the other techniques are good first steps, and serve to filter out unneeded calculations. I'm going to start out in 2D again to make things easy. Let me begin by considering only convex objects. Remember that convex objects are polygon meshes that con-

FIGURE 6. *A bounding sphere.*



LISTING 3. *Finding the nearest point on a line segment.*

```
///////////////////////////////////////////////////////////////////////////////
// Procedure: GetNearestPoint
// Purpose:   Find the nearest point on a line segment
// Arguments: Two endpoints to a line segment a and b,
//            and a test point c
// Returns:   Sets the nearest point on the segment in nearest
///////////////////////////////////////////////////////////////////////////////
void CFateView::GetNearestPoint(tPoint2D *a,tPoint2D *b,tPoint2D *c,tPoint2D *nearest)
{
/// Local Variables ///////////////////////////////////////////////////////////
   long dot_ta,dot_tb;
///////////////////////////////////////////////////////////////////////////////
   // SEE IF a IS THE NEAREST POINT - ANGLE IS OBTUSE
   dot_ta = (c->x - a->x)*(b->x - a->x) + (c->y - a->y)*(b->y - a->y);
   if (dot_ta <= 0) // IT IS OFF THE a VERTEX
      {
      nearest->x = a->x;
      nearest->y = a->y;
      return;
      }
   dot_tb = (c->x - b->x)*(a->x - b->x) + (c->y - b->y)*(a->y - b->y);
   // SEE IF b IS THE NEAREST POINT - ANGLE IS OBTUSE
   if (dot_tb <= 0)
      {
      nearest->x = b->x;
      nearest->y = b->y;
      return;
      }
   // FIND THE REAL NEAREST POINT ON THE LINE SEGMENT - BASED ON RATIO
   nearest->x = a->x + ((b->x - a->x) * dot_ta)/(dot_ta + dot_tb);
   nearest->y = a->y + ((b->y - a->y) * dot_ta)/(dot_ta + dot_tb);
}
```

tain no interior angles greater than 180 degrees. Figure 7 outlines the problem.

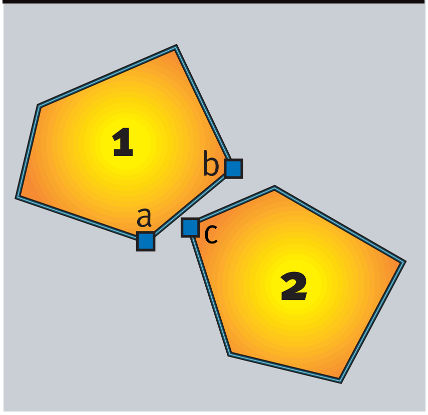
I am interested in deciding whether polygon 1 is colliding with polygon 2. I could use my point-in-polygon test from earlier, and test every point in each polygon and see if it's in the other. That wouldn't be very efficient though, and in 3D it would be even less reasonable. What I really want to find is a single feature that makes it impossible for polygon 2 to be inside polygon 1. It turns out that if I can find a line that separates the two polygons, then they cannot be colliding. To make it easier, I will use the edges of each polygon as a test line. If all the vertices of polygon 2 are on the other side of an edge in polygon 1, they aren't colliding.

You will recall from the convex point-in-polygon test that I used the dot product to determine if a point was inside an edge. I can use the same test to see if a point is outside. In other words, **`if vectorba dot vectorbc < 0, then point c is outside of polygon 1.`**

That dot product operation sure comes in handy. In 3D, you would be dotting the face normal with the test point, but it works out similarly. The first time you test to find this separating edge, you need to test all edges in each polygon against all the vertices in the opposite one. However, once you have found a separating edge, you can store this information so that edge is the first one you will test the next time you try. This caching of collision points can speed up testing quite a bit, and I highly recommend it.

That is all the time I have for this month. Next month, I will examine what exactly is required for detecting the collision point in 3D. I will also discuss what you can do with this information once you have it, and work out some cool samples to demonstrate it. ■

FIGURE 7. *Two convex polygons.*



FOR FURTHER INFO

• O'Rourke, Joseph. *Computation Geometry in C.* Cambridge University Press, 1993. A very good discussion of point-in-polygon strategies as well as path finding and convex hull operations (hint: may be handy).

• Heckbert, Paul S., Editor. *Graphic Gems IV.* Academic Press, 1994. Inside polygon strategies and routines.

• Baraff, David, and Andrew Witkin. "Physically Based Modeling," SIGGRAPH Course Notes, July, 1998, pp. D32 – D40. Collision detection and response methods.

Back

Playing God: World Creation in Real-Time 3D

Until very recently, creating a fully immersive RT3D environment used to be an exercise in minimalism. Software-only rendering engines and limited texture budgets set a bounding box that often resulted in a pixelated and blocky virtual world.

With today's hardware, the bounding box around our capabilities has expanded exponentially, and the digital vistas we create are beginning to look more and more like what we see outside our office windows. However, with the added functionality comes added responsibility and expectation; game players are becoming savvy to the RT3D scene, and have come to expect a level of quality that doesn't always come easily. Successfully creating a living, breathing environment that contributes to game play instead of compromising it can be a daunting task, especially with this industry's stringent deadlines. To top it off, the endless tweaking and reworking that we artists are all prone to while trying to "get it just right" can lead to resource bottlenecks and missed deadlines, which in turn create undue friction between art and design, and undermine the creative process for the entire team.

Part 1: Working with Game-Play—Critical Objects

Over the next two months, we'll examine how to successfully generate the art resources for a RT3D title, from the initial design document to the final gold-master CD. We'll look at planning strategies and artistic techniques, and also discuss some common pitfalls. The goal will be to look at the problem from both the standpoint of a beginning team tackling its first project, as well as from the point of view of a seasoned production staff ramping up in the first quarter of the year.

Step 1: Creating the Art Bible

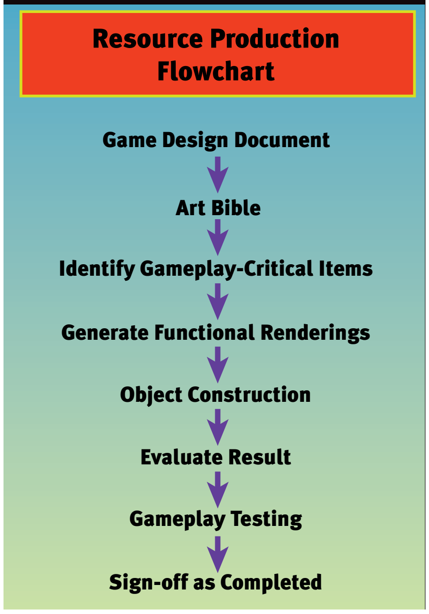
We'll begin with the assumption that your team has progressed through the initial design and planning stage of the project. You have generated a game design document that reflects the contributions made by the respective team leads. The basic game play mechanic, the artistic look-and-feel, and the engine and tools parameters have all been defined. Most importantly for this discussion, the section of the game design document pertaining to the look-and-feel of the game, the art bible, should be complete.

The art bible will serve as the governing reference document for the art team throughout the project. From an artistic standpoint, creating a solid art bible is one of the most important tasks that an art lead will face; it will focus the creative efforts of the art team and remove any ambiguity as to the artistic vision for the game.

What should be included in the art bible? As a general rule, most of the initial conceptual work for the game should be included. Pencil renderings, photographs, and computerized mock-ups should be included, all with written descriptions highlighting the pertinent features of each image. These should be of sufficient detail and depth to provide the entire art team with a clear idea of what the art direction will be; character and creature designs, architectural types, landscapes and scenery, and lighting styles should all be identified. Enough information should be available and presented in such a manner that a new artist coming on board can get a clear indication of the art direction of the game simply by reading the art bible.

This is not to say that the vision is immutable and not subject to change. Quite the contrary, because another property of a good art bible is that it remains liquid and alive throughout the development process (time permit-

FIGURE 1. *Resource production flowchart.*



Mel has worked in the games industry for several years, with past experience at EIDOS and Zombie. Currently, he is working as the art lead on DRAKAN *(http://www.surreal.com). Mel can be reached via e-mail at mel@surreal.com.*

ting). Having a benchmark against which to measure new ideas is the basis of any good iterative process. A solid vision early on is what allows the process to be fluid. Alternatively, a weak vision at the start can lead to indecisiveness regarding later courses of action. This can be a fatal flaw in the production cycle because a true, coherent vision of the game will probably never be reached.

Having a benchmark against which to measure new ideas is the basis of any good iterative process.

Step 2: Identify Game-Play–Critical Items

Now that you've outlined the game on paper, it's time to define and prioritize your work list. Working with design, you need to analyze and dissect your concept pieces to determine what aspects of each can be readily translated into the game engine. One way of doing this is to ask the question, "What must I create, as a minimum, to get the point across." Or, put another way, how many trees does it take to create a forest, when does a group of houses actually start to look like a town, and so on? Obviously, you can't expect to be able to duplicate your world down to every branch and leaf, but a forest with only a few trees probably won't be convincing.

All this is done with the underlying knowledge that the game still has to run on the target platform. Inevitably, it will come down to questions such as, "If I reduce my target polygon count on my town hall by 200 polygons, how many extra trees will this buy me?"

Once you've outlined the basic pieces necessary to create your game-play environment, you need to identify those aspects of the game that fall into the "game-play–critical" category. These objects will have to go through a more rigorous testing process to ensure functionality, and therefore should be treated as a separate case. These would include any items associated with traps, puzzles, mechanisms, and so on, but the category also includes those aspects of the game that are unique to the genre that you are creating. For example, let's say that in one of the levels, the player will encounter a pirate ship manned by orcs. The ship will have internal spaces, and a large portion of the level's game play takes place on or around the vessel. Therefore, the pirate ship falls into the category of "game-play–critical objects." To assure yourself that a certain task is game-play–critical, ask the question "If I removed this object from the game, would the game-play mechanic suffer significantly?" If the answer is yes, that object is game-play–critical.

After you've identified all of the game-play–critical aspects of the game, the remaining tasks can be grouped in the non–game-play–critical list. These are items that, although they don't affect the actual game-play mechanic directly, are crucial to the successful creation of a convincing environment. For instance, if we look at the previous example, the pirate ship in question was obviously critical to game play, because by removing it you would be eliminating one of the character's main objectives. However, the fish in the ocean, the clouds in the sky, the trees on the beach, and even the water itself are all examples non–game-play–critical objects. They are just as important to the overall gaming experience, for they are the backdrop against which the ship is made to appear real.

GAME-PLAY–CRITICAL VS. NON–GAME-PLAY–CRITICAL. The main difference here is that for game-play–critical items, the design team will need to work hand-in-hand with the art team to ensure proper execution of the game-play mechanic. This dialogue needs to begin early on, and should be a continuation of the initial process of generating the game design document. Getting input from the design team while the concept is still on paper will save headaches later on, and also guarantees that the artistic vision and game-play mechanic remain concurrent. One way to do this is to assign a designer and an artist to each task on the list.

Step 3: Functional Renderings

Form follows function, and for the game-play–critical objects, this is the point in the process where design's input is most appropriate, because the

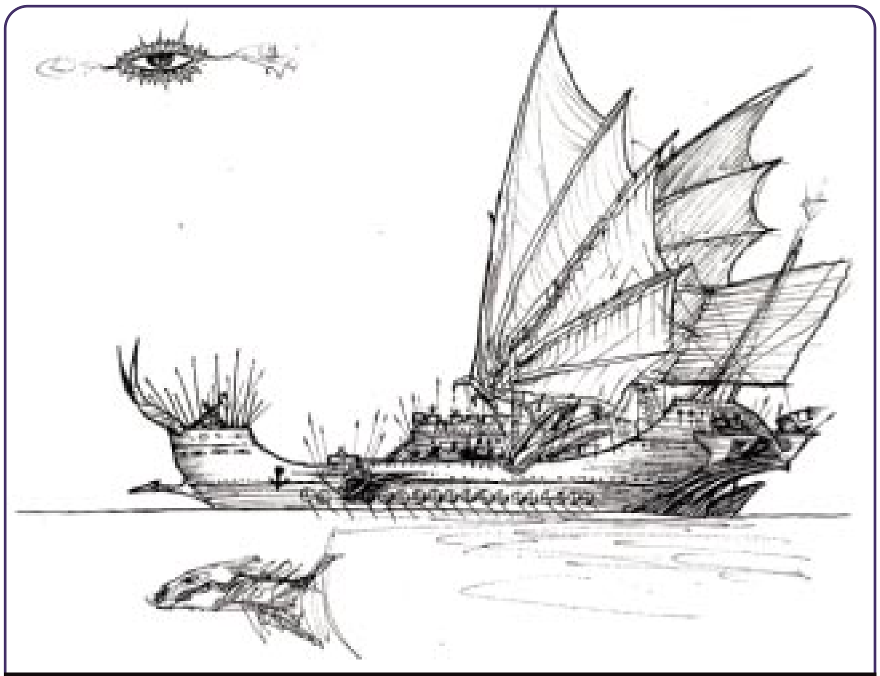


FIGURE 2. *Pirate ship concept piece.*

most crucial feature of these items is their functionality. How does this object work and how does the player interact with it? What are the parameters the artist must hold to ensure the functionality of the item? To answer these questions it is often necessary to reduce the original artistic concept rendering down to a line drawing that only identifies the key functional areas. Getting at the bare bones of the design will help the artist to isolate those aspects of game play that need to be preserved when creating the object in 3D.

Figure 2 shows the initial concept piece for a pirate ship in the game DRAKAN (in development at Surreal Software). The rendering is stylized and holds to the artistic vision of the game. Note the exaggerated details that will be almost impossible to reproduce: individual oars, spear posts, and ship's rigging.

Figure 3 shows a functional rendering of the pirate ship, based on the game-play mechanic of the designers. Note that now particular attention is being paid to the ship internals, where additional game play will take place. Character silhouettes are included so that there is no ambiguity as to the scale of the object. Note again that some of the features from the original concept piece have been removed, and the functional rendering, while not polygonal itself, is closer to the target that the artist will be aiming for when converting the 2D concepts into 3D content.

Note that while this phase usually takes only a few hours to complete for any given item, spending an extra hour working out the bugs on paper can save weeks of work later.

Step 4: Object Construction

Now it's finally time for the artist to go away and work at the computer. From the line drawing generated earlier, and the conceptual art created early on, we have created a clear vision in the artist's head of what he's going to build long before he sits down at his machine. At this point, the artist must merge the functional design with the artistic vision for the game.

Figure 4 shows an example of the pirate ship in its fully modeled and textured form. Note that the polygonal version of the object does not significantly differ from the concept pieces. Except for some minor changes to the decking and hull design, the artist has achieved a true interpretation of the concept piece.

Step 5: Evaluate the Result

Now that the object has been modeled, it is important for the artist and designer to look over the results briefly before implementing the object in the game. Has the artist stayed with-

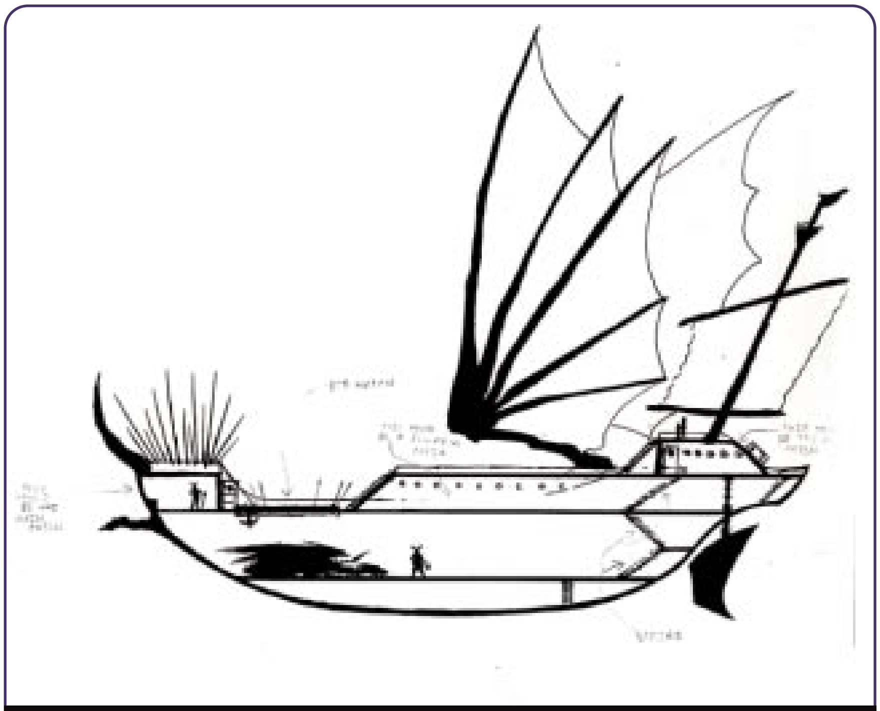


FIGURE 3. *Pirate ship functional rendering.*

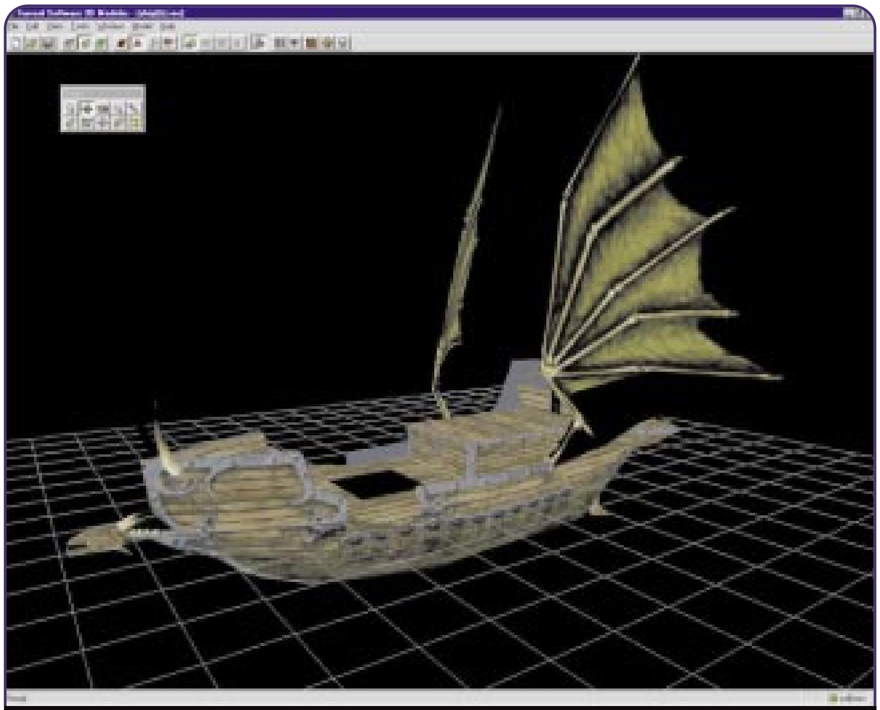


FIGURE 4. *Pirate ship model.*

in the boundaries set forth by the original artistic vision? Does the object meet the functional design criteria originally set forth? Have any problems been identified with the design as a result of actually building the object? Has the design changed significantly, and if so, how will this affect the flow of game play? All these questions should be answered, and the soundness of the design and the functionality should be verified before you spend the time to implement the object in the game.

Step 6: Game-Play Testing

This is the final litmus test to determine whether you've achieved your goal. If you've done a thorough job of completing the preceding steps, it's a good bet that you've hit it right the first time. The true value of the process will become clear the first time you try skipping a few steps and rushing the object into the game. Either the look will be wrong, or the object won't function properly, or both. And in a production environment, this can be a killer because tight deadlines and robust schedules are the norm in this industry. You just can't afford to have production come to a grinding halt as design and programming are bottlenecked while waiting for resources that are being redone for the umpteenth time.

The critical aspect about this phase is timing. Once objects are finished being created, you need to have an efficient pipeline for getting them into the game engine and tested as soon as possible. The dead time between object completion and game-play testing is a black hole in which the iterative feedback process suffers and creative motivation wanes. An artist may work on anywhere from a few dozen to a few hundred objects for a game, and getting the objects implemented and tested in the game while the vision is still fresh in the artist's mind can really make the difference.

If you haven't met your goal, it's time to back up a few steps and regroup with design to determine the problem. If the look is wrong, but the object works correctly, you only need to give the artist some more time working the model. However, if the design is wrong and the object functions incorrectly, both art and design need to work together to fix the problem, and fix it fast, because now you're probably going to be behind schedule and rushing to get the work finished.

Step 7: Sign Off as Completed

This is the last step in the process — when the artist can take the object and make some final adjustments. Most likely, the only thing left to do is polish up the texture maps and adjust mapping coordinates. If you find that you're having to do a lot of polygon modeling and vertex adjustment here, you've probably jumped the gun, and you'll need to go back and do some more game-play testing on the newly modeled areas.

Oddly enough, this is the step in the process where most artists run astray. In an effort to massage the art into looking just a little bit better, they inevitably introduce changes that require retesting the object and diving back into the art path. Furthermore, because most talented artists have a high standard for self-criticism, if left to their own designs, they will end up tweaking the object far more than is necessary or prudent.

The way to get around this is to ensure that the time scheduled for each task is sufficient for completing the process. That way, the artist has just enough time to follow through with a good effort without feeling the need to go back and spend more time on the object later. When you step back from the process and look at the result, you should have a piece of the game that fits with the overall artistic vision and merges seamlessly with the game-play mechanic for the rest of your world. ■

SPECIAL THANKS: Ben Olson, Mike Nichols, Louise Smith, and Phillip Campbell.

Avoiding Time Snafus: Advice to Artists

DEVELOP A PROCESS AND STICK TO IT. Artists are notorious for having inefficient and sporadic work cycles. After all, it's hard to schedule creativity. Have the discipline to create a process and stick to it, particularly at the initial stages of the production cycle. Get your team into the habit of following a process early on so that when it comes to crunch time, they do it instinctively.

USE REALISTIC TIMETABLES. The best plan in the world will fail if the people working under it are not given sufficient time to complete it. Be realistic in your estimates of how long it will take to accomplish each task. There is no point promising a piece of work in two days when you know it will take four. You'll just end up rushing the job and having to rework it anyhow, and with the aggressive schedules that are par for the course in this industry, you don't have that luxury.

DON'T SKIMP ON THE CONCEPTUAL WORK. This is a common error among inexperienced teams, especially with the increasing functionality of today's software packages. Don't forget that as artists, the first thing any of us learned to do was to sketch with pencil and paper, and this is where you'll do your most efficient work. You need to get your design fleshed out on paper before you go anywhere near the computer. You can't afford to wait until you've been working on a piece of art for three or four days to find out that it's not going to work as originally designed. That's time wasted that you just can't afford to lose.

AVOID TEMPORARY OR PLACEHOLDER ART. Even the most disciplined team will eventually resort to using unfinished resources for troubleshooting and game-play testing. This is often necessary to avoid creating a bottleneck for programming or design. Don't let this become common practice. Allowing the team become accustomed to working with placeholders sends you down a slippery slope towards mediocrity. Once a piece of art is in the game, it's all to easy for the artist to write it off and begin work on something else. Consider how unfortunate it would be to have to demo your game to a group of investors or worse, to a surprise interview with the press, and to hear yourself repeating "Oh, ignore that, it's just temporary..." or "Yeah, but when the game's finished that ballista won't really look like a VW, honest!" First impressions are hard to break, and chances are when they think of your game, they'll remember the VW.

Creative Labs' Move from Legacy to Future Products

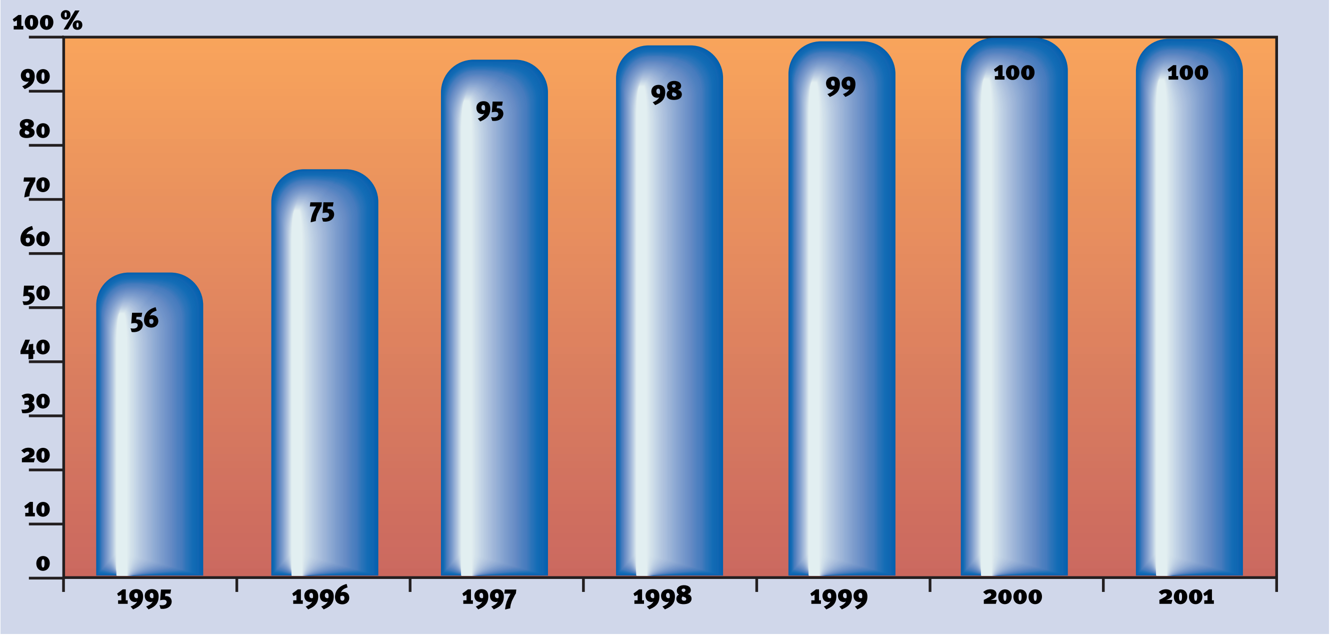
There is an interesting dichotomy in the audio market today. Faster CPUs are eating up the market for low-end audio products. They are taking on the burden of audio signal processing, while the high end is moving towards a greater reliance on 3D audio served up on dedicated hardware.

There is obvious enthusiasm for the realism and quality that high-end audio, particularly 3D audio, adds to a title. Yet, in the midst of this market activity, the game development community seems reluctant to step up its own audio development. It seems that Creative Labs, which pioneered DOS audio, may be the company to push the 3D audio market. Although Creative has more competitors in the 3D audio market than it ever had with Sound Blaster, there is no arguing with the company's sheer size and the extent of its reach. Therefore, in this issue of *Game Developer*, we are going to look at trends in the audio market from the perspective of this one powerful company.

The Market

Multimedia, particularly multimedia for games, is one of the primary selling attributes of the consumer PC. And so, not surprisingly, audio has been figuring more and more prominently as a standard feature in consumer PCs since 1994. Thus, this segment has rapidly become audio-enabled, with 98 percent of new shipments leaving the loading dock with audio in 1998. By definition now, a consumer PC must be audio-visual. As this segment continues to grow through the end of the decade, this saturation will become virtually complete. The chart in Figure 1.illustrates audio's current market penetration.

FIGURE 1. *World-wide audio penetration in the consumer market segment, 1995–2001. Source: International Data.*



Omid Rahmat works for Doodah Marketing as a copywriter, consultant, tea boy, and sole employee. He also writes regularly on the computer graphics and entertainment markets for online and print publications. Contact him at omid@compuserve.com.

Audio is going beyond the consumer PC as well. Roger Kay, senior analyst of IDC Research, tracks the audio market. Kay says that, "In 1994, a grand separation occurred in the PC industry. As part of an effort to segment products and markets more distinctly, OEMs began to target two broad customer types: commercial buyers and consumers. Not that this segmentation hadn't been recognized to some degree before, but now, in an effort to compete more effectively, vendors were designing, configuring, and distributing PCs in different ways depending on whether they were considered business or home systems. During the past four years, these distinctions have become ever more pronounced, with commercial PCs being those distributed primarily through resellers, VARs, and other indirect channels and configured with networking, management software, and business productivity applications. Consumer PCs are those sold directly or through retailers and configured with multimedia hardware and software, modems, and 'lifestyle' software (such as games, web access and applications, and consumer reference information).

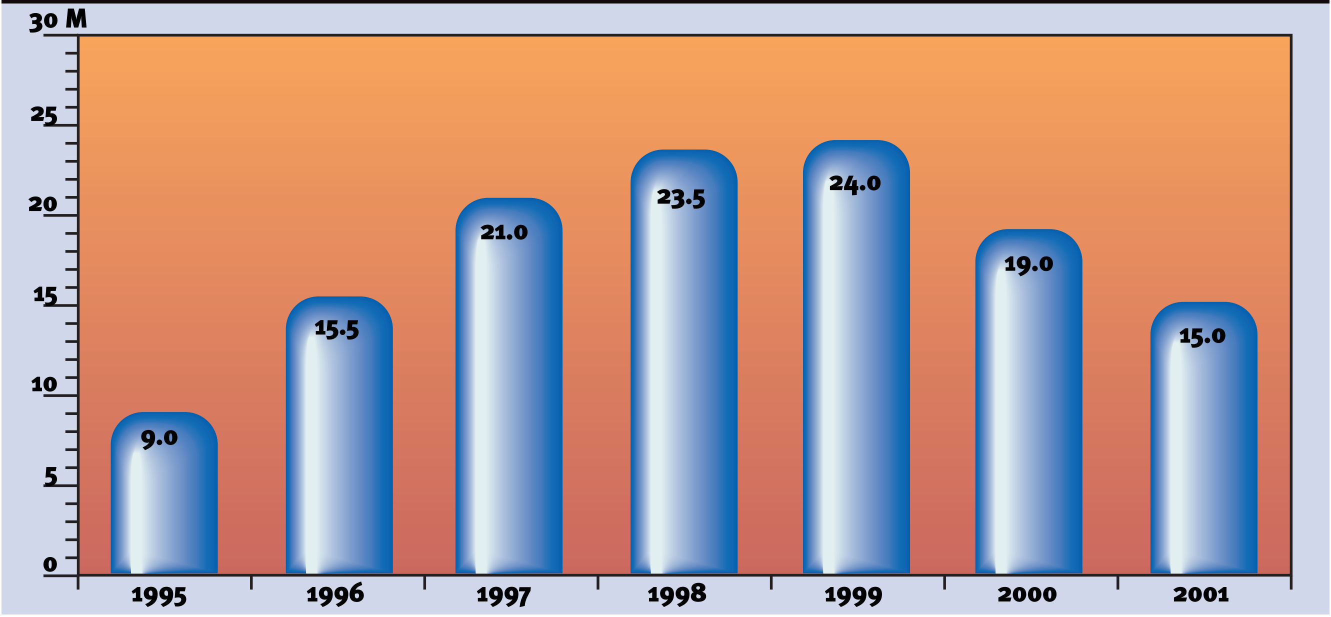
"Although commercial PCs have not needed much in the way of audiovisual capabilities until now, this situation is changing rapidly. Two related developments are primarily responsible for the increased need for audio-visual in commercial PCs. The first is the advent of the World Wide Web as a business tool. Many workers now have reason to go out on the Web to do their jobs. Researchers, journalists, marketers, and many other job categories need to find information rapidly about their subjects, customers, or markets, and the Web is a gold mine for this type of data. As audio-visual information becomes more common on the Web (in the form of singing, dancing Java applets, recordings of speeches, and other sound snippets), a greater need for audio-visual capabilities in commercial systems will arise."

... it does put game developers in the position of having to make nasty choices about the future. Just like 3D graphics, the question becomes, "Which 3D audio?"

So, simply put, we now know that just as every computer is guaranteed to have a display and graphics controller, it is equally likely to have an audio component as well. The installed base of audio is no longer the issue, but rather where the consumer market is pushed in order to position it away from mainstream audio, and thus, make it a more attractive multimedia option. Normally, we could rely on statistics of add-in board products. Add-in board shipments indicate the health of the market for peripherals in general. In the case of audio, however, it might seem otherwise judging by the figures shown below in Figure 2.

This is where it gets tricky for game developers, and why concentrating on Creative might help to paint a clearer picture. The drop-off in add-in board sales in the next couple of years is actually good news for quality audio. What is happening in both the audio and graphics markets is that the baseline, or integrated components in a consumer PC, are better quality and cheaper. They have to be in order to satisfy the low-cost PC model that seems to be driving the consumer market these days. This leaves the add-in board market as a smaller market, but one in which the performance and quality of products needs to be significantly higher than the baseline-

FIGURE 2. *Worldwide Audio Add-In Board Shipments, 1995–2001. Source: International Data Corporation*



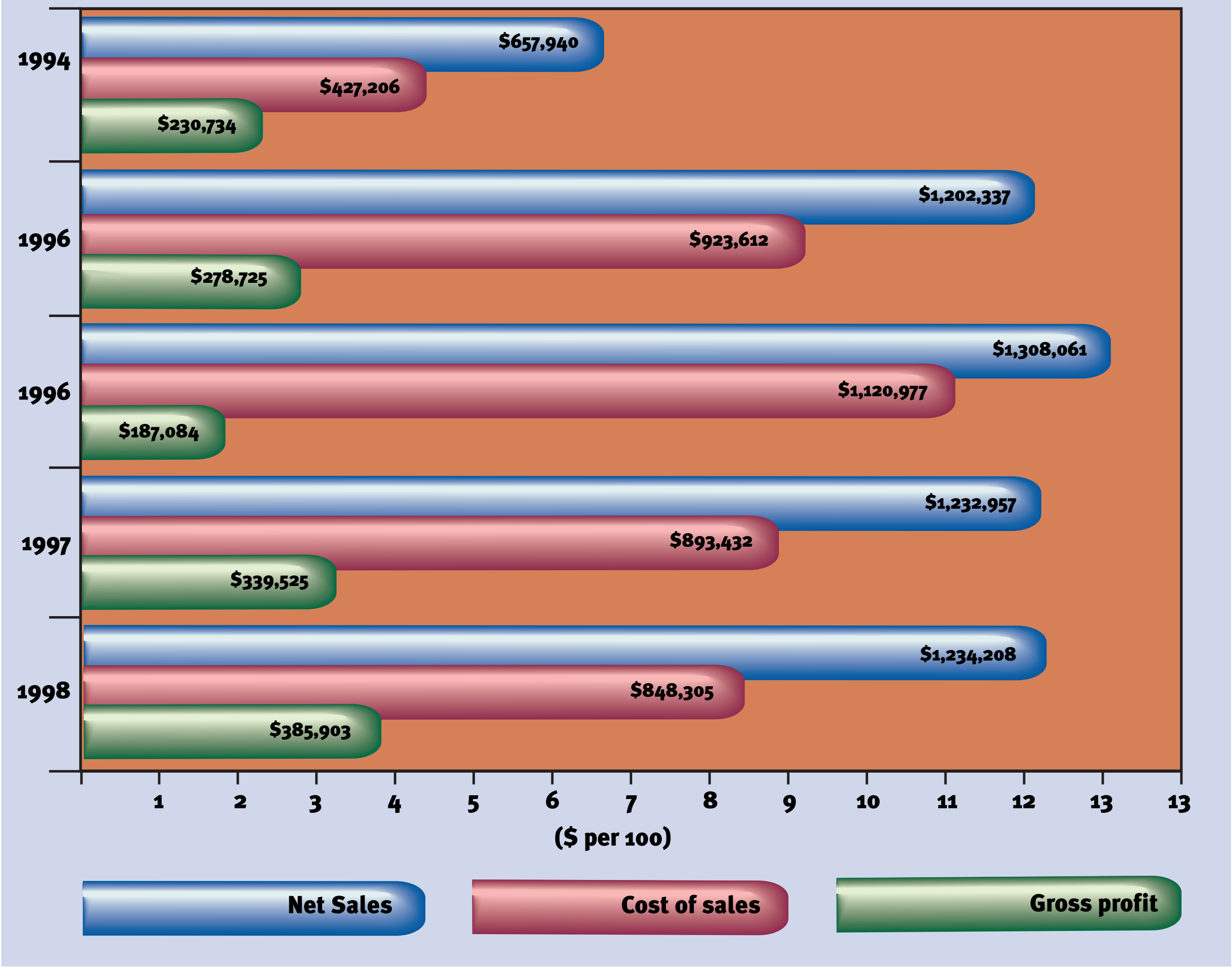
integrated products. Think of it as a natural segmentation: some PCs will come fully equipped like boom boxes, and some PCs will have separate, quality components, like high-fidelity stereo systems. The game players, at least the hard-core enthusiasts, will go for the separates. That's the good news. The bad news is that it does put game developers in the position of having to make nasty choices about the future. Just like 3D graphics, the question becomes, "Which 3D audio?" Yet, it probably won't be that bad because, unlike the graphics market where no single company has ever dominated the field for long, in the audio market Creative has been very solid, and looks to remain so.

Blasting Sound into the 3D Age

While I refer to Creative Labs, it is worth noting that the company is actually a wholly-owned subsidiary of Creative Technology of Singapore, and it also offers a number of related products, including graphics, video, communications, and multimedia kits and accessories. According to IDC, Creative Labs is by far the world's largest audio vendor, holding almost 90 percent of the U.S. retail add-in market and almost 60 percent of the worldwide OEM market. Having originated and pioneered the audio market with the Sound Blaster standard, Creative not only has an enviable legacy in the industry, but it also owns the market for legacy audio. If there is a DOS game, then it's served by Creative's technology.

Using its market dominance and financial clout, Creative has been steadily building up its technology and product reserves to maintain its leadership in audio. A few years ago, Creative purchased E-Mu Systems, and got an injection of technology in the form of the Advanced WavEffects Engine (AWE). The company also owns Ensoniq, and more recently acquired speaker manufacturer Cambridge SoundWorks. Today, Creative makes its own chips and supports everything from AC'97 to PCI, as well its own legacy. It numbers among its clients motherboard vendors such as Intel,

FIGURE 3. *Creative's financial performance suffered briefly between 1996 and 1997 as the company came to grips with the fact that it would no longer control audio standards. In the meantime, the company has amassed $400 million of cash to fend off its smaller, and less powerful competitors. (Creative's fiscal year ends in June).*



Micronics, Acer, ECS, Asus, and the Taiwanese motherboard crowd. In addition, the company also has a healthy OEM add-in card business with the direct PC vendors, Dell, Micron, and Gateway, as well as with a large number of VARs and systems integrators to which it sells directly or through distributors. Finally, there is the retail business, a great contributor to Creative's top line.

The challenge facing all audio products vendors is that the low-end add-in audio board business will disappear in the coming years. It is PCI audio that is driving the industry today, but with almost 100 percent consumer penetration of audio, built-in PCI audio in both consumer and business PCs is a given. Specifically, the impact of a multimedia-rich Internet is making audio a compulsory feature in all computing devices, whereas only two years ago, corporate PC buyers were adamant about not having audio in their users' systems. Of course, in referring to ubiquitous audio we are talking primarily about the influence of Microsoft and Intel, one supplying integrated OS support for audio, and the other handing over CPU cycles to process it. Taking a closer look at Creative's strategies to deal with the emergence of prevalent audio is a clear roadmap to the future of the industry.

Micah Stroud, audio product marketing manager for the Americas at Creative Labs, sees some important trends. "We are in the process of moving from ISA to PCI audio. By the first quarter of 1999, it should be complete. The PCI bus is obviously faster, and as a result, one of the primary things that has become of interest to developers is multi-channel audio. On the ISA bus, we had to deal with interrupt driven audio that demanded a disproportionate amount of the CPU's time, and required a step by step, interrupt driven approach to audio playback. On the PCI bus, you have bus mastering, which means you generate an interrupt to play the sound, and the rest is in the hands of the audio card. It all happens seamlessly, and it doesn't involve the CPU."

Stroud is careful to point out that this is not true of all PCI cards, but he clearly sees that this is the way all audio cards have to go. Another reason why the faster performance audio market is going to be important to the audio vendors is 3D audio. Stroud says that, "What we have found is that there is a great deal of interest in 3D audio. We are seeing, at the end of 1998, a shakeout of APIs, and the end result is that the interface for 3D audio is going to be DirectSound3D."

Creative admits that game developers might be cynical about DirectSound3D, but the company has taken the steps to build on top of DirectSound3D in order to push the market in certain directions. For Christmas 1998, the company lined up 50 game titles to support its Environmental Audio platform. Specifically, the company's API is called EAX. The extensions describe the fundamental properties of rooms. Absorbency of walls, reverberations, and up to thirty presets are exposed, as well as the parameters to tweak for optimal effects. Stroud says, "One of the things that we committed to the game developer community is that we will provide environmental audio extensions to anyone who wants it for free. So far, we have signed on companies such as QSound, Spatializer, and many more. Arguably, we cover 90 percent of the market with these companies. We have also offered EAX up to IASIG as an industry standard."

George Thorn, director of developer relations for Creative Labs, says, "The proof of the pudding is in shipping product. Christmas 1998 will establish DirectSound3D. For example, we ported our EAX renderer to our Ensoniq-based cards, using the host to render environment audio effects. There are 6 million of those cards out there. I firmly believe that, finally, audio is no longer being considered as something that you can throw in at the end of a product and expect to be competitive. You know, just effects and music. More developers are investing in a core team to produce original music for titles. Hollywood recognized that audio was important. I mean, if the sound in the movie theater wasn't going to be better than watching a video at home, then why bother? Sound is really, really important for a compelling entertainment experience."

But technology and quality may not be enough. Creative has also sought to litigate against its competitors, or rather, against the suppliers of its competitors. Take the case of Diamond Multimedia, a company that has used the success of its 3Dfx graphics products to go hand in hand with its 3D audio products. First, Creative sued ESS Technology, then Aureal. In the case between ESS Technology and Creative, the two companies have reached an agreement. However, the original suit was filed by Creative Technology in March 1998, just before the announcement of Diamond's Sonic Impact products using ESS's Maestro-2 audio chip. Creative is suing Aureal for false advertising, in addition to claims that Aureal's Vortex AU8820 infringes on Creative's Patent No. 5,342,990, named Digital Sampling Instruments Employing Cache-Memory. Aureal supplies the chipsets for Diamond's Monster 3D Sound product line.

The challenge facing all audio products vendors is that the low-end add-in audio board business will disappear in the coming years.

The Multichannel Product Line

One issue that Creative was told to deal with by game developers, and that may prove to be another competitive advantage for the company, is the speakers. Thorn says, "About two years ago Creative embarked on a certain path. We knew the future of audio was going to be multichannel. We talked to developers, and overwhelmingly the reply came back that they were interested in surround sound. The caveat was that developers have to mix down to two channels. They were riding the promise of people hooking up their PCs to their home stereos, and although it is happening, it is to a small extent. So, if we want to have multichannel audio, the problem is the speakers. Game developers basically told us that we had to go down the speaker route to enable this market." Creative bought into Cambridge SoundWorks, and is now putting multichannel speakers into affordable packages.

And just in case developers still don't get the message, the company is using the professional studio products of E-mu and Ensoniq, and filtering the technology into their mainstream markets. So, professional and semi-professional audio development products are also finding their way into Creative's customers' hands. This strategy isn't just about the technology filtering down, but also about putting boards in the hands of developers and saying, "Develop on this, and know that your users will be playing back the results on similar products." All these strategies put Creative well in advance of their competitors, but there is the fact that the company has shipped 55 million Sound Blaster products, and well over 10 million of that base is PCI products. As ISA disappears for good in 1999, Creative is doing all the right things to move its customer base onto its own PCI cards. And as the add-in board market shrinks, Creative is moving into selling audio chipsets for the motherboard and laptops. The future may determine that audio doesn't even sit inside of a PC, but is attached through USB. The audio experience is getting better, and the products to support 3D audio are becoming prevalent. Creative will be hard to beat in this arena for some time to come. ■

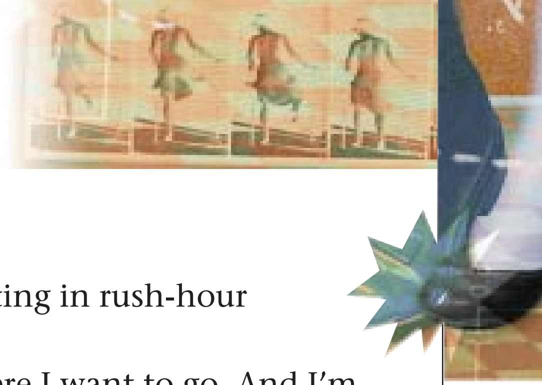
Back

Coordinated Unit Movement

B Y D A V E C . P O T T I N G E R

After several close calls, Dave managed to avoid getting a "real job" and joined Ensemble Studios straight out of college a few years ago (just in time to the do the computer-player AI for a little game called AGE OF EMPIRES*). These days, Dave spends his time either leading the development of Ensemble Studios' engines or with his lovely wife Kristen. Dave can be reached at dpottinger@ensemblestudios.com.*

Illustration by Stuart Bradford



How many times have you been sitting in rush-hour traffic thinking, “Hey, I know where I want to go. And I’m sure everyone around me knows where they want to go, too. If we could just work together, I’ll bet we would all get where we wanted to go a lot easier, faster, and without rear-ending each other”? As your frustration rises, you realize that impatient commuters aren’t the most cooperative people. However, if you’re a game player, uncooperative resource gatherers and infantry are probably even more frustrating than a real-life traffic jam.

Figuring out how to get hundreds of units moving around a complex game map in real time — commonly referred to as pathfinding — is a tough task. While pathfinding is a hot industry buzzword, it's only half of the solution. Movement, the execution of a given path, is the other half of the solution. For real-time strategy games, this movement goes hand in hand with pathfinding. An axeman certainly needs a plan (as in, a path) for how he's going to get from one side of his town to the other to help stave off the enemy invasion. If he doesn't execute that plan using a good movement system, however, all may be lost.

Game Developer has already visited the topic of pathfinding in such past articles as "Smart Move: Path-Finding" by Brian Stout (October/November 1996) and "Real-Time Pathfinding for Multiple Objects" by Swen Vincke (June 1997). Rather than go over the same material, I'll approach the problem from the other side by examining the ways to execute a path that's already been found. In this article, I'll cover the basic components of an effective movement system. In a companion article in next month's *Game Developer*, I'll extend these basic concepts to cover higher-order movement and implementation. Though the examples in these articles focus mainly on a real-time strategy game, the methods I'll describe can easily be applied to other genres.

Basic Definitions

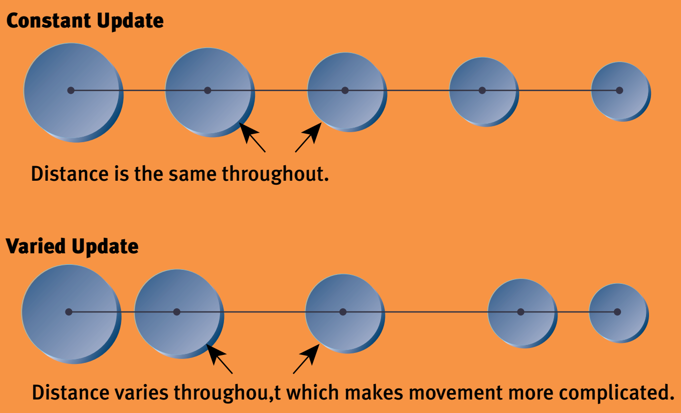
Movement. The execution of a path. Simple movement algorithms move a unit along a path, while more complex systems check collisions and coordinate unit movement to avoid collisions and allow otherwise stuck units to move.
Pathfinding. The act of finding a path (a planned route for a unit to get from point A to point B). The algorithm used can be anything from a simple exhaustive search to an optimized A* implementation.
Waypoint. A point on a path that a unit must go through to execute the path. Each path, by definition, has one waypoint at the start and one waypoint at the end.
Unit. A game entity that has the ability to move around the game map.
Group. A general collection of units that have been grouped together by the user for convenience (usually to issue the same order to all of the units in the group). Most games try to keep all of the units in a group together during movement.
Formation. A more complex group. A formation has facing (a front, a back, and two flanks). Each unit in the formation tries to maintain a unique relative position inside the formation. More complex models provide an individualized unit facing inside of the overall formation and support for wheeling during movement.
Hard Movement Radius. A measure of the volume of a unit with which we absolutely do not allow other units to collide.
Soft Movement Radius. A measure of the volume of a unit with which we would prefer not to collide.
Movement Prediction. Using the movement algorithms to predict where a unit will be at some point in the future. A good prediction system will take acceleration and deceleration into account.
Turn Radius. The radius of the tightest circle a unit can turn on at a given speed.

Movement Issues Facing Game Developers

Before we dive into coordinated unit movement, let's take a look at some of the movement issues facing game developers today. Most of these have to do with minimizing CPU load versus maximizing the accuracy and intelligence of the movement.

MOVING ONE UNIT VERSUS MOVING MULTIPLE UNITS. Moving one unit is generally pretty simple, but methods that work well for one unit rarely scale up effortlessly for application to hundreds of units. If you're designing a system for

FIGURE 1. *Varied update lengths cause units to move differing distances each update.*



Constant Update
Distance is the same throughout.
Varied Update
Distance varies throughou,t which makes movement more complicated.

hundreds of units, it will need to be very conservative in its CPU use.

SOME MOVEMENT FEATURES ARE CPU INTENSIVE. Very few games that move hundreds of units support advanced behavior such as modeling the acceleration and deceleration of these units. The movement of large ships and heavily armored units has a lot more realism with acceleration and deceleration, but that realism comes at a high cost in terms of extra CPU usage. The actual movement calculation becomes more complicated because you have to apply the time differential to the acceleration to create the new velocity. As we extend our movement system to handle prediction, we'll see that acceleration and deceleration complicate these calculations as well. Modeling a turn radius is also difficult because many pathfinding algorithms are not able to take turn radii into account at all. Thus, even though a unit can find a path, it may not be able to follow that path because of turn radius restrictions. Most systems overcome this deficiency by slowing the unit down to make a sharp turn, but this involves an extra set of calculations.

LISTING 1. *The movement algorithm in pseudocode.*

```
Top of movement state loop:
   {
   If we're in IncrementWaypoint state:
      Increment our waypoint.
      If we're on a patrol
         Grab the next waypoint as defined by the patrol direction.
         Set state to WaitingForPath.
      Else
         If we're out of waypoints
            Set state to ReachedGoal.
         Else
            Set state to WaitingForPath.

   If we're in ReachedGoal state:
      Make the appropriate notifications (if any).
      We're done. Stop the walking animation. Exit function.

   If we're in WaitingForPath state:
      Find a path and save it.
      If we could not find one
         We've failed. Exit function.
   Calculate the direction we need to head in to get to our desired waypoint.
   Modify that direction by any limitations such as turn radius.
   Using that new direction, calculate where we'll end up after this move.

   If that new position causes a collision
      Set state to WaitingForPath.
      Jump back to the top of the loop.

   Using the current and future position:
      If we're closer to the waypoint before moving
         Set state to IncrementWaypoint
         Go back to top of loop.
      If we're going to jump over the waypoint during this move
         Set state to IncrementWaypoint.

   Break out of loop.
   }
Set the accelerations accordingly.
Do the actual move.
Set or update any animation hooks that we might have.
Update our predicted positions.
```

DIFFERENT LENGTHS FOR THE MAIN GAME UPDATE LOOP. Most games use the length of the last pass through update loop as some indication of how much time to simulate during the next update pass. But such a solution creates a problem for unit movement systems because these lengths vary from one update to the next (Figure 1). Unit movement algorithms work much better with nice, consistent simulation intervals. A good update smoothing system can alleviate this problem quite a bit.

SORTING OUT UNIT COLLISIONS. Once units come into contact with one another, how do you get them apart again? The naïve solution is just never to allow units to collide in the first place. In practice, though, this requirement enforces exacting code that is difficult to write. No matter how much code you write, your units will always find a way to overlap. More importantly, this solution simply isn't practical for good game play; in many cases, units should be allowed to overlap a little. Hand-to-hand combat in Ensemble Studios' recent title AGE OF EMPIRES should have been just such a case. The restriction for zero collision overlap often makes units walk well out of their way to fight other units, exposing them to needless (not to mention frustrating) additional damage. You'll have to decide how much collision overlap is acceptable for your game and resolve accordingly.

MAP COMPLEXITY. The more complex the map is, the more complicated and difficult good movement will be to create. As game worlds and maps are only getting more intricate and realistic, the requirement for movement that can handle those worlds goes up, too.

RANDOM MAPS OR CONTROLLED SCENARIOS? Because you can't hard-code feasible paths, random maps are obviously more difficult to deal with in many cases, including pathfinding. When pathfinding becomes too CPU intensive, the only choice (aside from reducing map complexity or removing random maps) is to decrease the quality of the pathfinding. As the quality of the pathfinding decreases, the quality of the movement system needs to increase to pick up the slack.

MAXIMUM OBJECT DENSITY. This issue, more than anything, dictates how accurate the movement system must be. If your game has only a handful of moving objects that never really come into contact with one another (as is the case with most any first-person shooter), then you can get away with a relatively simple movement system. However, if you have hundreds of moving objects that need to have collision and movement resolution on the scale of the smallest object (for example, a unit can walk through a small gap between two other units), then the quality and accuracy requirements of your movement system are dramatically raised.

Simple Movement Algorithm

Let's start with some pseudo code for a simple, state-based movement algorithm (Listing 1). While this algorithm doesn't do much more than follow a path and decide to find a new path when a collision is found, it does work equally well for both 2D and 3D games. We'll start in a given state and iterate until we can find a waypoint to move towards. Once we find that point, we break out of the loop and do the movement.

There are three states: `WaitingForPath`, `ReachedGoal`, and `IncrementWaypoint`. The movement state for a unit is preserved across game updates in order to allow us to set future events, such as the "automatic" waypoint increment on a future game update. By preserving a unit's movement state, we lessen the chance that a unit will make a decision on the next game update that counters a decision made during the current update. This is the first of several planning steps that we'll introduce.

We assume that we'll be given a path to follow and that the path is accurate and viable (meaning, no collisions) at the time it was given to us. Because most strategy games have relatively large maps, a unit may take several minutes to get all the way across the map. During this time, the map can change in ways that can invalidate the path. So, we do a simple collision check during the state loop. At this point, if we find a collision, we'll just repath. Later on, we'll cover several ways to avoid repathing.

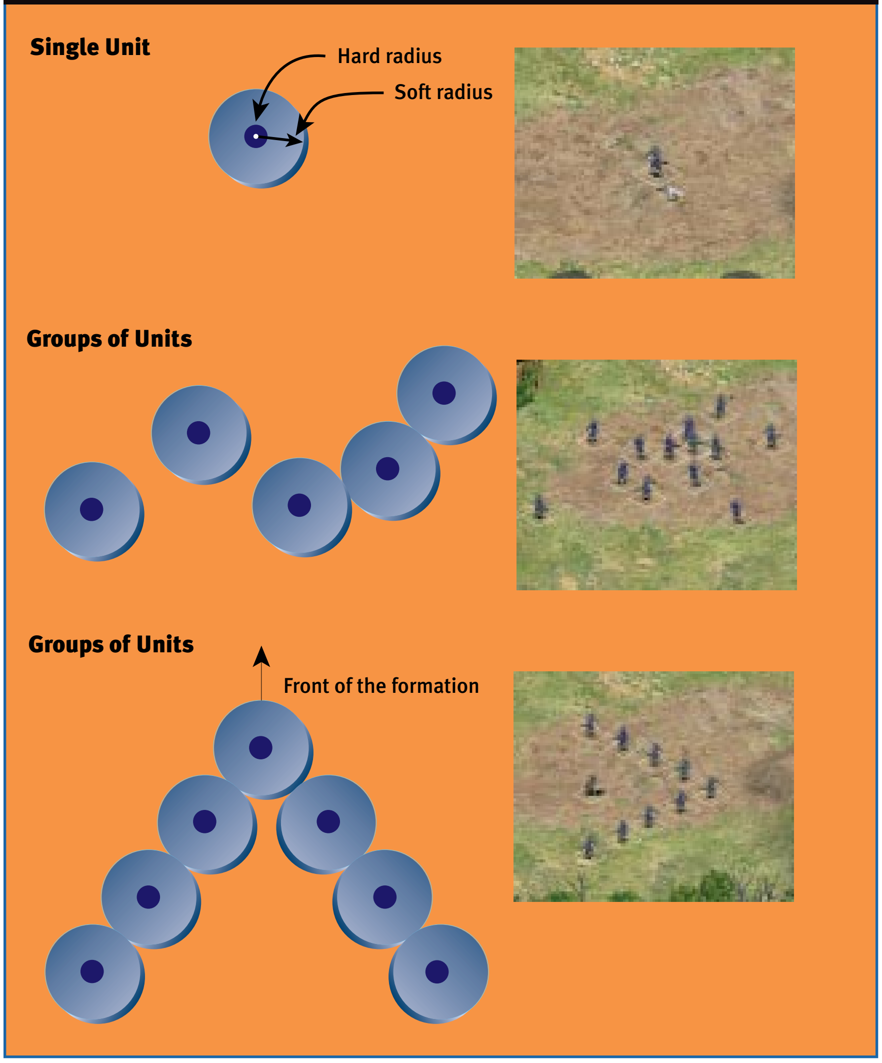


FIGURE 2. *Collision entities.*

Collision Determination

The basic goal of any collision determination system is to find out if two units have collided. For the time being, we'll represent all collisions as two-entity collisions. We'll cover compound collisions (collisions involving three or more entities) next month. Once a collision is found, each entity needs to know about the collision in order to make appropriate movement decisions.

Basic collision determination for most strategy games consists of treating all units as spheres (circles in 2D) and doing a simple spherical collision check. Whether or not such a system is sufficient depends on the specific requirements of a game. Even if a game implements more complex collision — such as oriented bounding boxes or even low-level polygon to polygon intersection tests — maintaining a total bounding sphere for quick potential collision elimination will usually improve performance.

There are three distinct entity types to take into account when designing a collision system: the single unit, a group of units, and a formation (Figure 2). Each of these types can work well using a single sphere for quick collision culling (elimination of further collision checks). In fact, the single unit simply uses a sphere for all of its collision checking. The group and the formation require a bit more work, though.

For a group of units, the acceptable minimum is to check each unit in the group for a collision. By itself, this method will allow a non-grouped unit to sit happily in the middle of your group. For our purposes, we can overlook this discrepancy, because formations will provide the additional, more rigid collision checking. Groups also have the ability to be reshaped at any time to accommodate tight quarters, so it's actually a good idea to keep group collision checking as simple as possible.

A formation requires the same checks as a group, but these check must further ensure that there are no internal collisions within the formation. If a formation has space between some of its units, it is unacceptable for a non-formed unit to occupy that space. Additionally, formations generally don't have the option to reshape or break. However, it's probably a good idea to implement some game rules that allow formations to break and reform on the other side of an obstacle if no path around the obstacle can be found.

For our system, we'll also keep track of the timing of the collision. Immediate collisions represent collisions currently existing between two objects. Future collisions will happen at a specified point in the future (assuming neither of the objects changes its predicted movement behavior). In all cases, immediate collisions have a higher resolution priority than future collisions. We'll also track the state of each collision as unresolved, resolving, or resolved.

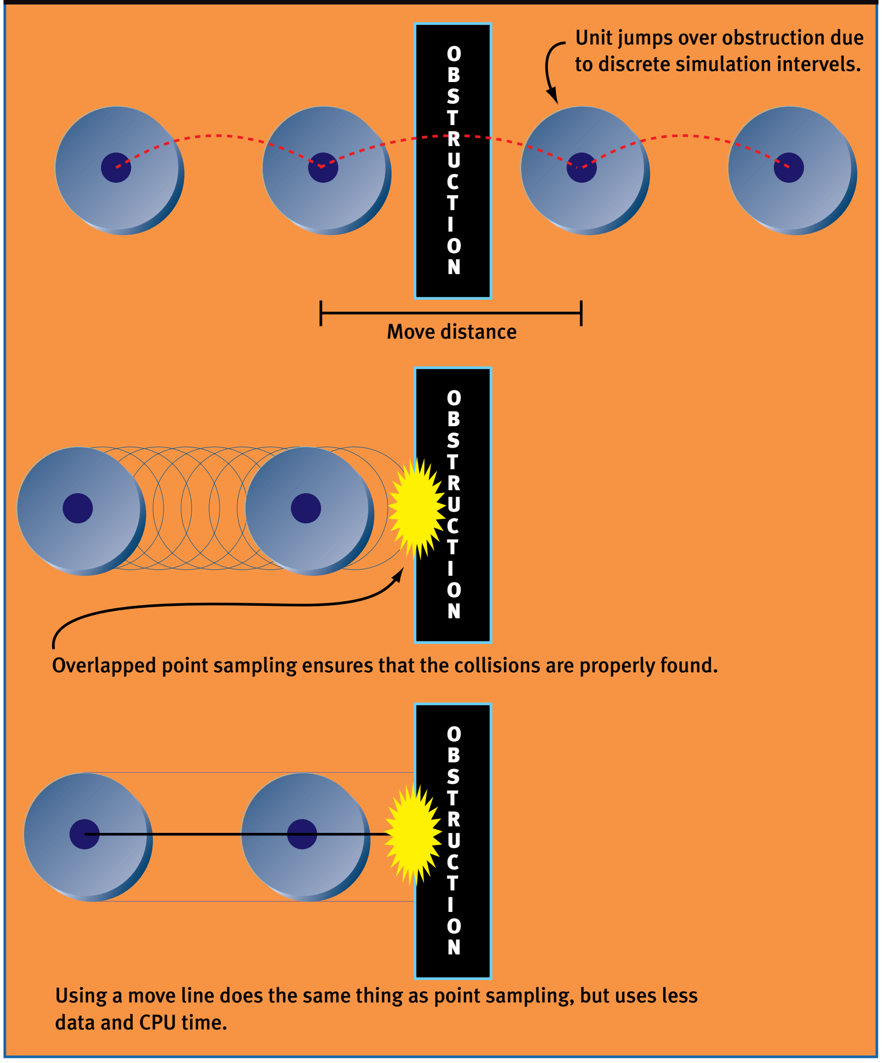
Discrete vs. Continuous Simulation

Most movement algorithms are discrete in nature. That is, they move the unit from point A to point B without considering what might be between those two points, whereas a continuous simulation would consider the volume between the two points as well. In a lag-ridden Internet game, fast moving units can move quite a distance in a single game update. When discrete simulations are coupled with these long updates, units can actually hop over other objects with which they should have collided. In the case of a resource gathering unit, no one really minds too much. But players rarely want enemy units to be able to walk through a wall. While most games work around this problem by limiting the length of a unit's move, this discrete simulation problem is relatively easy to solve (Figure 3).

One way to solve the problem is to sub-sample each move into a series of several smaller moves. Taking the size of the moving unit into account, we make the sampling interval small enough to guarantee that no other unit can fit between two of the sample points. We then run each of those points through the collision determination system. Calculating all of those points and collisions may seem overly expensive, but later on we'll see a potential way to offset most of that cost.

Another method is to create what we'll call a move line. A move line represents the unit's move as a line segment starting at point A and ending at point B. This system creates no extra data, but the collision check does have an increase in complexity; we must convert from a simple spherical collision check to a more expensive calculation that involves finding the distance from a point to a line segment. Most 3D games have already implemented a fast hierarchical system for visible object culling, so we can reuse that for collision culling. By quickly narrowing down the number of potential collisions, we can afford to spend more time checking collisions against a small set of objects.

FIGURE 3. *Solving the problem with discrete movement simulation.*


OBSTRUCTION
Unit jumps over obstruction due to discrete simulation intervels.
Move distance
OBSTRUCTION
Overlapped point sampling ensures that the collisions are properly found.
OBSTRUCTION
Using a move line does the same thing as point sampling, but uses less data and CPU time.

Predicted Positions

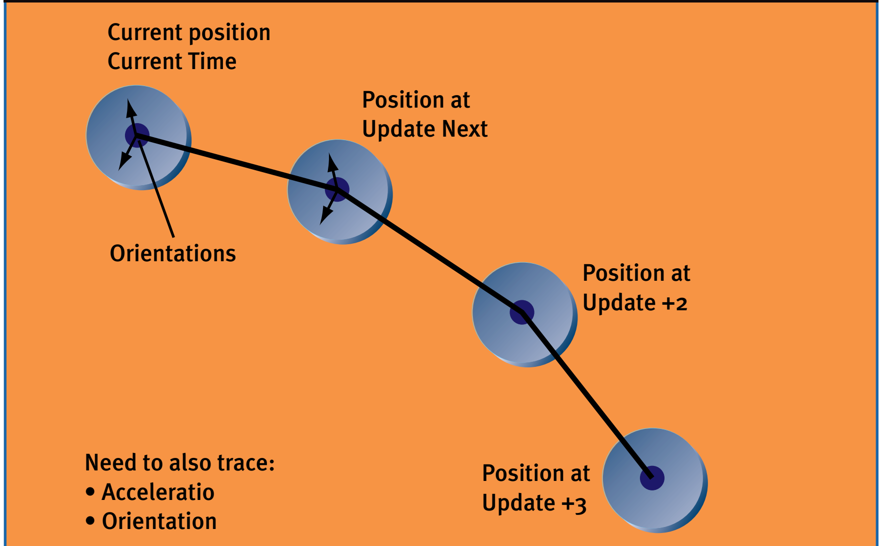
Now that we have a simple movement algorithm and a list of unit collisions, what else do we need to get decent unit cooperation? Position prediction.

Predicted positions are simply a set of positions (with associated orientations and time stamps) that indicate where an object will be in the future (Figure 4). A movement system can calculate these positions using the same movement algorithm that's used to move the object. The more accurate these positions are, the more useful they are. Position prediction isn't immediately free, though, so let's look at how to offset the additional CPU usage.

The most obvious optimization is to avoid recalculating all of your predicted positions at every frame. A simple rolling list works well (Figure 5); you can roll off the positions that are now in the past and add a few new positions each frame to keep the prediction envelope at the same scale. While this optimization doesn't get rid of the start-up cost of creating a complete set of prediction positions the first time you move, it does have constant time for the remainder of the movement.

The next optimization is to create a prediction system that handles both points and lines. Because our collision determination system already supports points and lines, it should be easy to add this support to our prediction system. If a unit is traveling in a straight line, we can designate an enclosed volume by using the current position, a future position, and the unit's soft movement radius. However, if the object has a turn radius, things get a little more complicated. You can try to store the curve as a function, but that's too costly. Instead, you're better off doing point sampling to create the right predicted points (Figure 6). In the end, you really want a system that seamlessly supports both point and line predictions, using the lines wherever possible to cut down on the CPU cost.

FIGURE 4. *A closer look at the predicted positions.*



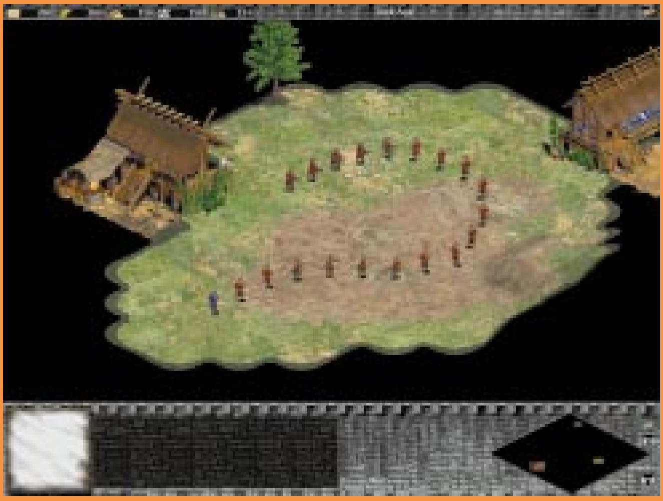


FIGURE 5. *Rolling list of predicted positions.*

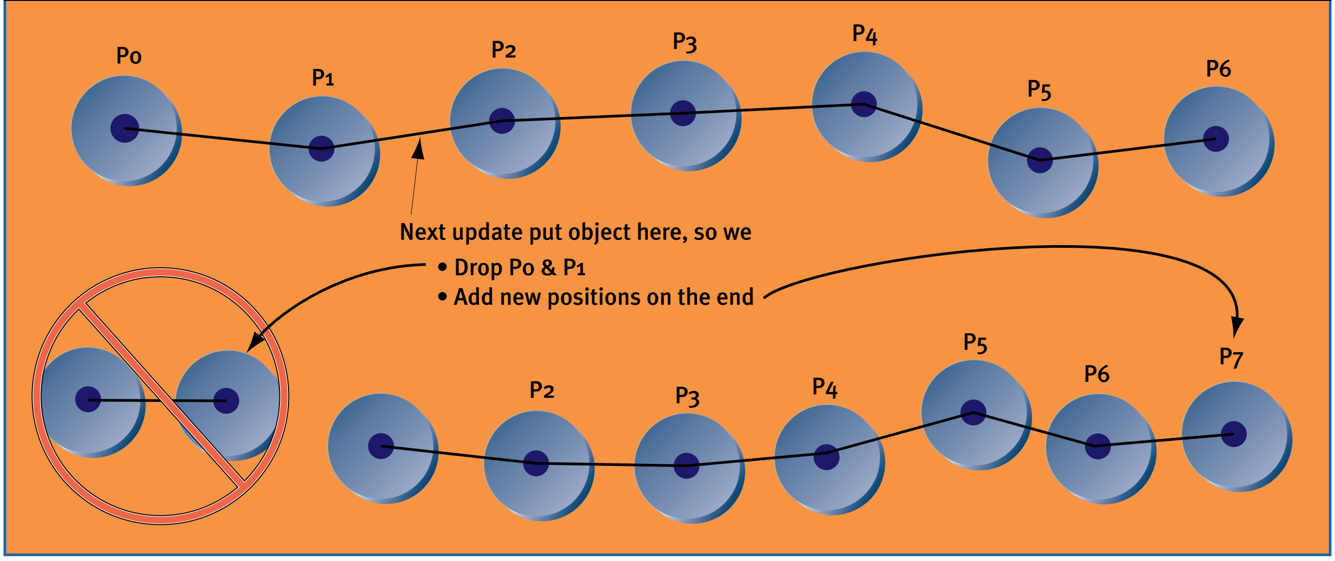
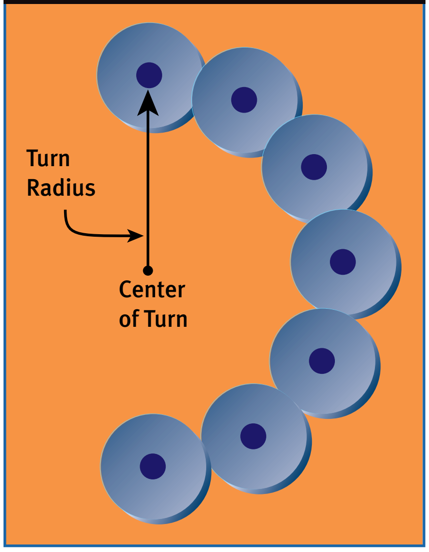

Po
P1
P2
P3
P4
P5
P6
Next update put object here, so we
• Drop Po & P1
• Add new positions on the end
P2
P3
P4
P5
P6
P7

FIGURE 6. *Using predicted positions with a turn radius*



The last optimization we'll cover is important and perhaps a little nonintuitive. If we're going to get this predicted system with as little overhead as possible, we don't want to duplicate our calculations for every unit by predicting its position and then doing another calculation to move it. Thus, the solution is to predict positions accurately, and then use those positions to move the object. This way, we're only calculating each move once, so there's no extra cost aside from the aforementioned extra start-up time.

In the actual implementation, you'll probably just pick a single update length to do the prediction. Of course, it's fairly unlikely that all of the future updates will be consistent. If you blindly move the unit from one predicted position to the next without any regard to what the actual update length currently is, you're bound to run into some problems. Some games (or some subset of objects in a game) can accept this inaccuracy. Those of us developing all the other games will end up adding some interpolation so that can quickly adjust a series of predicted points that isn't completely accurate. You also need to recognize when you're continually adjusting a series of predicted positions so that you cut your losses and just recalculate the entire series.

Most of the rest of the implementation difficulties arise from the fact that we use these predicted positions in collision detection just as we do for the object's actual current position. You should easily see the combinatorial explosion that's created by comparing predicted positions for all units in a given area. However, in order to have good coordinated unit movement, we have to know where units are going to be in the near future and what other units they're likely to hit. This takes a good, fast collision determination system. As with most aspects of a 3D engine, the big optimizations come from quickly eliminating potential interactions, thus allowing you to spend more CPU cycles on the most probable interactions.

Unit to Unit Cooperation

We've created a complex system for determining where an object is going to be in the future. It supports 3D movement, it doesn't take up much more CPU time than a simple system, and it provides an accurate list of everything we expected a unit to run into in the near future. Now we get to the fun part.

If we do our job well, most of the collisions that we must deal with are future collisions (because we avoid most of the immediate collisions before they even happen). While the baseline approach for any future collision is to stop and repath, it's important to avoid firing up the pathfinder as much as possible.

This set of collision resolution rules is a complete breakdown of how to approach the problem of unit-to-unit collision resolution (from a unit's frame of reference).

UNRESOLVED COLLISIONS.

Case 1. If both units are not moving:

1. If we're the lower-priority unit, don't do anything of our own volition.
2. If we're the higher-priority unit, figure out which unit (if any) is going to move and tell that unit to make the shortest move possible to resolve the hard collision. Change the collision state to **`resolving`**.

Case 2. If we're not moving, and the other unit is moving, we don't do anything.

Case 3. If we're moving and the other unit is stopped:

1. If we're the higher-priority unit, and the lower priority unit can get out of the way, calculate our "get-to point" (the point we need to get to in order to be past the collision) and tell the lower-priority unit to move out of our way (Figure 7). Change the collision state to **`resolving`**.
2. Else, if we can avoid the other unit, avoid the other unit and resolve the collision.
3. Else, if we're the higher-priority unit and we can push the lower-priority unit along our path, push the lower priority-unit. Change the collision state to **`resolving`**.
4. Else, stop, repath, and resolve the collision.

Case 4. If we're moving and the other unit is moving:

1. If we're the lower-priority unit, don't do anything.
2. If collision with hard radius overlap is inevitable and we're the higher-priority unit, tell the lower-priority unit to pause, and go to Case 3.
3. Else, if we're the higher-priority unit, calculate our get-to point and tell the lower-priority unit to slow down enough to avoid the collision.

RESOLVING COLLISIONS.

- If we're the unit that's moving in order to resolve a Case 1 collision and we've reached our desired point, resolve the collision.
- If we're the Case 3.1 lower-priority unit and the higher- priority unit has passed its get-to point, start returning to the previous position and resolve the collision.
- If we're the Case 3.1 higher-priority unit, wait (slow down or stop) until the lower-priority unit has gotten out of the way, then continue.
- If we're the Case 3.3 higher-priority unit and the lower-priority unit can now get out of the way, go to Case 3.1.
- If we're the Case 4.3 lower-priority unit and the higher-priority unit has passed its get-to point, resume normal speed and resolve the collision.

One of the key components of coordinated unit movement is to prioritize and resolve disputes. Without a solid, well-defined priority system, you're likely to see units doing a merry-go-round dance as each demands that the other move out of its way; no one unit has the ability to say no to a demand. The priority system also has to take the collision severity into account. A simple heuristic is to take the highest-priority hard collision and resolve down through all of the other hard collisions before considering any soft collisions.

If the hard collisions are far enough in the future, though, you might want to spend some time resolving more immediate soft collisions.

Depending on the game, the resolution mechanism might also need to scale based on unit density. If a huge melee battle is creating several compound hard collisions between some swordsmen, you're better served spending your CPU time resolving all of those combat collisions than resolving a soft collision between two of your resource gatherers on a distant area of the map. An added bonus to tracking these areas of high collision density is that you can influence the pathfinding of other units away from those areas.

Basic Planning

Planning is a key element of unit cooperation. All of these predictions and calculations should be as accurate as possible. Inevitably, though, things will go wrong. One of the biggest mistakes we made with the AGE OF EMPIRES' movement was to make every decision within a single frame of reference. Every decision was always made correctly, but we didn't track that information into future updates. As a result, we ended up with units that would make a decision, encounter a problem during the execution of that decision, and then make a decision that sent them right back on their original path, only to start the whole cycle over again the next update.

Planning fixes this tautology. We keep around the old, resolved collisions long enough (defined by some game-specific heuristic) so that we can reference them should we get into a predicament in the future. When we execute an avoidance, for example, we remember what object it is that we're avoiding. Because we'll have created a viable resolution plan, there's no reason to do collision checking with the other unit in the collision unless one of the units gets a new order or some other drastic change takes place. Once we're done with the avoidance maneuver, we can resume normal collision checking with the other unit. As you'll see next month, we'll reuse this planning concept over and over again to accomplish our goals.

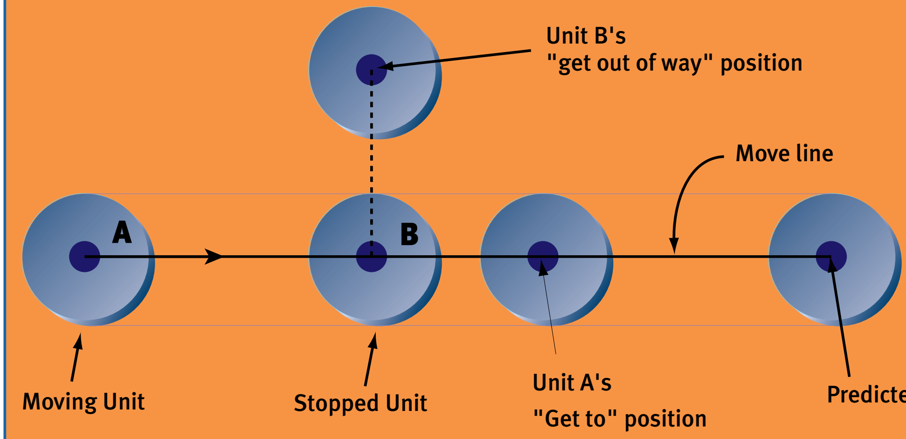
Simple games are a thing of the past; so is simple movement. We've covered the basic components necessary for creating a solid, extensible movement system: a state-based movement algorithm, a scalable collision determination system, and a fast position prediction system. All of these components work together to create a deterministic plan for collision resolution.

Next month, we'll extend these concepts to cover higher-order movement topics, such as group movement, full-blown formation movement, and compound collision resolution. I'll also go into more detail about some implementation specifics that help solve some of the classic movement problems. ■

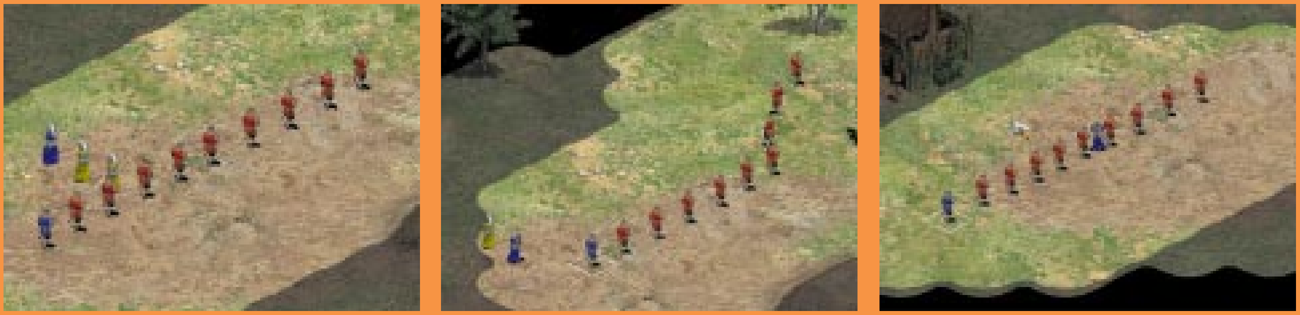
FOR FURTHER INFO

- Take a look at Craig W. Reynolds' Boids work at http://hmt.com/cwr/boids.html.
- Steven Woodcock's Game AI web site is located at http://www.cris.com/~swoodcoc/ai.html.
- Also see Patrick Winston. *Artificial Intelligence*, 3rd ed. (Addison-Wesley, 1993.)

FIGURE 7. *Resolving a collision between a moving unit and a stopped unit.*



1. Find collision
2. Calculate position for Unit B to move to in order to get out of Unit A's way.
3. Calculate position for Unit A to get to in order to be past the collision. Once Unit A reaches that point, have Unit B move back to its original spot.



Back

S c ed E ce i Ha dli g

by Bruce Dawson

When TOTAL ANNIHILATION (TA) was released in September 1997, most of those who played it found it to be an enjoyable and robust game. However, despite our best testing efforts here at Cavedog Entertainment prior to release, some of our customers reported crashing bugs. One of my tasks at the company was to track down the cause of these crashes.

It quickly became apparent that this was going to be a tricky problem. A disquieting number of the bug reports looked something like this: "I'd been playing multiplayer TA for two hours when the game suddenly exited back to the desktop without displaying an error dialog." For some reason the standard Windows 95 crash reporting dialog wasn't appearing, so I had absolutely no evidence as to where in the million bytes of code the crash was occurring.

The Solution

Shortly after I started my task, a random meeting with a member of the DirectX team gave me a crucial clue. If a program crashes while DirectDraw has the screen locked, then any attempt to bring up the error dialog would result in a deadlock. I found that the DirectX team had put in a structured exception handler that would suppress the error dialog when necessary to avoid deadlocks after crashes. Clearly, TA was running afoul of this error dialog suppression.

Once I knew what the problem was, the solution became clear. The best way to fight a structured exception handler is with another structured exception handler. All I had to do was wrap each of TA's threads in a structured exception handler that would record the error information to a file — then it wouldn't matter whether or not the error dialog came up.

The Tedious Details

Some of you are probably wondering what a structured exception handler is. Structured exception handling is a Win32 feature that allows programs to set up a list of routines that should be called when something bad happens. This process is very similar to C++ exception handling, with two important differences. First, structured exception handlers are given a data structure that contains important information, such as the contents of the CPU registers when the error happened. This information is vital to what we'll be doing. Another difference — one that we won't be making use of — is that under certain circumstances, a structured exception handler can actually restart the faulting instruction after fixing whatever caused the error.

This article isn't concerned with the dirty details of how structured exception handling works. You can use it very effectively — including writing handlers — without understanding the internal details. You just have to trust that the black magic works. See the "For Further Information" section for details.

If you have a Windows program with an entry point at `WinMain()`, it's trivial to wrap it in a structured exception handler so that all errors will be caught. Simply rename your `WinMain()` function as `HandledWinMain()` and insert the code from Listing 1.

Your entire program executes inside the `__try` block. The function in the `__except` block gets called if your program (or your main thread, at least) crashes. So now all you need is a `RecordExceptionInfo()` function (`GetExceptionInformation()` is a system function). The basic idea behind the `RecordExceptionInfo()` function is trivial. All you have to do is open a file, write out the register information from the `PEXCEPTION_POINTERS` structure, then return `EXCEPTION_CONTINUE_SEARCH`, a magic value that tells Win32 to proceed with its normal error handling mechanism. This step is important

Bruce Dawson is senior software engineer at Cavedog Entertainment. He can be reached at comments@cygnus-software.com.

because it means that an error dialog will pop up if possible and your debugger will happily coexist with the structured exception handler.

As usual, the devil is in the details. You need to make sure that your exception handler will work in unusual circumstances. Because a crash can happen at any time, the C run time may be in a bad state, so you have to try to avoid using it. If the exception was an FPU exception, then you can't use the FPU or you'll trigger the exception again. Finally, you might as well make use of the exception handler to record additional information about the environment in which the crash happened — information that might be invaluable in tracking down the problem. The exception handler supplied with this article (you can download the .ZIP archive from *Game Developer*'s web site) uses a number of unusual tricks to accomplish this, and should run on any Win32 x86 system.

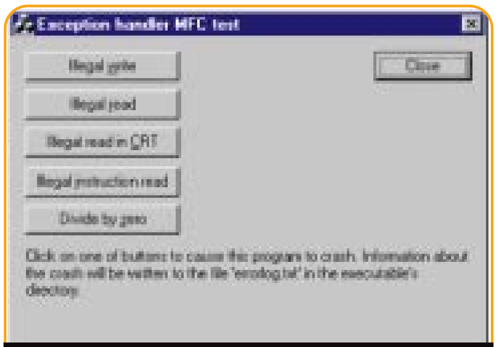
The Win9x error dialog has a reasonable layout, so I followed its example where possible for the error log file, making additions as needed. For instance, the sixteen words of stack needed to be increased, so that as much of the local variables and the call stack as possible will be saved. Locating the bottom of the stack is easy — it's just the ESP register. The location of the top of the stack isn't as obvious, but it is fairly easy to find. It's stored in the task information block, four bytes into the **fs** segment, and you can get it with these two lines of assembly language:

```
mov eax, fs:[4]
mov StackTop,eax
```

I use **VirtualQuery** and **GetModuleFileName** to scan through memory looking for loaded .DLLs, which makes use of the fact that a .DLL's module handle is nothing more than its load address. For each .DLL, I print out enough information to identify it uniquely, so that I can verify what versions were being used. Finally, to allow easy formatted printing without the C run time, I wrote a formatted printing routine that uses **wvsprintf** and **WriteFile**, both Win32 functions.

Test Drive — Seeing It in Action

The exception handler, available on the *Game Developer*'s web site, comes with three sample applications to demonstrate it in action. These are a console application, a WinMain application, and an MFC application. The MFC application requires a different technique to hook in the exception handler; see the EXCEPTIONATTACHER.CPP file for instructions on how to do it. The MFC application is the easiest with which to experiment. Just click on one of the crash buttons, then look at the ERRORLOG.TXT file to see what gets recorded.



Click on one of buttons to cause the program to crash. Information about the crash will be written to the file 'errorlog.txt' in the executable's directory.

FIGURE 1. *The MFC Exception handler.*

Interpreting the Results: The Easy Stuff

The sheer volume of information that the exception handler produces (see Listing 2 for an abbreviated example) in response to a crash can be daunting. It's not uncommon for programmers to feel that the only change with the exception handler installed is that instead of having a small amount of indecipherable information, they now have a large amount of indecipherable information. This is a genuine concern — some of the information that the exception handler dumps is quite cryptic. However, there are two thoughts that should give you solace. One is that if you ever run across a critical crash that relies on that extra information, you'll probably figure out how to interpret it. The other is that a lot of the information in the error logs is actually quite easy to interpret, once you learn how.

Perhaps the most critical piece of information that the error log gives you is a complete description of what executable and .DLLs caused the crash. Whether you're dealing with QA or with customers, this knowledge will avoid a huge number of mistakes because you'll frequently be able to tell them, "Oh, well of course it crashes. You're running Tuesday's version." For each .DLL and for the executable, the error log lists several pieces of vital information. The full file name of the code module, its file size, the time stamp of its file, and — most importantly and unambiguously — the link time stamp. The link time stamp is vital because, though you may end up with multiple executables that are the same size, and while file times can be altered by installers and time zones, the link time stamp is stored in the file at link time and is always unique and unaltered. The exception handler could easily be modified to record version information, but I haven't yet found that necessary.

LISTING 1. *Wrapping your Windows program's entry point in a structured exception handler.*

```
int __cdecl RecordExceptionInfo(PEXCEPTION_POINTERS data, const char *Message);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
          LPTSTR lpCmdLine, int nCmdShow)
{
          int Result = -1;
          __try
{
Result = HandledWinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow);
          }
          __except(RecordExceptionInfo(GetExceptionInformation(), "main thread"))
          {
// Do nothing here.
}
          return Result;
}
```

Another critical piece of information that the error log tells you about each code module is where it was loaded in memory. This information allows you to interpret addresses in the stack as possible return addresses, and it allows you to adjust your bug interpreting if .DLLs are relocated when loaded.

The "Bytes at CS: EIP" field, a hex dump of the instructions that caused the crash and a part of the standard system error dialog, is useful as a final check that the correct code was being executed. If you load the offending executable into the debugger and notice that the code bytes don't match, then you're loading the wrong executable or the .DLLs don't match. Alternately, if just a few bits are different, it's possible that you've just found the cause of the crash, and identified it as a hardware problem. Code segments are write-protected in Windows 95, so they can't be corrupted by stray pointers. Therefore, changed code bytes generally suggest an error in the memory, disk, or processor. While tracking down TA crashes, I've come across two crashes that were definitely such hardware failures, and three others that seemed likely.

System details, such as the processor type, time of crash, and the amount of memory, can be helpful in detecting unusual configurations that could be causing the problem. There is huge potential for recording additional information, such as DirectX driver versions, video card types, and so on. You could even save your data section and all of your allocated game data. If you record too much information, you may run into privacy concerns from some users, but as long as you don't try automatically mailing the crash log to yourself, you should be safe.

Interpreting the Results: The Hard Stuff

Finally, we get to the hardest part of interpreting the results. If your customer isn't running the beta version of your game on a 386 with 4MB of faulty memory, then you have to start considering the possibility that your precious game might have a bug. In an ideal world, the games that we released to the public would have no bugs, sound drivers would never misbehave, and it would only rain at night. However, it's a fact of life that no matter how carefully you write your games, they will always crash occasionally — not because of the code that you write of course, but because everybody has careless coworkers. When these bugs cannot be reproduced, and leave no evidence as to where they occurred, you can easily have an expensive problem.

As always, "Be prepared" is a good motto. Whenever you send out a version of your game, whether it's to QA or to manufacturing, you need to make sure that you've kept a copy of everything you need to aid in bug analysis. Basically, this means that you need to archive the released files, the debug information, and the source.

With your debugging information handy, load your copy of the executable that crashed into the debugger by selecting it from the file dialog. You want your debugger to load your executable into memory and create a process, but not start running your program. You do this by selecting the Single Step or Step Into command. Once all the symbols have loaded, open up a register display window and click on the EIP register to place the insertion point there. Now type in the EIP value from the error log file, and the debugger should take you to the code that crashed. By carefully examining the assembly language, the relevant source code, the register contents from the error log, and the stack dump from the error log, you will frequently be able to track down the cause of crashes — even crashes that only happen on machines on the other side of the continent.

LISTING 2. *Some of the information returned by the exception handler.*

```
mfctest caused an Access Violation in module mfctest.exe at 001b:00402178.
Exception handler called in Main Thread.
Error occurred at 10/18/1998 22:28:46.
D:\bsrc\exceptionhandler\mfctest\Release\mfctest.exe, run by bruce.
1 processor(s), type 586.
96 MBytes physical memory.
Write to location 0 caused an access violation.

Registers:
EAX=00000000 CS=001b EIP=00402178 EFLGS=00010286
EBX=00000001 SS=0023 ESP=0012fa00 EBP=0012fa18
ECX=ffffff00 DS=0023 ESI=004033a8 FS=0038
EDX=00000000 ES=0023 EDI=0012fe70 GS=0000
Bytes at CS:EIP:
88 08 eb 47 8b 55 fc 8a 02 88 45 f8 eb 3d 33 ff
Stack dump:
0012fa00: 0012fe70 004033a8 ffffffff 000003e8 0012fe00 00000000 0012fa34 0040160f
...

         Module list: names, addresses, sizes, time stamps and file times:
D:\bsrc\exceptionhandler\mfctest\Release\mfctest.exe, loaded at 0x00400000 - 18432 bytes -
362ab0f3 - file date is 10/18/1998 22:24:38
C:\WINDOWS\System32\MFC42.DLL, loaded at 0x5f400000 - 954640 bytes - 345ef14a - file date
is 5/11/1998 18:19:00
```

Success

The crashes in TA were eventually tracked down to a few memory buffer overruns and a buggy sound driver. Because the only machine we could get these crashes to happen on was in New York, I spent many hours poring over error logs and piecing together pieces of the puzzle. Without the exception handler to reliably record the critical data, I might still be puzzling over why TA sometimes crashed... instead of playing it. ■

FOR FURTHER INFO

- Anything by Matt Pietrek, especially the January 1997, May 1996, April 1997, and May 1998 editions of his "Under the Hood" column in *Microsoft Systems Journal*. These are available at http://www.microsoft.com/msj/ or on MSDN.
- Windows NT/95/98 developers should consult the WINNT.H file, which describes the `EXCEPTION_POINTERS` structure.

S e i g he Digi al Vide La d ca e

by Ben Waggoner

Video in games has gone from an impossibility, to the holy grail, to the source of all evil, to a common, appropriate component in most games. Video is done badly in many cases, resulting in poor quality and requiring more work than necessary. This article looks at video compression and playback technologies currently being used in games, some upcoming options, and some ways for game developers to achieve better results with less headache.

Playback Resolution

Let's begin by briefly going over some important topics that are pertinent to the use of video in games. First, we'll look at the issue of playback resolution. Video doesn't need to play back at the same resolution as the rest of the content. For example, if the game engine runs at 1024×768, but cut scenes only play by themselves in full-screen mode, it makes sense to drop the resolution of the monitor down to 640×480 for playback. All the APIs that we'll look at in this article support resolution switching.

Scaling is another technique. Many of today's graphics cards have built-in high-quality hardware scaling; at low data rates, a 320×240 movie scaled up 200 percent can often look a lot better than a native 640×480, as scaling artifacts are generally less objectionable than compression artifacts. In addition, because the video card does the scaling, processor's cycles are saved for running the game itself.

A cheap way to do pixel doubling is to draw only every other line. Duck's TrueMotion and RAD Game Tools' Smacker offer this option, and it's always employed by Eidos Escape. The main advantages are lower video bandwidth and processing power requirements and an interlaced video-like effect that tends to hide video limitations. This last aspect is especially important. The latter installments in the WING COMMANDER game series crammed a large amount of video on each CD-ROM using this technique. Its drawback is that because every other line is black, the image is only half as bright as it otherwise could be.

Frame Rate

Sometimes it's not possible to use the full frame rate of the source due to bandwidth, data, or other concerns. When reducing frame rate, the final frame rate should be evenly divisible by the original frame rate. This restriction is critical to achieving the appearance of smooth motion. Imagine that you have video that was shot at 30 FPS being played back at 20 FPS (so every third frame is dropped). A moving object in the original video would move a certain amount on each frame, but after conversion to 20 FPS, each frame samples one thirtieth of a second or one twentieth of a second, alternating. This can cause an object to move different amounts in alternating frames.

To address this problem, each frame of the final video has to represent an

Ben Waggoner is the chief technologist of Journeyman Digital, a digital media production, post-production, and consulting company serving the interactive entertainment industry. He hates it when good games have bad video, and you don't want Ben hating you. You can reach him at ben@journeyman.com.

Older Cinepak movies were high-quality, but the large files resulted in multi-disc games such as WING COMMANDER IV.

Bungie's MYTH: THE FALLEN LORDS used Smacker from RAD Game Tools for its animated cut scenes.

equal sample of the original motion. In order to achieve equal samples, we divide the frame rate by an integer value. So, a 24 FPS film should reduce into a frame rate of 24, 12, 8, 6, and so on. Video at 30 FPS should go to 30, 15, 10, 7.5, 6, and so on.

Keyframes

The codecs that we'll look at in this article are all interframe codecs, which means that they compress video by comparing differences between frames. Most frames aren't stored as complete images, instead storing only those parts of the frame that differ from the frame before it. The frames that store a complete image are the keyframes; the partial frames are delta frames. Because delta frames are smaller than keyframes, generally the fewer keyframes, the better the compression. Of course, when a video frame changes a lot (such as at the start of a new scene), there is often no advantage to using a delta frame. Most codecs would automatically insert a keyframe if the delta frame was 85 percent of the size of the keyframe or larger.

As a result, when you randomly access a segment of video, you can quickly go to a keyframe, but other frames may not be pulled up so easily. For instance, say you wanted to look at a frame of video that was 100 frames after the previous keyframe. You'd have to decode the keyframe, and then process all 99 delta frames off-screen before you could view the right frame. Needless to say, this process can be a little slow. Also, if a frame is dropped during playback due to some performance hitch, the video will pause on the last decoded frame until it gets to the next keyframe (generally, the audio will keep playing).

In many games, random access to video segments isn't necessary. But if it is, make sure you have a keyframe at the point to which you'd like to skip. This point will often fall at the start of a scene, which as we said before, will often be a keyframe anyway. Sorenson Video allows you to change the keyframe insertion sensitivity, so you can tune appropriately. Also, Adobe Premiere and Media Cleaner Pro let you add markers at certain frames so that upon compression, those frames are automatically made keyframes.

Data Rate vs. Data Size

Now that most machines have 24x or better CD-ROM drives, throughput isn't much of an issue. More important is balancing playback performance, quality, and disc usage. We can easily play back full-screen, full-motion 1MB per second Cinepak movies — but only at ten minutes of video per disc. The result, of course, has been multi-disc fests such as WING COMMANDER III and IV, the six-disc BLACK DAHLIA, and others.

For most games, average data rate is much more important than peak data rate. But most compression tools still default to fixed data rates. We're much better off having the codec vary its data rate with the content, going up in difficult portions and down with easy parts, preserving an even quality. We have some options for varying data rates. Some codecs allow you to set a fixed quality level. For instance, with Indeo 5.06, a quality setting of 85 at 320×240 and 30 FPS will generally yield files around 130K per second, which look better than files compressed at a fixed 150K per second. The problem with this approach is that you can't predict how big the files will be; you may find yourself performing multiple compressions of the video before you can exactly meet your bit budget. Better yet is Sorenson's VBR mode, which lets you select both an average and a peak data rate; it then optimizes quality given those parameters. This same concept is used in MPEG-2 for DVD. We should encourage other vendors to adopt this option as well.

Tools for Video Compression

ADOBE PREMIERE. The traditional tool for compressing video has been Adobe Premiere, which was primarily designed for video editing. However, it does an adequate job of compression, and it's the best tool available for Windows-based platforms. Premiere is capable of outputting QuickTime 3.0 and .AVI files, but there are a few compression issues to consider. First, if you're able to work off of progressive scan source, make sure you turn off

Interplay's Star Trek *franchise used the TrueMotion-S codec.*

field rendering in the Video dialog. If you're using an interlaced source, make sure field rendering is turned on.

You'll find some important settings in the Special tab in the Export Movie dialog. First, make sure that Better Resize is turned on. This option forces Premiere to use high-quality bicubic scaling instead of its default, bad-looking nearest-neighbor method. Bicubic scaling alone can substantially improve video quality. Also, if you're coming off of a video source that has noise or black around the edges, you can select the area to be cropped for the final image.

TERRAN MEDIA CLEANER PRO. A much better tool for compression is Terran's Media Cleaner Pro. Unlike Premiere, it's a dedicated compression tool that's full of great features for doing this type of work. Right now, it's only available for Macintosh and doesn't handle .AVI directly. Terran has promised a Windows version with support for other file formats. But Media Cleaner is so good that it's worth using a Macintosh just so you can preprocess your video with this tool, even if you just make intermediate files and copy them over to a Windows system for compression.

Playback Technologies

There are three main playback architecture technologies used in games. Having grown beyond their original play-video-in-a-window origins, these products now provide complex APIs to implement video in myriad ways. They all have strengths and weaknesses, and choosing which to use depends on the given project needs.

APPLE QUICKTIME. Until last spring, the full authoring API and media-layer technologies for QuickTime were only available for Macintoshes, so its use in games was largely supplanted by other technologies. However, the new 3.0 version offers the same API on both Macintosh and Win32.

QuickTime's biggest advantage is in its ubiquity. QuickTime has robust solutions in every part of the content chain from digitization through editing, effects, compression, and playback. QuickTime 3.0 also includes advanced media-layer functionality, allowing authors to combine video and audio with VR, 3D, music synthesizers, and animated stills in very exciting ways.

QuickTime also goes far beyond simple audio and video playback. It works on a track metaphor, in which each track corresponds to a certain media type. Beyond audio and video tracks are MIDI, QuickDraw 3D animation, video effects, sprites, and even interactive elements. For example, it would be easy to make a QuickTime movie that consisted of a real-time rendered 3D object that is composited on the fly over a playing video, with a digitized vocal track with MIDI accompaniment playing. All these media could be stored in a single .MOV file that would play on any QuickTime 3.0 machine. Addressing the API would make it easy to swap out the 3D model at run time, and dynamically change the MIDI track while it plays. QuickTime is far beyond other systems in offering this kind of functionality, and it verges on being a platform of its own: there are over 7,000 pages of official documentation available for it now. However, using QuickTime needn't be complex. Because the file format can encapsulate all the media integration information, media authors can do the heavy lifting; from an API-level, playback of the most complex movie requires no more coding than for the simplest.

QuickTime's biggest drawback is that it lacks MPEG-1 support on Windows, and has no MPEG-2 support at all. QuickTime also eats up a few MB of RAM while running, which can be a concern for RAM-hungry titles.

QuickTime has long been a staple of multimedia CD-ROM, providing video in such chestnuts as MYST. It has somewhat fallen out of favor with game developers, largely due to the anemic state of the older QuickTime for Windows APIs. This could change with QuickTime 3.0's excellent API, which has a lot to offer even Wintel-only products. It's well worth a look by all developers. The first major game title shipping with QuickTime 3.0 is X-FILES. The new LEGO MINDSTORMS product also uses QuickTime 3.0 heavily in its cut scenes and building instruction sections.

DIRECTSHOW. DirectShow is a descendent of a long line of Microsoft attempts at a QuickTime killer. It started out as Video for Windows, it later became ActiveMovie in the 32-bit world, and is now DirectShow and is a part of DirectX. For a few years in there, it seemed that Microsoft kept rereleasing the same product road map under new names instead of shipping a complete product. Nonetheless, the current DirectShow 2.0 has a lot to recommend it.

DirectShow's target market is similar to QuickTime's — it offers an "everything to everyone" suite of digital video services. DirectShow's main advantage over QuickTime is that it handles MPEG-2 and DVD content well. If your product is targeting computers using DVD playback hardware, DirectShow provides a single API to play the MPEG-2 content, regardless of which MPEG-2 hardware is installed. The installed base of DVD-ROM and MPEG-2 is growing rapidly, and will become increasingly important.

Unfortunately, DirectShow lacks flexibility as a file format and authoring system, and .AVI is showing its age. While you can use the DirectShow API for media-layer functions such as overlaying video with text, it's impossible to make a simple file that holds more than a video and an audio track. While Microsoft does have the "Advanced Authoring Format," at this point it's largely vaporware — it remains to be seen if the AAF will reach critical mass among software vendors. There is a dearth of high-quality compression tools for the DirectShow formats — Adobe Premiere is the best available.

SMACKER. Smacker, from RAD Game Tools, has been the dominant system used in commercial games for the last few years. It was originally designed to play 320×200 full-screen video from DOS games on 486-based Intel machines, it's been under continuous development, and it's more popular than ever in the Win32 DirectX age.

Where QuickTime and DirectShow have breadth, Smacker has depth. Instead of being a general-purpose video architecture, Smacker has unwavering focus on giving game developers exactly what they need in a playback architecture. Smacker doesn't have a capture architecture, 3D support, or third-party tools available for it. All there is the Smacker compressor and the Smacker codec. It supports QuickTime 3.0, .AVI, and other formats for input.

Smacker's main advantage is its incredible performance and flexibility for game creators. It's not just used for cut scenes, but throughout programs for simple animation. For example, all the in-game animations in TOTAL ANNIHILATION are Smacker-based. The Smacker SDK is very flexible.

Smacker's main limitation is that it's 8-bit only. Yes, in our high-color world, Smacker only does 8-bit graphics. This limitation is less severe than you might think, and is partly responsible for Smacker's excellent performance. Smacker has a very good palette optimization system, and if the video will be played in a high-color environment, you can switch palettes at will (they recommend every 10 frames). However, this won't help if you have a lot of continuous tone imagery in a single frame. Smacker is best suited to rendered video or live video with limited colors.

RAD Game Tools has a 24-bit follow up to Smacker in progress, called Bink. Bink should offer all of Smacker's focus on game developers, but finally with great true-color support (see the sidebar, "Upcoming Codecs").

The Smacker tools are a free download, or $95 with support. Player and Scriptor applications without credit screens and the Director Xtra cost $995. Licenses cost $3,000 per product, with volume and site licenses available. Smacker costs $1,000 for the SDK and $3,000/title for playback distribution.

Odds are good that any game you can think of uses Smacker. A random smattering of titles include INTERSTATE 76, STARCRAFT, DUNGEON KEEPER, MYTH: THE FALLEN LORDS, and BLACK DAHLIA.

Video Codecs

As important as the architectures are the codecs. Codecs (compressors/decompressors) are plug-ins to the different architectures that do the actual video processing. Both QuickTime and DirectShow have some built-in codecs, with a number of third-party codecs available as well. In many cases, a given codec can be available for both architectures.

TRUEMOTION. Duck's TrueMotion family of codecs has been one of the most successful codecs in games to date, both in terms of titles produced and revenue. There have been three main products in its lifetime: TrueMotion-S, TrueMotion RT, and TrueMotion 2.0.

TrueMotion-S was an intraframe-only codec with limited quality but excellent playback performance. Its main selling point was the ability to play back reasonable-quality full-screen video on a variety of platforms. It was used in a number of products, most notably Interplay's *Star Trek* franchise. However, wide-scale acceptance was hampered by difficult tools and unpredictable data rates.

Duck decided to produce a new interframe codec that would maintain the good playback of TrueMotion-S, while providing higher quality at lower data rates. This product is TrueMotion 2.0 ($395), currently available for DirectShow and, hopefully by the time you read this, QuickTime. Duck sells TrueMotion 2.0 directly, and still charges both for tools and a per-product licensing fee. The company also sells an SDK for TruePlay, which is a high-performance proprietary playback environment.

TruePlay is used by a number of major game developers such as Activision and Electronic Arts. Given its quality and performance, TrueMotion 2.0 is widely expected to become the dominant for-sale codec for higher data-rate CD-ROM and DVD-ROM content. A decompressor is also built into Microsoft's DirectX from version 5.1 on, a boon for developers who don't need the playback performance of TruePlay.

The TrueMotion 2.0 Compression Toolkit is $395. The TruePlay 6.0 SDK is $5,000. Non-retail kiosk and high-volume application licenses are $1,000 per product. Retail kiosk and application licenses are $2,500 per product. The TruePlay license costs $5,000 per product. Note that TrueMotion playback is built into DirectShow.

Major happenings are on the horizon for TrueMotion. A new version, currently called TrueMotion 2x, should be out in a few months (see the sidebar, "Upcoming Codecs"). Even in the early development version I've worked with, it offers substantially improved quality and compression. Sega has licensed TrueMotion for Dreamcast, so we can expect to see even more TrueMotion in the future.

The first major game using TrueMotion 2.0 and TruePlay was FINAL FANTASY VII. Games using older versions of TrueMotion include PHANTASMAGORIA: A PUZZLE OF FLESH and most of the *Star Trek* titles.

SORENSON VIDEO. Sorenson Vision's Sorenson Video codec was one of the big pushes in the rollout of QuickTime 3.0, and rightly so. Derived from a video conferencing application for the deaf, it excels in preserving important motion at low data rates.

Apple and Sorenson struck a deal by which Apple included a full decoder into the basic QuickTime 3.0 package, in exchange for also including a limited version of the compressor. The deal worked well: QuickTime 3.0 has been downloaded over one million times since its release, Sorenson-compressed videos are becoming common on the Internet, and the Sorenson Developer Edition has been selling briskly at $499. The Developer Edition adds a number of additional configuration options, such as keyframe insertion threshold, and (when run from Media Cleaner Pro) Variable Bit Rate (VBR) encoding.

Video in X-FILES is based on Apple's new QuickTime 3.0.

Final Fantasy VII's cut scenes were based on TrueMotion 2.0.

While Sorenson has been pushed primarily as a video solution for the Web, it has a lot to offer games in which disc space is at a premium. Unlike TrueMotion or Smacker, Sorenson can produce good results with many kinds of content at 50-100K per second. If you want to use Sorenson, I highly recommend that you do it with Media Cleaner Pro and the Developer edition. The VBR produces dramatically better results. In essence, you tell the compressor the average data rate at which you want a clip to play back, and what the highest peak data rate your target platform can achieve during playback. From this input, the compressor figures out how best to distribute bandwidth throughout the clip for maximum quality. There's a lot of depth and subtlety to Sorenson.

Sorenson's major drawback is its performance. It is slow to encode, especially in its VBR mode. It also requires a fast Pentium II-class computer to play back anything more than 320×240 movies. However, it has a very high-quality scaling mode, so pixel doubling can produce very good results. For game use, it's probably best to encode at 320×240, and then scale up to the desired size. Lastly, Sorenson uses YUV-9 color space, which causes trouble with highly saturated colors.

The Basic version of Sorenson Video in included free in QuickTime 3.0. The Developer version of the codec is $499. No games currently use Sorenson, but I think it's a great technology in cases where disc space is at a premium.

EIDOS ESCAPE. Eidos Technologies, sister company to the publisher of TOMB RAIDER, developed a codec for use in its games. Happy with the results, Eidos took it to market as Eidos Escape, available separately or with the Eidos Escape VideoStudio.

One of the more charitable descriptions I've heard of Eidos Escape was, "Just like TrueMotion 2.0, but worse." Escape's two main problems are that it's only 16-bit (causing terrible banding with low-contrast images) and that it has no data rate control beyond a quality slider. Escape has been a bomb in the marketplace, and I can't think of any problem for which it is the ideal solution. The only titles I'm aware of that use Escape are the TOMB RAIDER series and other Eidos Interactive titles.

CINEPAK. Cinepak was the original CD-ROM video codec. Created by SuperMac and acquired by Radius, it was licensed to virtually every digital video platform, including QuickTime, Video for Windows, 3DO, Sega Saturn, Sega CD game systems, and now in the Java Media Layer. Free to end users, its combination of good overall performance and great price made it the dominant codec in multimedia until last year. Unless a game is targeting older machines, or wants to eschew any license cost, Cinepak doesn't have much to recommend it anymore.

This didn't have to be true. Given the enormous audience for Cinepak content, Radius developed a Pro version of Cinepak, demonstrating a nearly final version at NAB '95. However, Radius fell apart financially, and development of Cinepak effectively ceased. Several former Radius employees formed Compression Technologies Inc. and licensed back the work-in-progress from Radius. Initial hopes for a quick release and rapid enhancement proved vain, however. CTI released a $499 package of the new codec and its own compression tool. While it's capable of impressive results for projects needing Cinepak delivery, Cinepak Pro has consistently shipped late and been technically troubled; its window in the marketplace has largely closed. While apparently still available as a commercial product, it is Macintosh-only, unstable, and barely marketed.

Cinepak is included for free in DirectShow and QuickTime. Cinepak Pro costs $499 for a single-user license.

INDEO. Intel's Indeo line is unique in that it was never intended to make money. Early versions were attempts to grow the multimedia industry, and thus sell more Intel processors. In an era where Macintosh dominated digital video, it was also a way to encourage users to use Intel's video capture and video playback cards in Windows-based machines. In the early days of CD-ROM digital video, the Indeo 3.2 codec was second only to Cinepak in use, and had strong advantages for talking-head content. When Pentium machines began to come out, Intel wanted a new codec that would demonstrate the new processor's power, while still being usable on older machines. They redesigned Indeo 4.0 from the ground up, using a scalable wavelet-based compression technology.

Even though Indeo has been continually improved and is free, its use has dropped precipitously in recent years. This inattention may be mainly due to a lack of marketing — presumably Intel didn't want to spend marketing resources on a giveaway technology.

Indeo has a number of special features of interest to game developers. It supports a one-bit alpha channel, and can do good real-time compositing with playback. This is great for achieving a number of game effects, such as compositing an actor over a run-time generated background in real time.

The Indeo API can do local decoding of a region of a movie. For instance, imagine a submarine game in which the user looks through a periscope. One could render a 1,024×192 Indeo movie, representing a panoramic image around the sub. The Indeo codec could then decode only the 192×192 region that the periscope is looking at, dramatically saving processing time.

Indeo, using its wavelet compression system, can also do scalable playback. With scalability turned on, when there is insufficient processor power to decode the video, the codec can dynamically drop quality instead of dropping frames. This provides a much better experience for the end user.

There are two caveats with Indeo. First, it's processor intensive. Only the fastest Pentium II machines can play back a 640×480 Indeo movie. Second, it uses YUV-9 color space. With YUV-9, the codec stores color information in 4×4 blocks. This means that a sharp edge of a highly saturated color can look extremely blocky. When running under MMX, Indeo automatically smoothes out these blocks, substantially enhancing quality. If your game isn't going to be MMX-only, make sure to test the video quality on a non-MMX machine.

Interestingly, there are currently four available versions of Indeo. The one that you, as a game developer, should use is Indeo 5.06, which offers the best performance, quality, and compression. Indeo 3.2 is an old cross-platform talking-head codec little in use today. Indeo 4.1 is the last version that ran under Win16. Indeo 4.4 is available for .AVI and QuickTime playback under Windows, and it's generally comparable to Indeo 5.06, except for slightly inferior compression and no MMX YUV-9 correction.

As I mentioned, Indeo is free. It hasn't been used much for games to date. The only high-profile title that has used it, to my knowledge, is CIVILIZATION II.

MPEG-1. MPEG-1 is an open-standard digital video format. It was originally intended to run only with hardware acceleration. As with many open standards, MPEG-1's adoption was hampered by different vendors implementing the standard slightly differently. While there was an early '90s push for hardware-assisted MPEG-1–based games, it never reached critical mass. DirectShow now includes a robust MPEG-1 software playback system that yields good results on fast machines. But MPEG-1 doesn't offer nearly as much to the industry today. It's not very flexible and can't go past 352×240 in resolution. A few games still do use MPEG-1, such as Microsoft's CLOSE COMBAT 2. Software-based MPEG-1, especially with full-screen playback, can be quite processor intensive; CLOSE COMBAT 2's video playback was the most system taxing part of the game.

MPEG-1 hardware and software compressors are available from a number of different vendors. My favorite is Heuris-Pulitzer's MPEG Power Professional ($499). MPEG-1 decompression is built into DirectShow and QuickTime for Macintosh (QuickTime for Windows support is forthcoming).

MPEG-2. A high-resolution, high-quality enhancement of MPEG-2 is the dark horse in game video right now. Fast Pentium II systems with good AGP video cards can do a marginal job of MPEG-2 playback in software (without much CPU power left over for anything else). However, with the advent of DVD, an increasing number of computers are being sold with DVD-ROM drives and MPEG-2 playback hardware. As the installed base increases, this could become a tempting video playback option for games. I would expect the installed base of MPEG-2–capable systems is already approaching that of, say, 3Dfx graphics cards (although the demographics of those consumers might differ from the typical game-hungry 3Dfx card owner). Some companies, including Electronic Arts and Sierra, are developing MPEG-2–based products for DVD-ROM users (initially converting older multi-disc titles to single MPEG-2 based DVDs). Electronic Arts is already shipping bundle-only DVD-ROM versions of WING COMMANDER IV and NFL GAME DAY 98, with more in the offing. I expect development of games using DVD-ROM and MPEG-2 to increase over time. ■

Upcoming Codecs

The next few months are going to be an exiting time for codecs. Duck's TrueMotion 2X and RAD Game Tools' Bink are very promising enhancements. I've been working with an alpha of Bink and a beta of TrueMotion 2X, and I'm impressed with both.

Bink finally brings Smacker users into the true-color world. It will work pretty much as Smacker does, with similar tools, APIs, and so on. Although it's 24-bit instead of 8-bit per pixel, data rates are much less than three times as much. For some kinds of content, with lots of continuous tone gradients, I expect Bink will be able to produce better quality than Smacker, but at the same data rates. Bink's pricing model should be very similar to Smacker's, although it might cost slightly more.

Duck's Truemotion 2X, even in early beta, promises to be a major enhancement. Originally planed as a 2.x upgrade to 2.0, the development went so well Duck redubbed it 2X, as a new product. The biggest enhancements are seen in data rate control and compression quality, historically Truemotion's biggest weaknesses. The codec's final version should also offer meaningful enhancements in motion quality. Dreamcast developers should definitely be paying attention to 2X, as it will be built into the Dreamcast development environment.

FOR FURTHER INFO

Adobe Premiere:
San Jose, Calif.
(408) 536-6000
http://www.adobe.com

DirectShow:
Microsoft Corp
Redmond, Wash.
(425) 882-8080
http://msdn.microsoft.com/developer/sdk/dxmedia.htm

Eidos Escape:
Eidos Technologies Ltd.
Wimbledon, London, U.K.
+44 (181) 636 3000
http://www.eidostechnologies.com

Indeo:
Intel Corp.
Santa Clara, Calif.
(408) 765-8080
http://developer.intel.com/pc-supp/multimed/indeo/overview.htm

QuickTime:
Apple Computers Inc.
Cupertino, Calif.
(408) 996-1010
http://www.apple.com/quicktime/index.html

Smacker:
RAD Game Tools Inc.
Kirkland, Wash.
(425) 893-4300
http://www.radgametools.com

Sorenson Video:
Sorenson Vision Inc.
Logan, Utah
(888) 767-3676
http://www.s-vision.com

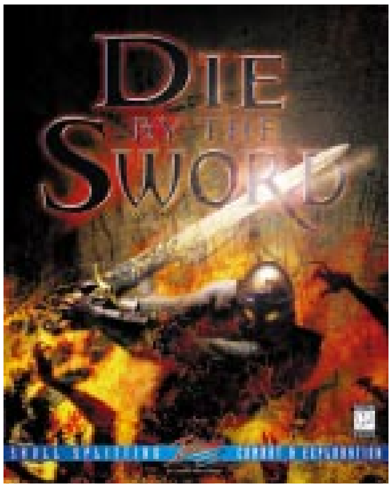
Terran Media Cleaner
Terran Interactive
Los Gatos, Calif.
(408) 356-7373
http://www.terran-int.com

Back

Teach Die By The Sword

by Jamie Fristrom

When Peter Akemann and Don Likeness (a.k.a. The Brains) first moved to the mountain town of Wrightwood, Calif., to prototype the then-unnamed DIE BY THE SWORD, they imagined the game as a sort of BILESTOAD 3D. They wanted to create a fighting game that wasn't just real-time rock, paper, scissors, but an actual sword-fighting simulation, where players could target exactly where their swords went and physics would take care of the rest. And, similar to BILESTOAD, when players did enough damage to their



opponents' limbs, said limbs would come right off. Pete and Don had connections at Interplay who introduced them to Brian Fargo; they showed Brian the prototype, signed a contract, moved to Venice, Calif., and started building a development team.

Jamie Fristrom's girlfriend keeps trying to assure him that he's not a geek. This article is non-fiction about computer geeks; to see some of Jamie's fiction about computer geeks, go to http://id.mind.net/~fristrom.

The Team

It took a while for a good team to stabilize; we tried working with out-of-house contract artists to no avail. Then Chris Busse, an old friend whom we'd hired as a producer, started experimenting with LightWave and ended up as our first 3D artist. And then, by stroke of luck, we hired Chris Soares, our art director, who'd previously worked for Microprose (he created the cover of X-COM); Chris Soares entirely turned around the look of the game. The rest of our team included myself, a coder whose biggest previous title had been DELUXE JEOPARDY FEATURING VANNA WHITE; Mark Nau, a frustrated ex-Atari game designer; Charles Tolman, a programmer who did the GRETSKY port for the N64; and Tomo Moriwaki, an art student whom we originally hired as a flunky but who ended up as our other level designer. And that was it. An eight-person team, most of whom had been friends of The Brains in high school or college. We were a very democratic bunch; although Don acted as our fearless leader for the first half of the project (partly because Pete took some time off to finish his Ph.D.) and Pete for the second half, we usually did things by consensus — if just one team member didn't like something, we'd argue until that person was convinced, or we'd figure out something better.

Interplay's sound department did almost all of the sounds, hired and recorded the voice actors, and composed the music. Also, Interplay gave us an animator and a production assistant during the last crunch. Although the sound and music sometimes came in frustratingly late, the animator was a production machine.

Slippage

Depending on who you talk to here at Treyarch, the project schedule slipped anywhere between three and five times. Technically, if you go by the contracts with Interplay, it slipped three times, but there were also periods of time between contract renegotiations where we had some verbal agreement on a ship date that later changed.

The first slip was due to the realization that our out-of-house artists weren't going to produce models of acceptable quality on time. So we asked for more time, hired an in-house guy, and reassigned Chris Busse to the art department.

The second slip was due to a slight design drift. Nobody really thought a fighting game for the PC would sell, and we'd been dropping hints about how we'd like our next project to be a sort of PRINCE OF PERSIA, but in 3D. (Believe it or not, we hadn't seen TOMB RAIDER yet.) So Interplay asked for some proof of concept. Charles came up with the idea of portals (we called them scissors, back then) and coded a camera that could stay inside caves. Pete developed the rope trap to prove that physics really was an asset to a computer game. I took an event system that we'd originally built to control our demo at E3 and turned it into a scripting language. We finished our proof-of-concept demo on the morning that we had show it to the executives at Interplay, and they loved it.

The last time slip was due to the fact that the game just needed more polish. We used this last slip as an opportunity to update our lighting model, which was no good compared to the competition.

The DIE BY THE SWORD team: (left to right) author Jamie Fristrom coded much of the game and wrote the scripting language; Don Likeness, the project's cofounder and sometime team leader, built the renderer; Tomo Moriwaki modeled many of the game's levels; Dr. Peter Akemann, cofounder and team leader, built VSIM, the game's engine; Mark Nau designed the game's levels; Charles Tolman shared coding responsibilities with the author; Chris Soares, art director, modeled and painted the game's characters and levels; and Chris Busse, in addition to level-building responsibilites, served as DIE BY THE SWORD's producer.

Despite the setbacks, we did finally manage to ship a product, and even though it came out over a year and a half later than we had originally intended, it was still current: *Next Generation* gave it five out of five stars and *Computer Gaming World* called it "a fine game indeed."

Our Tools

The prototype was originally a 16-bit DOS program written in BorlandC++. Before we showed the prototype to Interplay, The Brains took it to 32 bits with the Borland DOS extender. ("No sleep 'til 32!" was their rallying cry.)

DIE BY THE SWORD

Treyarch LLC
Culver City, Calif.
(310) 670-4111
http://www.treyarch.com
http://www.diebythesword.com

Team Size: Eight full-time developers at Treyarch. A handful of assistants at Interplay.
Release date: March 1998.
Budget: Approximately $1,000,000 overall.
Time in development: Two and a half years.
Intended platform: Windows 95
Typical workstation at the beginning of the project: 90Mhz Pentium with 16MB RAM.
Typical workstation at the end of the project: 200Mhz Pentium II with 32MB RAM. Artists had 64MB.
Critical software: Visual C++ 4.2, CoolEdit, Paint Shop Pro, Image Robot, Photoshop, SourceSafe, LightWave, 3D Studio MAX, and Premiere
Notable technology: Treyarch's VSIM engine

Borland's DOS extender wasn't adequate, so we switched to Watcom. Then, when we started doing dual development (DOS and Windows 95), we switched to Visual C++ 4. We used C++'s extended features heavily, including templates, exception handling, and some standard template libraries (STL). (I remember a conversation with Jon Blossom at a CGDC in which he asked if we were using STL. I said, "No, I mean, we've pretty much internalized how to do a linked list." The very next week I wrote a linked list with a stupid bug. The next day I switched to STL.) We used Microosoft Assembler for the software rasterizer. Although we tried Starbase's Versions for source control, we quickly switched to Microsoft SourceSafe, because of shortcomings in Versions.

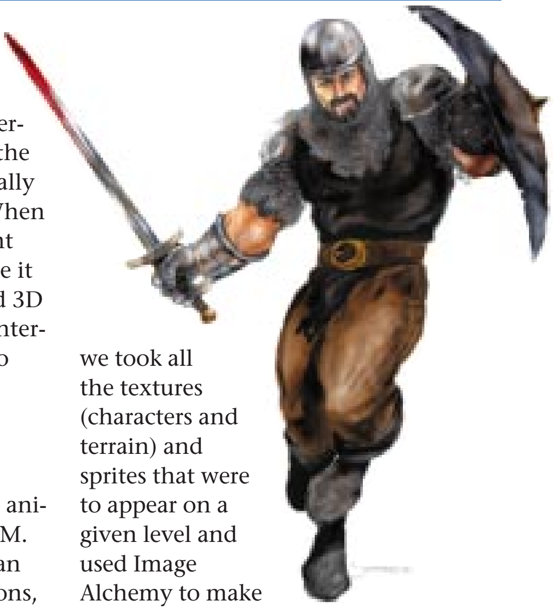
When we started, we had two art-houses, one building in LightWave and the other in PowerAnimator. Because of the differences in texture-mapping between the two programs, the game could actually handle two different file formats. When we got rid of the art-houses, we went with LightWave exclusively, because it was much cheaper. Still, Soares used 3D Studio MAX to make fire and user-interface decorations. We used 3D Studio MAX and Adobe Premiere to do the post-processing on our cut-scenes. CoolEdit was our favorite utility for managing sounds.

Pete built our physics and human animation system, which we called VSIM. VSIM is an animation system that can be used for walking, swinging weapons, reaching for door handles, looking at opponents, and hanging upside down from ropes. Most of the combat animation, and a lot of the simple gestures that the characters made, were created using VSIM in the in-house Move Editor. The Move Editor allowed us to apply motion targets to a character's arms; a left-to-right sword swing was a target on the left followed by a target on the right. We thought the Move Editor was so clever, we decided to ship a slick, polished version of it with the game. "Create your own custom moves" was the marketing bullet. Unfortunately, it was a rarely used feature, as most people felt it was more effective to control Enric on the fly than to store up prerecorded moves for him. A great in-house tool can be a very mediocre feature.

The Art

As I mentioned before, Chris Soares' art direction made a world of difference to the look of Die By The Sword. A number of his tricks and techniques contributed greatly to the success of the game as great-looking product.

Cool palette tricks. In Die By The Sword, Chris Soares came up with a great method for dealing with palette restrictions. Our artists did all their painting in true color. Then



we took all the textures (characters and terrain) and sprites that were to appear on a given level and used Image Alchemy to make an optimized palette for that art. Then we'd remap all the textures for that level to that palette. So, for different levels, character models use different texture maps that are optimized for each specific level's palette. This method resulted in much more subtle coloration than we could have achieved by choosing one palette at the beginning of the game and then picking all of our colors from it.

Still, this method did have its limitations; a palette is still a limited set of colors, and you can't have a level with a wide variety of dramatically different colors without losing your nice gradients.

Consistent art design. One thing that Chris Soares insisted on after we hired him was that the game have a consistent look — and that doesn't just mean dark fantasy all the way through. For example, one place where the consistency is most apparent is in the user-interface shell screens: every menu or dialog box that pops up is a translucent dark banner that has a bracket of weapons, skulls, and other dark fantasy accessories. And most of the banners animate.

Character models. Originally, the characters were made up of rigid polygonal hulls for every limb segment; the sword was a model, the forearm a model, the upper-arm a model, the torso a model, and so on. This construction looked fairly ugly (especially when the twisting torso turned on the pelvis) and caused some overdraw. We changed the models so that matching vertices on different limbs could be attached, and would stretch when the limbs moved, creating the appearance of a single-mesh that didn't overdraw.

What Went Right

A project that slips as often as ours often suffers from the STONEKEEP phenomenon: the game sounds intriguing when you first see the PR for it, but by the time it finally hits the shelves, it's out of date. We managed to avoid becoming another STONEKEEP. This is how I think we did it:

1. We only employed excellent people. In his *Decline and Fall of the American Programmer* (Yourdon Press Computing, 1993), Edward Yourdon says this tenet is key to the success of a development project. But it doesn't apply to just programmers; it applies to every member of the team: artists, producers, level designers. To achieve this goal, we practiced nepotism: we hired friends from college who had some experience in the game industry and we knew we could trust. Also, we got lucky a couple of times. But a couple of times, we ended up with mediocre people whom we had to let go. Maybe in a project with a larger team, we would have had to figure out how to work with mediocre people. DIE BY THE SWORD wasn't that project.

2. C++ was our friend. When Pete and Don hired me, I began a campaign of whining about proper object-oriented design. This whining turned out to be productive, because well-designed programs are capable of more evolution than poorly-designed ones. And as it turned out, DIE BY THE SWORD needed to evolve.

Layers between the game and the hardware allowed us to port from DOS to Windows without coughing up our lungs. And when the game evolved from a fighting game to a TOMB RAIDER-style dungeon crawl, it was solid object-oriented design that allowed this evolution. Also, Interplay always wanted new features; object-oriented design enabled the development of a lot of those features.

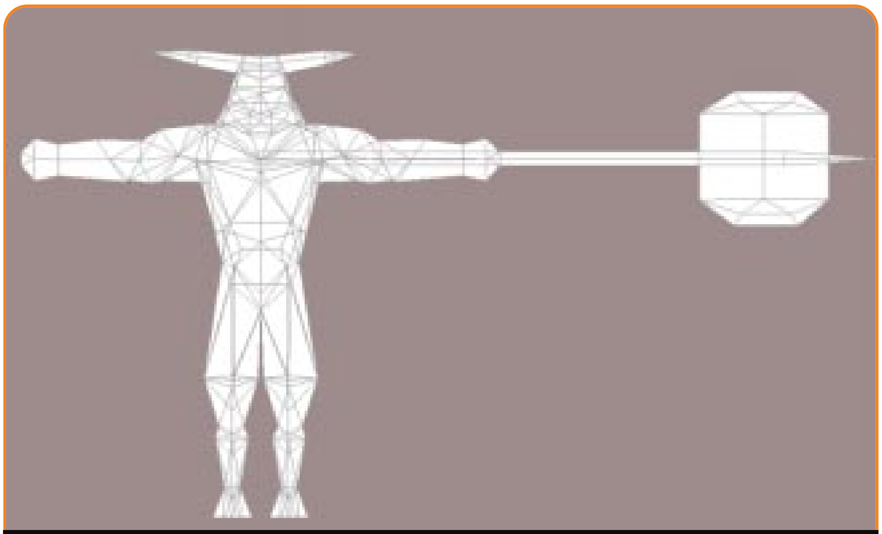
3. Our scripting language. We had an event-driven, and in a sense multithreaded, scripting language that facilitated Tomo and Mark's work immensely. I was leery of scripting languages because I was used to working on projects with mediocre production assistants who either couldn't figure out the language or wouldn't step up to take advantage of all its features. Mark was a programmer in his own right, however, and Tomo was just a hell of a smart guy.

This scripting language is my favorite example of how object-oriented code can evolve to meet new needs. Originally, there was just the `event` class, a class intended to time camera changes and some AI mode switching for an E3 demo. But these events soon became something a level designer could enter from a text file, and from there they evolved into a scripting language that created chains of complex behavior.

This scripting language wasn't as powerful as it could have been if it were written as a virtual stack machine with a real compiler; it's still better to do things correctly from the start than to evolve the system you need incrementally. But you should design well so that when you need to add a feature that you never dreamed of, you can do it. Steve McConnell, the author of *Code Complete* (Microsoft Press, 1993) pointed out that changing requirements are just a fact of life: "Maybe you think the Edsel was the greatest car ever made, belong to the Flat Earth Society, and vote for George McGovern every four years. If you do, go ahead and believe that requirements won't change on your projects."

The scripting language turned into an extremely quick way to get a new feature into the game and turn it over to Mark and Tomo for fine-tuning. It also gave the game its distinctive personality, the feeling that you're in an actual world that's going on without you, not just a world where you walk from room to room and kill whatever's behind the next door. For example, the room in which you come across two orcs in the process of beating on a kobold hanging from a rope until you open the door and engage them…

We even used the scripting language to make our between-level cut-scenes: Mark would direct the actors with it,

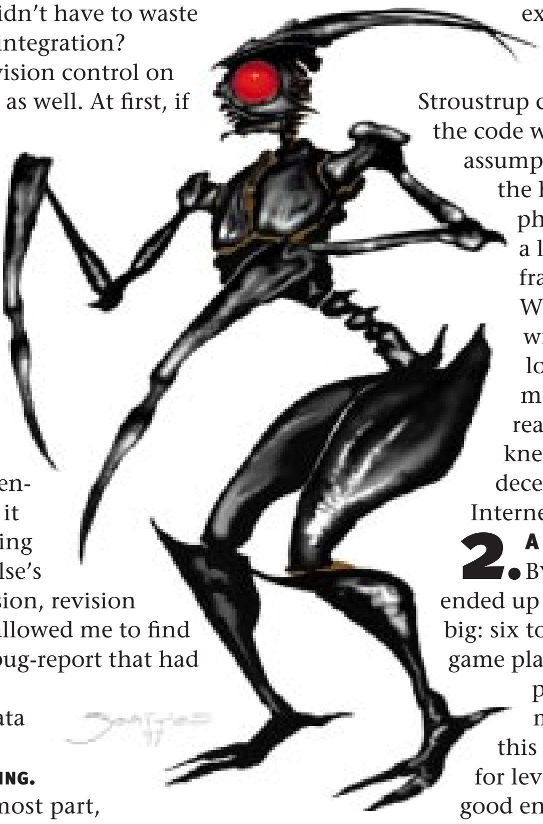

The Minotaur character, in wireframe.

then shoot movies which we converted to Interplay's proprietary movie-playing format.

4. Source control was our friend. We weren't using source control for the first couple of months of the project, and when we finally started, it was so nice that we wondered what took us so long. We learned to use concurrent checkouts without hesitation or fear.A few times over the life of the project, some routines and modifications were mysteriously lost, but for the most part everything worked beautifully.

I read in a previous issue of *Game Developer* that the guys at id don't use revision control (Brian Hook, "How I Spent my Summer Vacation, or What I Learned While Working on QUAKE II," January 1998). I think they're insane. How much better would QUAKE II have been if they didn't have to waste time on code integration?

We used revision control on the game data as well. At first, if you wanted a file fixed, you had to tell the person who "owned" it to fix it for you. Revision control empowered anyone to fix a problem whenever he found it without stepping on someone else's toes. On occasion, revision control even allowed me to find a bug from a bug-report that had disappeared because the data had changed.



5. Bug hunting. For the most part, we fixed the bugs before adding features. This regimen is recommended in Steve Maguire's *Debugging the Development Process* (Microsoft Press, 1994) and was my motto from day one at the company. This is an easy procedure to forget when your milestone is approaching and your publisher just wants to see features; sometimes, we'd put off bug-fixing until after a milestone release. But in general, we fixed the bugs as soon as possible. So, in the final release, there was only one serious bug we had to patch: although you could bring custom moves with you into the arena portion of the game, you couldn't bring them into the quest.

What Went Wrong with Our Development? Where Do I Start?

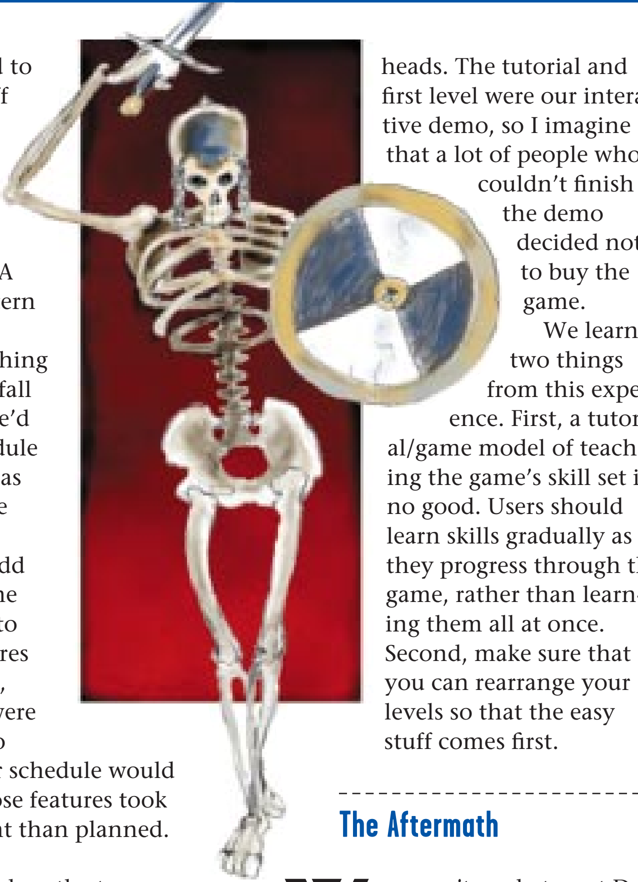
1. Not all of the code was well-designed. Unfortunately, the prototype that The Brains wrote three years ago and showed to Interplay was Pete's first C++ project. Some parts, such as the human motion engine, suffered from a nearly complete absence of even the basic principles of good code, and other parts of the project suffered from wacky C++ experiments that would have made Bjarne Stroustrup cringe. Because the code was laced with assumptions, deep in the heart of the physics engine was a locked down 18 frames per second. We were stuck with a frame-locked net-play model for similar reasons, which we knew wouldn't be decent over the Internet.

2. A short game. DIE BY THE SWORD ended up not being very big: six to eight hours of game play in the quest portion. The main reason for this is that our tools for level design weren't good enough. This had two effects: we had to get rid of artists who couldn't adapt to our tools, and the artists we did keep took more time to make a level than they would have needed if the tools were better. "Scissoring" a level, or dividing it into sectors, was a laborious process. The scripting language had no facilities for debugging, and its error messages were often quite unenlightening. Furthermore, the game could take minutes to load a single level. Our level designers would typically make a change to a script that took ten seconds to type in, and then read a magazine while the level loaded. It often took longer than the coder's compiles. Unfortunately, the programmers were adding new lame features, such as the tournament, rather than writing better tools to facilitate the level designers.

On our next project, our producer will be there to facilitate the programmers and the programmers will be there to facilitate the level designers. When a level designer gets jammed due to a bad programmer, the programmer will be given forty lashes.

3. Lighting. We experimented with shadow maps in the middle of the project and discarded them because we felt the memory was better used making higher-resolution, unshadowed textures. Chris Soares repeatedly asked, "Are you sure?" because if his maps weren't going to be shaded, he was going to paint the shadows right into them. We told him to go ahead. Unfortunately, the end result wasn't as good, not because the textures were badly painted, but because there still wasn't enough memory. Next, we tried vertex shading, but that meant that Chris Soares had to go through and undo all the shading he'd put into the textures, because everything was coming out too dark. Also, it was too slow in software to run on our benchmark

machine. Users had to turn the shading off for a decent frame rate, so they ended up with unshaded, washed-out terrains.

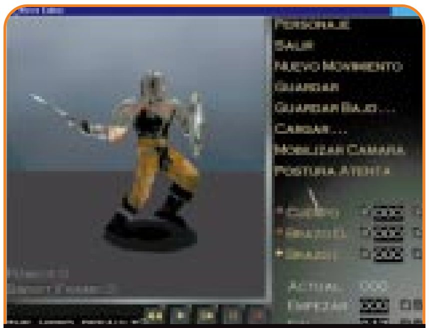


4. FEATURE CREEP. A recurring pattern when our schedule slipped went something like this: we would fall behind schedule; we'd agree that the schedule would have to slip; as long as the schedule was slipping, it was quite tempting to add new features into the schedule. We tried to say no to new features as much as possible, but some of them were just too tempting to resist. And then our schedule would slip again when those features took longer to implement than planned. And so on.

Some features, such as the tournament mode, seem a total waste of time in hindsight. But other features, such as the improved lighting model, were probably essential to the game's success. We did manage to ship eventually, but Interplay tells us we didn't hit our marketing window (whatever that is.)

5. STEEP LEARNING CURVE. The game ended up being too hard to play, and a lot of newcomers got stuck in the tutorial, where there were some jumping puzzles that were almost as hard as jumps in the middle of the game. Also, the first monsters you meet are so short that the default attack goes over their heads. The tutorial and first level were our interactive demo, so I imagine that a lot of people who couldn't finish the demo decided not to buy the game.

We learned two things from this experience. First, a tutorial/game model of teaching the game's skill set is no good. Users should learn skills gradually as they progress through the game, rather than learning them all at once. Second, make sure that you can rearrange your levels so that the easy stuff comes first.



The final version of the Move Editor, which shipped with the retail version of the game. This screenshot happens to be from the Spanish version.

The Aftermath

We weren't ready to put DIE BY THE SWORD away once we'd shipped. No, a two-and-a-half-year habit is much too hard to break. Besides, we had issues to resolve.

MAKING THE PATCH. We always knew that our game would be awful over the Internet, partly because 250 milliseconds of latency makes a swordfighting game impossible in the first place and partly because we were stuck with a frame-locked model from the beginning. Still, people tried it over the Internet, and those with cable modems actually had some success.

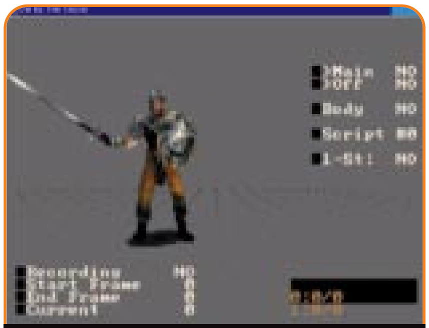
The abysmal Internet play was the largest complaint on the bulletin boards, even though Interplay's marketing department was wise enough to say nothing about Internet play on the box. As a result, most of the patch dealt with getting a frame-locked Internet game tolerable. We were able to optimize the game to a relatively acceptable level. If both players were on modems, it was playable (at something like 5 frames per second); if one had a cable modem, it was okay (maybe 11 frames per second); and if both had cable modems, it was decent.

MAKING THE EXPANSION PACK. We managed to convince Interplay that a $20 expansion pack would be a worthwhile addition to the game. Unfortunately, we then had to develop it, when we would have preferred to start creating our new engine. The whole process went quite smoothly. Pete gave up the producer reins entirely to Chris Busse, who managed everything. The expanision pack hit the shelves in the beginning of October 1998.

DIE BY THE SWORD JAPANESE. Out of nowhere, Interplay asked us about the feasibility of doing the game in Japanese. We had impressed Interplay by being the first game in the history of their company to ship in America and Europe at the same time, but no one had said anything about Japan until now. Chuck Tolman handled the port almost exclusively, converting strings into double-byte format and finding ways to fit ridiculous amounts of kanji onto the screen. And because the Move Editor shared so much code with the main executable, once he got that working, the Move Editor took less than a day to get into the new language. Our next product will be Japanese-ready from the start.

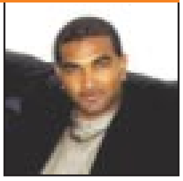
What Have We Learned

Unless you're willing to cut features, your game is probably going to slip. And when it slips, you're going to need to add features to stay current. The only way to do that, on the programming side, is to take the hit from day one, always doing things correctly instead of doing things quickly. Sometimes something as simple as a poor choice of variable name will be a thorn in your side until the end of the project. ■



Treyarch's in-house Move Editor.

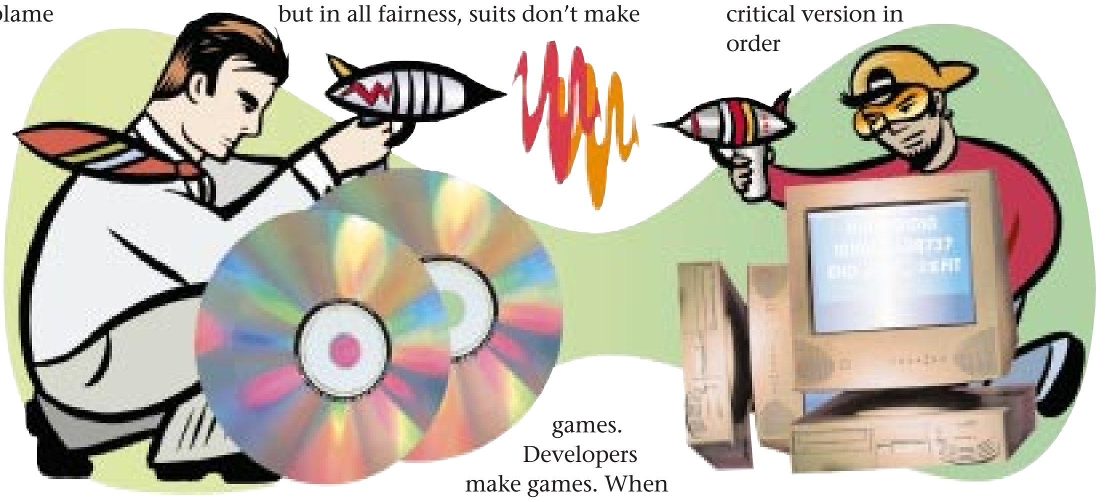
by Derek Smart



It's Ready When it Ships

There's a war going on out there. It's a fragfest that involves: (a) publishers shipping games before they're ready, (b) developers taking flak for games released incomplete, (c) retailers putting these duds on the shelves, (d) some members of the press giving inaccurate reviews and fueling the hype of games they know are released incomplete, and (e) game players who purchase said duds and then attempt to blame someone.

If the current trend of low-cost titles as well as high-end ones finding their way into bargain bins within weeks of shipping is any indication, one can assume that the monetary clout of game players will win, and that this practice of shipping incomplete games will cease. Right? Wrong. It's going to take a lot more than flame wars, lawsuits, and foul language to resolve this problem. So, what the heck is going on out there? Below, is my take on the situation.



The Usual Suspects

The publisher. Your typical outfit is run by suits who ignore the fact that releasing a game incomplete is just plain fraudulent. Whether it's an innovative title or the next great shovelware, the game players are the last factors considered in a bean-counting exercise that peers at the bottom-line sales figures. No matter how long a game is in development, there will come a time when it will go in a box, even if it's not ready. Publishers are blamed for this mishap, but in all fairness, suits don't make games. Developers make games. When a development team schedules a game to be completed in 18 months and takes large advances from publishers, what recourse does the publisher have, when 24 months and a bloated budget later, the game is still not ready? Where can a publisher draw the line between shipping an incomplete product and recouping at least a portion of its investment? In an industry where the release date of a title makes a world of difference, most games are subject to the Christmas Syndrome.

The developer. A producer from the publisher is usually tasked with ensuring that the publisher's interests are protected and that the team and game are on track. No respectable development team plans to ship an incomplete game. There are a variety of situations that cause this problem as a result of games slipping and going over budget. It's a problem that is prevalent, and one which is not as easy to resolve as one would think. In-house developers have milestones and a regular paycheck. If they slip, heads roll, and the suits say "Ship it." All the developer can do at that point is ask, "When?" An outside team that is paid on milestones will find itself in a bind if a game slips because it can't deliver that critical version in order to collect pay for that milestone. At the end of the day, what will the publisher do when they've invested millions of dollars in a title, and its schedule slips? Give the team more time and money? Fire the team? Cancel the project? Most times, the team is either given more time or the game ships incomplete. In some cases, team members leave under suspicious circumstances only to resurface elsewhere and repeat the same mistakes. When a game slips, the developer loses leverage with the publisher, and unless there's a bargaining chip tucked away somewhere, the game is going to ship if the publisher says so.

Continued on page 71.

Derek Smart is the designer/lead developer of BATTLECRUISER 3000AD, one of the most disastrous publisher-induced game-release-mishaps since OUTPOST. Outgunned, outnumbered, and unabashed, Smart is behind enemy lines doing what he does for fun: making industry deadbeats and anti-social misfits very, very mad. You can flame him at dsmart@3000ad.com.

Continued from page 72.

Developers are tasked with accurately scheduling their projects, but no matter how accurate they are, Murphy's law always enters the picture.

The Court Martial Candidates

THE PRESS. If this war were regulated, some members of the press would be on trial for treason — if you assume that they are on the side of the game player. The publishers need the press to get the word out on their games. The press need the publishers' ad dollars, as well as the game players' subscriptions. Most game players factor what they read into their buying decisions. Gaming magazines with years of experience and reader loyalty find themselves in the combat zone, trying to cater to both the publishers and the game players. Playing both sides against the middle is common practice for some press types. Certain members of the media will trash a bad game that everyone knows is bad, and hope that the next time they give a high rating to a truly bad game (from a publisher withad purchases) that no-one will notice. Take a wild guess at what happens when a magazine assigns a sports game columnist to review a flight sim — an inaccurate review. With the number of game release blunders, grossly inaccurate reviews and previews, and hype without substance on the rise, game players have a hard time figuring out who to trust. What happens when a game is hyped for years based on countless reviews of previews, beta versions, and so on,.and then game bombs because (a) it was plain bad or (b) it was released incomplete? The game player will find someone to blame, and will often forget that the press is at fault for generating inaccurate hype in the first place.

THE RETAILER. It isn't surprising that several publishers are either going out of business or peddling shovelware in order to make up for losses in big budget games. The retailing tactics amount to something short of outright extortion. It is becoming more and more difficult to put games on the shelves, especially for small publishers and independent developers who want to self-publish. There was a time when retailers made their money on gaming software from profit margins. Nowadays, they rely on big budget publisher-funded marketing campaigns. Therefore, fewer games make it to the shelves. Those that do make it don't last very long because the retailer expects price reductions in order to move them and make way for the next best thing. If the retailer paid low cost for non-returnable inventory, this means that the retailer is stuck with the inventory and has to shovel it. When disgruntled game players return dud games, you would expect that action to ultimately affect the retailer-publisher relationship, right? And that if the retailer didn't sell duds, that publishers would cease producing them. Wrong. As long as publishers continue to pay retailers to put bad games on the shelves, this cycle will continue. The retailers must be thinking that if the publishers are going to peddle duds on their shelves, they may as well pay for the forthcoming downtime that those returns are going to cause.

Casualties of War

THE GAME PLAYER. Meet the warrior clan: the casual gamer, the hobbyist, and the anti-social misfit. With the exception of the third caste, who only use game-release mishaps as an excuse to behave badly online, this clan just wants to play games right out of the box. When a game player has to search the Internet for a patch that makes a new game playable, you know somebody is going to take the flak for it. What is a game player to do when a favorite publisher releases a game incomplete? The game player can start a war or threaten a boycott, but this practice has been going on for years and incomplete games are still the number one complaint among game players. The solution is not as easy as buying another game. If the game player is locked into a genre, for example, a flight jock who has never played QUAKE, then it compounds the problem. Publisher brand recognition is not what it used to be, but the damage is already done.In a choice between RED BARON II and FLYING CORPS GOLD, how many flight jocks relied on the Sierra name brand and ignored a better alternative in FLYING CORPS GOLD? Or games go only halfway. UNREAL is an awesome single-player game, but it has dismal Internet play. So, we all continue playing our favorite QUAKE II mods online. There isn't too much flak about UNREAL because we have the QUAKE alternative.

Let me give an example from personal experience. When my game, Battlecruiser 3000AD, was released incomplete by my publishers in 1996, there were no alternatives in that genre. So, the fallout was catastrophic and it's still going on to this day. There aren't that many choices, and when eagerly awaited titles don't live up to the hype or are released in a dismal fashion, it alienates the game players and places the developers under siege.

Developers, Publishers, Retailers, Press, Game Players . . . FIGHT!!

The Quality Assurance guys are seasoned vets who are paid to play games and break them. Period. However, there is always the possibility that a game will ship with known or unknown bugs. This is one reason why patches appear online before ten people have even bought the game. A time comes when a decision has to be made. Publishers assume they don't have much choice when development schedules slip and there are millions of dollars at stake. Most often, they will ship the game incomplete, recoup some of their money, and hope that the game is good enough for game players to hang on to while the development team continues to work feverishly in order to fix the bugs and complete the game.

The advancements in technology which create the hardware and software cocktails prevalent in today's systems ensure that somewhere out there, a bug is waiting to surface. This, in itself, is quite different from shipping a game with known bugs that should've been fixed prior to release. I predict that in the coming months, especially with the ULTIMA ONLINE lawsuit, we will start to see game disclaimers the size of legal documents. Epic has already started the trend with their inbox notice acknowledging that there may be unforeseen problems in the boxed version of UNREAL. This is a step in the right direction, but one which I feel will be misused by some publishers as yet another loophole to shovel duds. One thing is certain, the war will continue to rage on. ■